Magiszter

Számítástechnikai szerkesztőség

szoftver IBM PC,XT,AT és kompatibilis mikroszámítógépekre

MATEX

programcsomag a TEX szedőprogram magyar nyelvű használatához

Programvédelem nélkül!



FADGYAS—MIKLÓS

MATEX

programcsomag a TEX magyar nyelvű használatához

5977/89

193643

FADGYAS TIBOR—MIKLÓS DEZSŐ

MATEX

programcsomag a TEX magyar nyelvű használatához

PC TEX 1.1 változattól

Micro TEX 1.5 változattól

MTA
KIK
0 00006 47098 2

Akadémiai Kiadó, Budapest 1988

885647

Szoftver-dokumentáció
Sorozatszerkesztő: Pintér Tibor
A kiadványt szakmailag ellenőrizte: Körömi Ferenc, Juhász Lehel

© Akadémiai Kiadó és Nyomda Vállalat
Fadgyas Tibor—Miklós Dezső, 1988
Minden jog fenntartva

MAGYAR
TUDOMÁNYOS AKADÉMIA
KÖNYVTÁRA

A szoftver-dokumentáció MATEX programcsomaggal készült
© Akadémiai Kiadó és Nyomda Vállalat · Fadgyas—Miklós, 1988
© TEX az Amerikai Matematikai Társulat védjegye
© PCTEX a Personal TEX Vállalat védjegye
© Micro TEX az Addison Wesley Publishing Company védjegye

M. TUD. AKADÉMIA KÖNYVTÁRA
Könyvtári szám 109 63 /19 88

LEVÉL A SZEDŐHÖZ

Kedves barátom. Ne csodálkozzék ezen a váratlan és —nem tagadom— szokatlan fordulaton, hogy a lusta és szeszélyes író megzavarja önt, derék, pontos munkást, munkája teljesítése közben, és levelekkel zaklatja. Higgye el, érett megfontolás után írom e sorokat, vigyázva formára és tartalomra, egy rezignált megismerés hajnalán, mely egészen új világításban mutatta meg nekem az irodalom egész jelentőségét. Látja, kérem, kedves, ismeretlen barátom, ma reggel olvastam a lapokat, háborúról, meteorológiáról, bajokról, és igen elkeseredtem és eltűnődtem. Az írás mégis csak általában ilyen levélforma valami, az ember, akinek eszébe jut valami, közli egy másik emberrel vagy több más emberekkel, hogy mi jutott az eszébe. Évek óta csinálom már magam is ezt a levelezést, néhány elképzelt emberhez intézve, bőségesen beszámolva mindenről, ami közben eszembe jutott. Sokféle formában írtam levelet, hol első személyben, hol képzelt alakokat szólaltatva meg, de hálátlan és reménytelen dolog ez, látja, az „olvasó", akihez az irodalom évszázadokon át írja szerelmes leveleit, ez a gőgös, rejtélyes hölgy — nem felel. Pedig hányszor szóltam hozzá, sürgetően, figyelmeztetve, lágyan és szigorúan. Látja, kérem, kedves szedő barátom, magának elpanaszolom: már a múlt évben, mikor ezek a háborús dolgok készülődtek, ugye, maga emlékszik, maga az egyetlen ember, aki emlékszik rá, és akire hivatkozhatom, már akkor megírtam, mi a véleményem nekem ezekről a dolgokról, hogy milyen hibásnak tartom az ilyen oktalan öldöklést, hogy milyen egyszerű volna abbahagyni, és hogy remélem, ezek után belátják, milyen meggondolatlanok voltak. Maga emlékszik a levelemre, maga szedte ki —na, és mit gondol, feleltek? Fenét. Nemcsak hogy nem feleltek az én okos beszédemre, hanem, mint ma olvasom, megint elölről kezdik a verekedést, mintha egy sort se kaptak volna tőlem.

Látja, kedves barátom, nem érdemes. Az olvasó nem felel az én bölcs, békéltető, elintéző és elsimító levelemre, az olvasó talán el se olvassa — mit tudom én? —, az olvasó nem hallgat rám, tovább csinál bajokat, háborúba keveredik egymással, összevész a feleségével, beugrik a Dunába, meggyilkolja a felebarátját, handabandázik, rendetlenséget csinál. Hát mi ez, nem kapta meg a levelemet, amiben figyelmeztettem, hogy ne tegyen ilyeneket, mert baj lesz belőle? Vagy megkapta és nem olvasta el? Talán nem is tud olvasni az olvasó.

Kedves szedő úr, látja, rajtunk nem múlik. Én szépen megírtam és maga szépen kiszedte. De hátha az olvasó nem olvasta el! Eltűnődtem ezen, és eszembe jutott, milyen bolond dolgokat csináltam én eddig. Írtam, írtam a leveleket egy ismeretlen valakihez, aki talán nincsen is, vagy nem akar tudni rólam. Aztán eszembe jutott, hogy egyetlen ember van, aki egészen bizonyosan elolvassa, el kell hogy olvassa, amit én írok: és ez ön, szedő úr, az egyetlen ember, akiben biztos lehetek. A régi írók így szóltak: „nyájas olvasó". Milyen képzelgés! Hátha nincs nyájas olvasó? Ellenben szedő, az biztosan van, az fáradságos, keserves munkával bizonyítja be nekem, hogy első betűtől az utolsóig figyelmesen elolvasta az írásomat, nem hagyott ki egy szót se, elfogadta úgy, ahogy van, és kiszedte. Én bizony sohase írom ezentúl, hogy „gondold meg, nyájas olvasó", vagy „hidd el, olvasó", vagy „mit szólsz ehhez, olvasó", hanem őszinte és becsületes leszek és szavaimat a szedőhöz intézem, akkor nem történhetik meg velem megint ilyen blamázs, hogy leírom a háborút és mégis kitör.

Hát bizony, kedves szedő úr, így vagyunk. Magunk között vagyunk, beszélhetünk egészen őszintén. Így még kényelmesebb is nekem, szabadabb és fesztelenebb lehetek, megírhatom a véleményemet mindenről, elpletykálgatunk, ugye, háborúról, időről, meg a többi íróról, szegényekről, akik még mindig írogatják a szerelmes leveleket az ismeretlen olvasóhoz, hol lelkesen, hol fenyegetően, hol gúnyosan, hol allegórikusan; sorok közé rejtve mondanivalóikat — sorok közé! Micsoda képzelődés, maga legjobban tudja, szedő úr, hogy a sorok között spácium van, egy kis vasléc, ami az ólomdarabkákat széttartja egymástól, és nem mondanivaló. Istenem, bár ott tartanánk már, hogy a sorokat elolvassák, nem hogy a sorok között is találnának valamit.

Na mindegy. Hogy van, szedő úr? Hát mit szól ezekhez a csúnya balkán dolgokhoz? Ugye, utálatos? Már igazán nem tudom mi lesz belőle. És mit szól ehhez az időhöz, nyár közepén télikabát. Vigyázzon kedves egészségére, az esték igen hűvösek. És kérem, szedje levelemet figyelmesen, hogy lássam, elolvasta. Isten vele, nyájas szedő.

Karinthy Frigyes

Tartalomjegyzék

Előszó **11**

1. Bevezetés **15**
1.1. Számítógépes szövegelőállítás 15
1.1.1. A TEX müködése vázlatosan 17
1.2. A TEX-hez kapcsolodó file-ok, programok 20
1.2.1. A TEX TFM file-jai 20
1.3. DVI file-ok 22
1.3.1. A betűk közötti távolság 22
1.3.2. Ligatúra 23
1.3.3. Kurziv kiegyenlítés 23
1.3.4. Sorkizárás 24
1.3.5. Lapokra tördelés 24
1.4. PXL file-ok 25
1.5. Makrók, makrócsomagok, formátumfile-ok 29

2. TEXnikai alapozás **31**
2.1. A TEX által használt betűk és jelek 31
2.2. TEX parancsok 36
2.2.1. Parancsjelek 36
2.2.2. Parancsszavak 38
2.3. Környezetek kijelölése 43
2.4. Paraméterek 46
2.5. Számlálók 52
2.6. Betűkészletek (fontok) megadása 53
2.7 Néhány szó a dobozokról 56

3. Az első TEX szöveg előállítása **61**
3.1. Az első lépések 61
3.2. Az első fut(tat)ás 69
3.3. Az első pofonok 73

4. Hogyan tördel a TEX? **77**
4.1. Hogyan adjuk meg az oldal formáját? 77
4.1.1. Az oldal paraméretei 77
4.1.2. Oldalszámozás, élőfej és lábsor 79
4.2. Hogyan tördeli a TEX a sorokat? 82
4.2.1. Az algoritmus 83
4.2.2. Hogyan befolyásolhatjuk az elválasztást? 87
4.2.3. Hogyan befolyásolhatjuk a soremelést? 91
4.2.4. Speciális sorok 94
4.3. Hogyan tördeli a TEX a bekezdéseket? 98
4.3.1. Sortávolság 98
4.3.2. Lapemelés 102
4.4. A TEX különböző módjai 106

5. Matematikai szövegek szedése **111**
5.1. Indexek, kitevők 119
5.2. Emeletes kifejezések 122
5.3. Szumma és integráljelek 124
5.4. Változó nagyságú matematikai jelek 128
5.5. Matematikai szövegek szedési stílusai 135
5.6. Mátrixok szedése 136
5.7. Több kifejezésből álló elrendezések 139
5.8. Matematikai függvények 143

6. Táblázatok, ábrák **147**
6.1. Dobozok 147
6.2. Táblázatok készítése 152
6.2.1. Állandó szélességű oszlopk 152
6.2.2. Bonyolultabb táblázatok 155
6.3. Ábrák 162
6.4 Lábjegyzet 165

7. Új utasítások definiálása 169
7.1. Mik azok a makrók? 169
7.2. A legegyszerűbb definiciók 171
7.2.1. A definiciók megadása 172
7.2.2. Az utasítás neve 174
7.3. Változók megadása 178
7.4. Egyéb definiáló utasítások 182
7.4.1. Utasítások definiálása más módon 182
7.4.2. Új paraméterek, mértékek és dobozok definiálása 184
7.5. Feltételes utasítások 186
7.6. Lábsor és élőfej könyvekben 191
7.7 Lábjegyzet automatikus számozása 194

8. Hibajavítás 197
8.1. A hiba megtalálása 197
8.2. A hiba kijavítása 201
8.3. A leggyakoribb hibák 206
8.4. Bonyolultabb hibák megtalálása 213

9. Makrócsomagok 219
9.1. Általános tudnivalók 219
9.2. A plain és mplain formátumok 222
9.3. A LaTeX formátum 223
9.4. A MaTeX programcsomag 225
9.5. Az AMSTeX és VANILLA makrócsomag 231
9.5.1. Az AMSTeX matematikai jeleiről 232
9.5.2. Egyéb parancsok 233
9.6. Az AMSSPT stílus 236

Függelék

A. TeX programok a gyakorlatban 239
A.1. A különböző TeX implementációk 240
A.1.1. Az `mplain` implementálása 240
A.1.2. A PCTeX programcsomag 241
A.1.3. A MicroTeX programcsomag 246
A.2. A nyomtató programok 251

A.2.1. A PCTEX-hez árult meghajtók 253
A.2.2. A MicroTEX-hel árult meghajtók 255
A.2.3. A Preview program 258

B. A MATEXprogramcsomag installálása 265
B.1. A MATEX lemez tartalma 265
B.2. Az installáláshoz szükséges programok 266
B.3. A billentyűzet átdefiniálása 267
B.4 A TFM file-ok konvertálása 270
B.5 Az MPLAIN formátum installálása 272
B.6 A PXL file-ok konvertálása 273
B.6 A MAPIX program 276

C. Fonttáblázatok 279

D. TEX parancsok listája 289

Bibliográfia 295

Tárgymutató 297

Előszó

Több próbálkozás után született meg végül ez a könyv, amely a TEX szövegszedő program magyar leírása. Ennél azonban sok tekintetben több funkciót is betölt. Talán ez Magyarországon az első könyv — de mindenképpen az első könyvek egyike — amely a fejlettebb országokban már elterjedt ún. Desktop Publishing, vagy röviden DTP számítógépes felhasználásról szól. A Desktop Publishing szó (a számítástechnikában már szinte szokásossá vált módon) csak körülírható, de magyarra le nem fordítható. Szó szerinti jelentése íróasztalon végzett könyvkiadást jelent, azaz olyan programok gyüjtőneveként használják, amelyek lehetővé teszik a könyvkiadás leginkább technikai jellegű részének, a szedésnek mikroszámítógépen való, tehát akár az otthoni íróasztalnál történő elvégzését. Magyarországon leggyakrabban kiadványszerkesztő programoknak nevezik a Desktop Publishing szoftvereket.

Mivel egy ilyen program leírására vállalkoztunk, számos kifejezést, sőt talán egy kicsit bizonyos gondolkodásmódot is honosítanunk kellett. Valószínűleg csak nálunk szokatlan még — számítástechnikusok és szedők számára egyaránt — hogy egy számítástechnikai könyv rengeteg nyomdai kifejezéssel és szakszóval tarkítva, lényegében a szedésről és a tipográfiáról szól. Ennek, tudjuk, óhatatlan következménye, hogy egyik szakember sem fogja magáénak érezni. A szerzők érzése szerint mégis inkább a számítástechnikusok értik majd meg jobban, márcsak azért is, mert a szerzők maguk is közelebb állnak ehhez a témához. De — szinte biztos — előbb-utóbb ki fog alakulni a tipográfusok egy olyan nemzedéke, akik már számítógépeken — köztük mikroszámítógépeken — is képesek lesznek (legalább részben) végezni munkájukat.

A könyvben használt elnevezések is jobban kedveznek a számítástechnikusoknak. Ezen a szakterületen ugyanis rengeteg olyan fogalom van, amit nem lehet egyszerűen leírni, és nincs is rá igazán jó magyar elnevezés. Gon-

doljunk csak a DOS nevére. Ezzel szemben mindenki tudja, hogy mi az a kisbetű. Ezért — elnézést kérve a tipográfusoktól, amiért az ő terminológiájukat mellőzzük — inkább ezt használjuk, mint a nyomdatechnikában bevett „kurrens" elnevezést. Így aztán egy számítástechnikus minden különösebb ismeret nélkül forgathatja a könyvet, a tipográfusoknak viszont előbb tanácsos ismereteket szerezniük a számítógépekről. Bizonyos szempontból mégis az utóbbiak vannak előnyben: egy számítástechnikus attól még nem lesz jártas a tipográfiában, hogy megtanulja a TEX programot használni. A tipográfus viszont a program ismeretében teljesen készen áll arra, hogy szép könyveket állítson elő a segítségével.

Néhány szót a TEX programrendszerről: ez egyik első képviselője a DTP programoknak. Mindenképpen kiemelkedik a társai közül a kivételes matematika és műszaki szedési képességeivel. Másik nagy előnye, hogy míg a többi mikroszámítógépre írt hasonló rendeltetésű program csak néhány különböző tipusú gépen fut (általában IBM PC-ken és Macintosh-on), addig a TEX számos más felhasználói rendszerre is elkészült. A PC-n futó programok közül talán ennek a legkisebb a hardver-igénye, vagyis már viszonylag kis kiépítéssel is használható. Persze fejletteb hardver mellett szebb eredmény létrehozására képes. Legsúlyosabb hátrányaként azt szokták emlegetni, hogy nem igazán „felhasználó-barát". Ez azonban nem gátolja a gyors terjedését hazánkon kivül, sőt mostanra már itthon sem. A hosszú nyomdai átfutási idő mindenképpen indokolttá teszi, hogy a legbonyolultabb, legmunkaigényesebb műszaki, matematikai feladatokat adó kéziratokat a számítógép közelében lévő szerzők maguk szedjék ki.

A DTP programok erősen függnek attól a nyelvtől, amelyre felhasználják őket. Gondoljunk csak arra, hogy még a latin ábécét használók közül is szinte minden nyelvnek más a betűkészlete és teljesen különböznek az elválasztási szabályai. A pontmátrixos nyomtatás igénytelen külalakja még elviselhetővé tette egy-egy magyar betű hiányát, de nyilvánvaló, hogy egy tipográfiailag magas színvonalú nyomtatvány nem engedheti meg még az 'Í' betű hiányát sem.

A könyv egyben a vele együtt forgalomba kerülő, a TEX programot a magyar nyelv kezelésére felkészítő MATEX programcsomag leírása, felhasználói kézikönyve is. A MATEX-ről bevezetőben csak annyit mondunk, hogy igyekeztünk kifejlesztésekor megőrizni a TEX program két nagy előnyét: a különböző TEX programok kompatibilitását és a viszonylag kis hardver igényt.

Így a MATEX minden IBM PC-n futó TEX rendszerrel használható és nem okoz különösebb gondot az sem, ha egy programot ugyanazon a gépen néha angol, néha magyar nyelvű szövegek feldolgozására kívánunk használni.

Félő, hogy a megcélzott feladatok közül (a DTP rendszerekbe való bevezetés, a TEX program leírása, a MATEX felhasználói utmútatója, a Magyarországon forgalomban levő TEX implementációk ismertetése) nem mindegyiknek fog a könyv maradéktalanul eleget tenni. Ezért előre is elnézést kérünk az Olvasótól.

A következőkben néhány tanáccsal szeretnénk szolgálni a könyv olvasásához:

Aki már jól ismeri a TEX programot, annak csak a MATEX-ről szóló részeket szükséges részletesebben elolvasnia. Könnyítésül azokat a bekezdéseket, amelyek különösen fontosak ebből a szempontból, megjelöltük a MATEX logoval. Ezeken kívül még a 9. fejezet egy szakasza szól teljes egészében a MATEX-ről. Természetesen a Függelék a MATEX installálásáról szóló része is szükséges számára.

A TEX-et egyáltalán nem ismerő felhasználóról is feltételezzük, hogy valamennyire járatos a PC-k, és tágabban a számítógépek világában, ismeri a DOS-t és valamilyen szövegszerkesztőt. Az A. függelékben megtalálható az IBM/PC/XT/AT-kre forgalmazott kétféle TEX installálásának leírása. Ennek elolvasása után a B. függelékben leírtak segítségével elvégezhetjük a MATEX installálását. Az első két fejezet nagy vonalakban vezet végig a TEX működésén, a harmadik pedig egy konkrét példán, a fülszövegen szemlélteti az addig tanultakat. Aki jobban szeretné először kiprőbálni a programot, az a 3. fejezetben leírtak alapján, az első két fejezet ismerete nélkül is megteheti ezt.

A további gyakorlati tudnivalókat a 4.–9. fejezet tartalmazza. Ezek közül a 4. az általános tördelésről, az 5. a matematikai szedésről, a 6. a táblázatok, ábrák elhelyezéséről szól. A 7. fejezet alapján új, saját utasítások definiálhatók, a 8. a hibákat foglalja össze, a 9. pedig a a TEX-re írt makrócsomagokat ismerteti röviden.

1. fejezet

Bevezetés

1.1. Számítógépes szövegelőállítás

A különböző kiadványokat a számítógépek megjelenése előtt a nyomdászok aprólékos, nagy hozzáértést igénylő munkával állították elő. Az évszázadok alatt kialakult nyomtatási, szedési tapasztalatok, tipográfiai elvek – amelyeket ma alkalmazunk – rendkívül fontosak ahhoz, hogy könnyen, gyorsan tudjunk olvasni nyomtatott szövegeket. A számítógépek megjelenésével a szövegszerkesztők ismert előnyei tették fontossá, hogy ily módon is elő lehessen állítani nyomtatványokat. Kezdetben azonban ezek a próbálkozások nem voltak versenyképesek a hagyományos nyomdai eljárásokkal szemben, mert a „listák" csúnyák és nehezen olvashatók voltak.

A számítógép eközben a nyomdaiparba is betört bonyolult, drága célberendezések, az ún. fényszedő gépek formájában. Ezek használata azonban komoly szakértelmet igényel. A számítógéppel segített igényes szövegelőállítás tehát csak akkor terjedhetett el széleskörűen, amikor megjelentek az első olyan szövegszedő programok, amelyek viszonylag kevés szedési és számítógépes ismeretet kívántak. Ezek közül a legelső, és a mai napig is legfontosabb a TEX .

A TEX egy számítógépes *szövegszedő program*, amely sokféle kis- és nagy-, mikró- és makró-, mega- és mini-számítógépen fut szerte a világon. Készítenek vele könyvet, újságcikket, tudományos publikációt, pamflettet, névjegykártyát, műszaki dokumentációt, magán- és hivatalos levelet.

A TEX program szinte minden szempontból szakértő „nyomdásznak" tekinthető. Ha esztétikai, tipográfiai elveket veszünk alapul, a számítógéppel

előállított kiadványok közül kiemelkednek azok a dokumentumok, amelyeket TEX-hel állítottak elő. Az Olvasó azonnal meggyőződhet erről, ha ezt a könyvet – melyet teljes egészében TEX-hel állítottunk elő – összehasonlítja a szokásos számítógéppel készült kiadványokkal.

A TEX programot a 70-es évek közepén D. E. Knuth fejlesztette ki. A program a nevét a görög τεχ szóból kapta, amely művészetet jelent. Mivel a görög χ hangot erős h-nak kell ejteni, a TEX kiejtése körülbelül ez: tekhh. Annak érdekében, hogy már a program neve is érzékeltesse a benne levő szedési lehetőségeket, ha a program nevét leírják, az E betűt egy kicsit lejjebb csúsztatják, és a T ill. X betűket közelebb húzzák hozzá. (Mivel a TEX-ről szóló kiadványok szinte kizárólag TEX-hel készülnek, a név ilyen előállítása semmilyen nehézséget nem jelent.

A szövegelőállító programokat három csoportba oszthatjuk:

- Az első csoportba tartoznak a *szövegszerkesztő programok*, amelyek segítségével legépelhetjük a szöveget, azon interaktív módon különböző javításokat hajthatunk végre, mint pl. törlés, beszúrás, már meglevő szövegek, táblázatok beillesztése, jelek, szavak kicserélése stb. Szövegszerkesztő például az EDLIN, a PE vagy a NORTON EDITOR.
- A második csoportba tartoznak a *szövegformázó programok*, amelyek arra szolgálnak, hogy a számítógépben tárolt – általában szövegszerkesztők segítségével előállított – „ömlesztett" formában levő szöveget sorokra, lapokra tördeljék, a fejléceket, lábjegyzeteket a nekik megfelelő helyre tegyék, stb. A szövegformázó programok általában nem interaktívak. A szövegszerkesztők által előállított file-t beolvassák, és a file-ban levő formázó utasítások alapján a szöveget a kívánt alakban kiírják. Szövegformázó programok például RUNOFF, a SCRIPT vagy a LETTRIX. Sok szövegformázó program tartalmaz egy szövegszerkesztőt is, így kényelmesebben használhatók. Az ilyen programok egy része képes arra, hogy a szöveget – az előre beállított paraméterek alapján – a begépelés közben azonnal formázza, így a képernyőn egyből a formázott szöveg jelenik meg.[1] A formázási hibákat ezekkel sokkal gyorsabban lehet kijavítani mint a file-ba beírt parancsokkal formázó programok esetében,

[1] Az angol szakirodalomban a *wysiwyg* (*w*hat *y*ou *s*ee *i*s *w*hat *y*ou *g*et = amit látsz azt kapod) rövidítést használják ezekre a programokra.

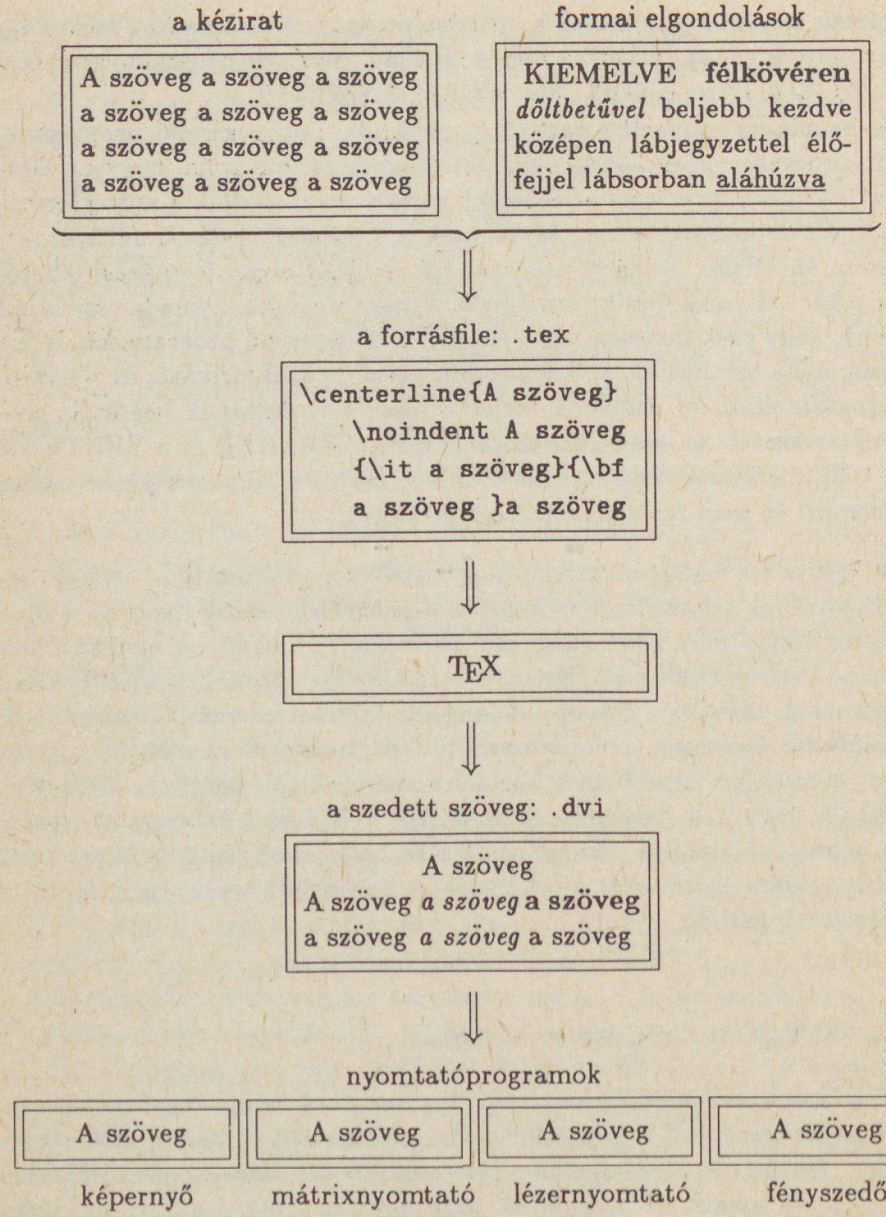
hiszen az utóbbiaknál csak a formázó program futtatása és a szöveg kinyomtatása után láthatjuk meg a hibákat. Wysiwyg programok például a WORDSTAR, KEDIT, MS WORD, XYWRITE.

- A harmadik csoportba tartozó programok a szövegszedő programok, ill. programnyelvek. Megkülönböztetésüket az indokolja, hogy az általuk nyújtott formázási lehetőségek messze meghaladják a szövegformázók által biztosítottakat, és képesek a nyomdai szedés munkáját teljesen átvállalni. Néhány program (pl. a TEX) olyan formázási lehetőségekkel is rendelkezik, amelyeket a hagyományos nyomdai szedéssel nem, vagy csak nehezen lehet elérni. A szövegszedő programokat is két csoportba oszthatjuk: szövegközbeni parancsokkal formázó, és wysiwyg programokra. Az utóbbiak természetesen tartalmaznak beépített szövegszerkesztőt is, ezek közé tartozik a PAGEMAKER és a VENTURA PUBLISHER. A TEX viszont *a szöveg közben elhelyezett parancsokkal formáz, és nem tartalmaz szövegszerkesztőt.*

A TEX ebből fakadó hátrányát nagymértékben csökkentik az olyan programok, amelyek a formázott szöveget a képernyőn jelenítik meg (pl. a Preview program, amely IBM PC-n fut). Másrészt a hátrányok mellett a parancsmódú szövegszedésnek előnyei is vannak: egy – korábban már formázó parancsokkal ellátott – szöveget könnyebb átalakítani más formátumúra; bonyolultabb szövegek (pl. matematikai kifejezések) könnyebben, egyszerűbben szedhetők. Általában a hosszabb szövegek (pl. könyvek, dokumentációk) ill. bonyolult szövegrészek szedésére a TEX az alkalmasabb eszköz, míg a néhány oldalas kiadványok (pl. kisebb terjedelmű, változó formátumú újságok, reklám szórólapok stb.) esetében valamelyik wysiwyg program a kényelmesebb eszköz.

1.1.1. A TEX működése vázlatosan

Az előállítandó szöveget és a formázó parancsokat először egy szövegszerkesztő segítségével kell begépelni. A szövegszerkesztőt szabadon választhatjuk meg. Fontos viszont, hogy ha a program szövegformázási lehetőségekkel is rendelkezik ezeket *ne használjuk.* Erre ugyanis – mivel a TEX formázza majd a szöveget – nincs szükségünk, másrészt a file-ba ekkor „láthatatlan", ún. vezérlő karakterek kerülnek, amelyek megzavarhatják a TEX működését.



1.1 ábra. A TEX működése

Ha elkészült a file (ezt a továbbiakban forrásfile-nak, a tartalmát forrásszövegnek nevezzük) meghívjuk a TEX programot, amely formázza, „szedi" a szöveget és ezt egy file-ba kiírja. A file-ok közötti eligazodást a következő elnevezési konvenció segíti:

- A forrásfile nevének kiterjesztése mindig tex. Például, ha a szöveget egy szoveg.dok nevű file-ban tároljuk, akkor a TEX nem fogja megtalálni, mert feltételezi, hogy a forrásfile kiterjesztése tex. A helyes elnevezés tehát szoveg.tex. A TEX programnak éppen ezért nem kell megadnunk a file teljes nevét, mert csak a tex kiterjesztésű file-ok között keres.
- A TEX által készített, formázott szöveget tartalmazó file mindig dvi kiterjesztésű. Esetünkben a TEX egy szoveg.dvi nevű file-t készít. Ezt a file-t azután különböző nyomtatóprogramok segítségével nyomtathatjuk papírra, képernyőre, vagy filmre.

A TEX programot Pascal nyelven[2] írták és az egész világon ingyen,- csupán a programot tartalmazó mágnesszalag áráért árulták. A TEX név azonban az AMS[3] védjegye, és az egyes gépeken való TEX implementációkat csak akkor engedték TEX-nek nevezni, ha az valóban kiállta a meglehetősen kemény — D. E. Knuth által „torture test"-nek nevezett — kompatibilitási próbát. (Természetesen olyan implementációk is születtek, amelyek nem az eredeti Pascal program adott gépre való fordításai voltak. Ilyen pl. az IBM PC-n futó MicroTEX, amely C nyelven íródott, de szövegszedési szempontból megfelelője az eredeti TEX programnak.) Ennek az üzletpolitikának áldásos következményeként a TEX gépfüggetlen szövegszedő nyelv lett. Tehát hasonlatos azokhoz a szabványosított programozási nyelvekhez, amelyeknél a forrásnyelvű program bármelyik gépen – amely az adott nyelv fordítóprogramjával rendelkezik – lefordítható és futtatható.

A TEX esetében azonban ennél még több is igaz: nem csak a TEX forrásfile-t vihetem át az egyik gépről a másikra, hanem az általa előállított, „szedett" szövegfile-t, a DVI file-t is. (Ez a programozási nyelvek esetében szóba sem jöhet, hiszen a fordítóprogram az adott gép assembler kódjára

[2] Valójában nem Pascal nyelven írták, hanem egy speciális Pascal-szerű nyelven. A TEX78 esetében SAIL, TANGLE nevezetű programok segítségével fordították le Pascal-ra, a TEX82 esetében azonban már készen állt az új WEB nevezetű program- és dokumentációs nyelv, aminek segítségével lefordították a forrásprogramot Pascal programra.

[3] AMS az American Mathematical Society rövidítése

fordít). A DVI rövidítés éppen erről kapta nevét: a DeVice Independent (gép független) szavakból alkotott betűszó. A TEX program a szöveg szedésekor nem tudja (nem is tudhatja), hogy az elkészített szöveg a képernyőn, papíron vagy filmen jelenik-e majd meg.

A gyakorlatban a DVI file tartalma legtöbbször a képernyőre kerül. Ez a leggyorsabb, legkényelmesebb eszköz, ha javítunk vagy hibát keresünk. Ha a szedett szöveg megfelelő, a nyomtató program segítségével papírra nyomtatható anélkül, hogy a TEX programot újra futtatni kellene. Ha a mátrixnyomtató nem megfelelő felbontású, a DVI-file-t kinyomtatható lézerprinteren — vagy ha magasabb esztétikai igényeket kell kielégíteni — fényszedő gépen is. Természetesen ahhoz, hogy mindezt meg lehessen tenni, a TEX program mellett az egyes nyomtatóprogramokra is szükség van, ugyanis minden eszközre más és más nyomtatóprogram kell.

A TEX lehetővé teszi, hogy valaki az íróasztal mellett ülve, a személyi számítógépén írjon egy könyvet, amely esztétikailag is megfelelő. A nyomdában aztán csak egy automatikus feldolgozásra van szükség, nem kell korrektúrákat javítani. A szerző — a TEX szakmai segédletével — egyben szedője is írásművének.

1.2. A TEX-hez kapcsolodó file-ok, programok

1.2.1. A TEX TFM file-jai

Ahhoz, hogy szép kiadványokat, leveleket, stb. lehessen előállítani, sokféle betűtípus kell. A TEX-et használók számára általánosan a következő betűkészletek állnak rendelkezésre:

- AMR készletek: 'Almost Modern Roman' — majdnem modern antikva ez az alapértelmezés: a szövegek nagy része ezzel a betűtípussal készül
- AMBX készletek: 'Almost Modern Bold eXtended' — majdnem modern félkövér nyújtott; **címek szedésére**
- AMSL készletek: 'Almost Modern Slanted Roman' — majdnem modern dőlt antikva; *szövegkiemelésekre*
- AMTI készletek: 'Almost Modern Text Italic' — majdnem modern szöveg kurzív; *szintén szövegkiemelésekre*

- AMTT készletek: 'Almost Modern Typewriter Type' — majdnem modern írógép típus; **programok szedéséhez**
- AMMI készletek: 'Almost Modern Math Italic' — majdnem modern matematikai kurzív; *matematikai* kifejezésekhez
- AMSY készletek: 'Almost Modern Math Symbols' — majdnem modern matematikai szimbólumok
- AMEX készletek: 'Almost Modern Math Extension' — majdnem modern matematikai kiterjesztések (kiterjesztett szimbólumok)

A fontkészleteket a C. függelékben részletesen bemutatjuk. A felsoroltakon kívül még sok betűtípus létezik (pl. Medley), de a TEX programot vásárló felhasználó általában ezekre építhet.

Az egyes betűtípusok általában 5–12 pont[4] közötti nagyságban érhetők el, vagyis van 5 pontos amr betű, van 6 pontos amr betű ... van 10 pontos amr betű. A különböző pontú betűk nem tekinthetők egymás egyszerű nagyításának vagy kicsinyítésének, hanem valóban különböznek egymástól. Ha az 5 pontos 'a' betűt felnagyítjuk akkorára mint a 10 pontos 'a' betű, megláthatjuk a különbséget.

Általában csak a legfontosabb betűtípusok vannak meg minden pontmagasságban, a többi legtöbbször 10 pontos nagyságban található meg. Annak érdekében, hogy a TEX program minden betűtípussal tudjon dolgozni, a betűtípusokról semmilyen információt nem építettek be a programba. Ezeket az információkat külön file-ok, ún. TFM (TEX Font Metric) file-ok tartalmazzák. Ha egy új betűkészletet[5], fontot kapunk, s ehhez a megfelelő TFM file is rendelkezésünkre áll, a TEX-hel ezt is használni tudjuk. A TFM file nem tartalmazza a betűképet, csak a legfontosabb, a szedéshez szükséges információkat mint pl. az egyes betűk szélessége, magassága, a dőlésük, szavak közötti távolság nagysága és így tovább. Ezek a file-ok nem szövegfile-ok, rendkívül tömör (bináris) formában vannak tárolva az információk, így közvetlenül nem olvashatók.

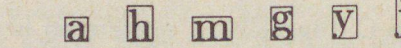
---

[4] A pont nyomdászati egység; pl. 7 pontos betűtípus azt jelenti, hogy a betű magassága 7 pont. 1 cm=28,453 pont. Erről részletesen a 2. fejezetben szólunk.

[5] az angol megfelelője, a *font* nálunk is meghonosodott, ezért a könyv további részében mi is használjuk

1.3. DVI file-ok

A TEX a következőképpen kezeli a betűket. A betűk egy képzeletbeli vonalon — alapvonalon — helyezkednek el. Az olyan betűk, mint pl. az 'a', 'h' vagy az 'm' csak a vonal felett íródnak, míg a 'g', 'y' vagy a 'j' betűk lenyúlnak a vonal alá is.



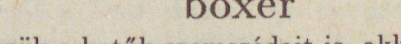
Szedéskor a betűk elhelyezkedését az alapvonalon kívül az alappont is segíti. A betűhöz tartozó alapvonalat és alappontot a betű tervezésekor határozzák meg. A TEX a betűt mint egy téglalapot[6] kezeli, amelynek magassága, szélessége, mélysége van. A betű azonban ki is nyúlhat a dobozból, például akkor, ha a betű dőlt.[7]



A TEX úgy építi fel a sorokat, oldalakat, hogy bizonyos szabályok alapján rakja a betűknek megfelelő dobozokat egymás mellé ill. alá. Mik lehetnek ezek a bizonyos szabályok?

1.3.1. A betűk közötti távolság

Egy szép nyomtatott szövegre jellemző, hogy a betűk elhelyezése a szomszédos betűktől is függ. Nézzük például a boxer szót nagyítva. Ha a betűket a szélességüknek megfelelően egymás mellé rakjuk ezt kapjuk:



Ha figyelembe vesszük a betűk szomszédait is, akkor a tipográfiai elveknek megfelelően a o és az x betűt egy kicsit közelebb húzzuk egymáshoz, hiszen a x „szájába" szépen illik az o domborulata. A TEX így szedi ki ezt a szót:

---

[6] Angolul 'box'(=doboz)-nak hívják

[7] Angolul két kifejezést használnak dőlt betűk esetén: 'slanted'(=döntött) betű ill. 'italic'(=latinos), aminek magyar megfelelője a kurzív.

boxer

A betűket befoglaló téglalapok ekkor többé-kevésbé átfedik egymást. A közelebb húzás mértéke természetesen függ a betűtípustól is, ezért ezt a megfelelő TFM file tartalmazza minden betűtípusra. Például az írógép-szerű betűtípusban egyáltalán nincsen ilyen: `boxer`.

1.3.2. Ligatúra

A „közelebb húzás" különleges fajtájának tekinthetjük, amikor bizonyos betűpárokat, betű-hármasokat egyetlen komplex jellel helyettesítjük. Például a fiam szóban az 'f', 'i' betűket a fi jellel, ún. ligatúrával helyettesítettük. A magyar szövegekben szokásos ligatúrák a következők:

ligatúra	ligatúra nélküli szó	ligatúrával
ff	puffanás	puffanás
fi	fiam	fiam
ffi	maffia	maffia
fl	mafla	mafla
ffl	mufflon	mufflon

Az egyes ligatúrák is változnak betűtípusonként. A maffia szó írógép típusú betűvel: `maffia`.

1.3.3. Kurzív kiegyenlítés

Ha a sorban úgy emelünk ki egy vagy több szót, hogy dőlt betűvel szedjük, akkor a dőlt és álló betűk találkozásánál túlságosan kevés szabad hely marad. Ezt kiegyenlítendő a TEX egy kis plusz helyet iktat be a két szó közé:

Így *milyen közel* van?

Ez a sor kurzív kiegyenlítés nélkül:

Így *milyen közel* van?

A kurzív kiegyenlítéshez a TEXnek külön parancsot kell adni, erről a következő fejezetben még szó lesz.

1.3.4. Sorkizárás

A legtöbb szöveg bal és jobb oldalra zárt, vagyis a soroknak egyenlő hosszúaknak kell lenniük. Azt nem tudjuk mindig elérni, hogy a sorok ugyanannyi betűt tartalmazzanak még akkor sem, ha elválasztásokat alkalmazunk. Másrészt azonban a betűk sem egyenlő szélesek, ezért a sorok csak úgy lehetnek egyenlő hosszúak, ha a szavak ill. a betűk közötti távolságot aszerint csökkentjük ill. növeljük, hogy melyikre van szükség a sorkizárás eléréséhez. Ezzel azonban nagyon óvatosan kell bánni.

Túlzott alkalmazása csúnya nyomtatási képhez vezethet.

Nyújtsuk például az előző mondatot amely 95 mm hosszú, 110 mm majd 120 mm hosszúra:

Túlzott alkalmazása csúnya nyomtatási képhez vezethet.

Túlzott alkalmazása csúnya nyomtatási képhez vezethet.

A TEX automatikusan nyújtja, sűríti a sorokat, de egy bizonyos határon túl — ez a határ ott van, amit egy gyakorlatlan szem is éppen észrevehet — hibajelzést ad. Ez akkor fordulhat elő leggyakrabban, amikor egy szó nem fér ki egy sorba és a TEX nem tudja elválasztani. Ha az egész szót a következő sorba teszi, az eredeti sor olyan röviddé válik, hogy nyújtással nem lehet a sorkizárást elérni.

1.3.5. Lapokra tördelés

Ha megadjuk a TEX-nek a szedéstükör[8] méreteit, akkor automatikusan beállítja a sorok közti távolságot úgy, hogy az előírt méret jöjjön ki. Ha ábrák, címek és egyéb, nem sorokra tördelt szövegek vannak azon az oldalon, a TEX a sorok közti távolságot növeli vagy csökkenti. Hasonlóan a sorok

[8] a nyomdászok szedéstükörnek nevezik azt a téglalapot, amelyben az oldalon lévő szöveg — a betűk — elhelyezkednek. Méreteit az első sor tetejétől az utolsó sor aljáig, ill. az egy sorban lévő első betű kezdetétől a sorban lévő utolsó betű végéig számolják. Például: ennek a könyvnek a (nem szabványos) B5 formátumához tartozó tükörmérete 126mm×170mm.

szedésénél tapasztaltakhoz, bizonyos határon túl itt is hibaüzenetet ad. Az egy oldalra kiszedett sorok száma függ a betűtípustól és méretétől is.

A TEX által szedett szöveget a DVI file tartalmazza utasítások formájában. Az utasítások ilyen alakúak:
„Vedd a ... fontkészlet ... méretű ... betűjét és helyezd az oldal ... , ... koordinátájú pontjára."

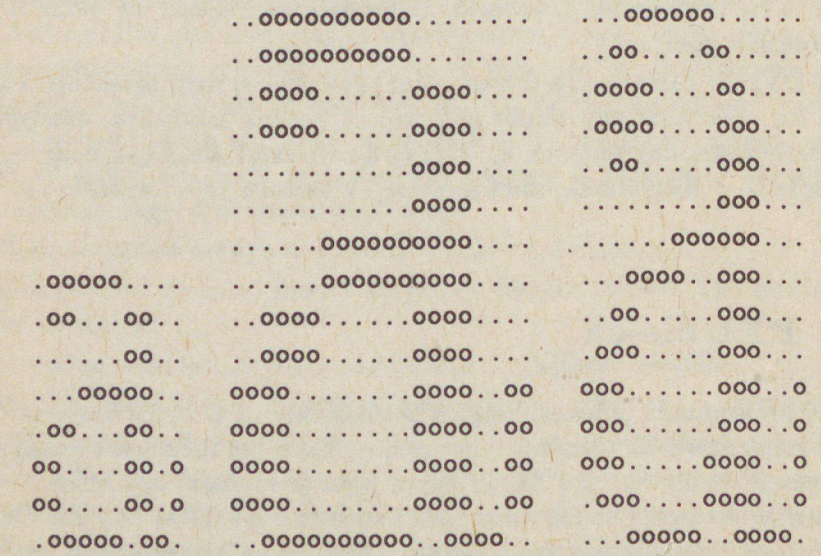
A DVI file tartalma (a forrásfile-lal ellentétben) nem olvasható közvetlenül. Az utasítások egy tömör kód szerint vannak felsorolva, amelyeket a nyomtatóprogramok fejtenek ki. A DVI file-ok ezért általában nem — vagy nem sokkal — nagyobbak, mint a hozzájuk tartozó TEX forrásfile-ok.

1.4. PXL file-ok

A TEX programnak nincs szüksége a betűképekre. A DVI file kinyomtatását végző programoknak azonban már igen; ezeket a betűképeket tartalmazzák az egyes PXL file-ok. A PXL az angol *pixel* (=rácsozat egy eleme) szóból származik. Az ugyanahhoz a betűkészlethez tartozó TFM és PXL file-okat joggal hasonlíthatjuk egy érem két oldalához. Mivel ugyanarról a betűkészletről tartalmaznak információkat, szorosan összetartoznak, de a TFM file csak a TEX számára szükséges adatokat, a PXL file pedig a nyomtató program számára szükséges adatokat tartalmazza. Ahogy a TEX program futtatható akkor is, ha egyetlen PXL file sincs, a nyomtató programok is futtathatók TFM file-ok nélkül is, mert a betűkészlethez szükséges információt az egyik kizárólag a TFM file-ból, a másik pedig kizárólag a PXL file-ból nyeri.

A gyakorlatban sokszor előfordul, hogy a címeket vagy más kiemelendő szavakat nagyobb betűvel kell szedni. Ilyenkor az adott betűtípus adott betűjére jellemző értékek, mint pl. magasság, mélység, szélesség, csak egy szorzótényezővel változnak meg, a betűt kinagyítjuk mondjuk a kétszeresére. A TEX az eredeti TFM file alapján ki tudja számolni a nagyított betű méreteit, tehát nincs szüksége egy másik TFM file-ra, amely a kétszeresére kinagyított betűk adatait tartalmazza. A nyomtató programnak azonban szüksége van a kétszer akkora betűképre. Megtehetnénk, hogy például a normál méretű 'a' betű képét kinagyítjuk kétszer akkorára és ezt használjuk, de ez megle-

hetősen csúnya lenne. A következő ábra bal oldalán az AMR5 fontkészlet 0-ás nagyítású 'a' betűjét rajzoltuk le, mellette a kétszeresére kinagyított betűkép, a harmadik pedig az a finomabb felbontású betűkép, amelyet a nyomtatóprogram a valójában használ, ha nagyobb betűt szedünk.



1.2. ábra. Különböző nagyítású és felbontású betűk.

Ezért minden betűkészlethez egy TFM file tartozik, és annyi PXL file, ahány nagyításunk van az adott betűkészletről. A TEX a következő nagyításokat használja:

nagyítás	TEX elnevezés	az elnevezés értelme
eredeti	0	$1.2^0 = 1$
1.09543..szeres	half[9]	$1.2^{1/2} = 1.09543..$
1.2 szeres	1	$1.2^1 = 1.2$
1.44 szeres	2	$1.2^2 = 1.44$
1.728 szeres	3	$1.2^3 = 1.728$
2.074 szeres	4	$1.2^4 = 2.074$
2.488 szeres	5	$1.2^5 = 2.488$

[9] a half angol szó, magyarul felet jelent

nagyítás	nagyság	
eredeti	0	a fiam kocog
1.09543.. szeres	half	a fiam kocog
1.2 szeres	1	a fiam kocog
1.44 szeres	2	a fiam kocog
1.728 szeres	3	a fiam kocog
2.074 szeres	4	a fiam kocog
2.488 szeres	5	a fiam kocog

Az azonos nagyításhoz tartozó PXL file-okat egy alkönyvtárban fogja össze. Nagyon fontos ez a szerkezet, hiszen az azonos betűkészlethez tartozó, de különböző nagyítású PXL file-oknak azonosak a neveik!

Csak a legalapvetőbb betűkészletek vannak meg minden nagyításban. Ezek a már felsorolt AMR, AMBX, AMTI, AMTT, AMMI, AMSL fontkészletek. Körültekintően kell megválasztanunk a betűtípusokat. Ha a szövegben szükség lesz az adott típus 'a' betűjének valahányszoros nagyítására, meg kell győződnünk arról, rendelkezünk-e a megfelelő nagyítású PXL file-lal. Ha ezt elmulasztjuk, a szöveget a TEX program hibaüzenet nélkül szedi ki (hiszen a TEX nem használja a PXL file-okat), de a nyomtató program egy hibaüzenettel leáll és a szöveget nem tudjuk kinyomtatni. (Erről részletesebben a 8. fejezetben szólunk.)

A PXL file másik jellemzője a felbontás. Felbontáson szemléletesen azt értjük, hogy milyen finom, aprólékos részletekből, pontokból rakja össze a betűket, jeleket. Mértékegysége a dpi, amely az angol 'dot per inch' rövidítése; magyarul inch-enkénti[10] pontok száma. A legutolsó ábrán az AMR10 betűtípus 'a' betűjét láthatjuk kinagyítva. Ezt a betűt egy 240 dpi felbontású PXL file-ból vettük, amely a normál méretű — vagyis 0-s nagyítású — betűket tartalmazza. Ez tehát olyan részletességgel megrajzolt betű, amely 2,54 cm-enként 240 fehér ill. fekete pontot használ. Mátrixprinterek esetében általában 120 vagy 180 vagy 240 dpi-s PXL file-okat használnak, míg lézerprintereknél a legelterjedtebb felbontás a 300 dpi. A nyomdákban hasz-

[10] 1 inch=2,54 cm

nált nagy fényszedő gépek felbontását 1000–2000 dpi-re szokták becsülni.[11] Ez a könyv egy 300 dpi felbontású PXL file segítségével lézernyomtatón készült.

Alapvető esztétikai elv, hogy a kinyomtatott szöveg minden betűje, jele a nagyítás nagyságától függetlenül, ugyanolyan felbontású legyen. Ezért a felbontás az egész PXL file készlet minden file-jára érvényes kell, hogy legyen: minden betűtípusra és minden nagyításra. Az előző pontban említettük, hogy a nyomtató program nem nagyít automatikusan; ekkor ugyanis egy kétszeres nagyítás esetén a betű mérete ugyan kétszer akkora volna, de a felbontása csak fele a szöveg többi része felbontásának.

A PXL file-ok nem szövegfile-ok, hanem bináris file formában tartalmazzák a betűképet. Különösen a nagyobb nagyítások esetén nagy méretűek (pl. AMR10 5-ös nagyítás mellett kb 45 kbyte méretű). Ha még ehhez hozzávesszük, hogy a PXL file-ok száma egy átlagos kiépítettséget (installációt) alapul véve kb 250–300, akkor láthatjuk, hogy a file-ok tömörítésével több megabyte tárkapacitást takaríthatunk meg. Ezért a PXL file-okat különböző tömörített alakban szokták forgalmazni. Ezeket a tömörített file formátumokat két csoportra oszthatjuk: az egyikbe tartoznak azok a file formátumok, amelyeket a nyomtatóprogramok konvertálás nélkül használni tudnak, ilyen pl. a PCTEXhez tartozó nyomtatóprogram által használt PK (packed PXL format) formátum, vagy a MicroTEXhez tartozó nyomtatóprogram által használt EPF (Epson Pixel Format) file formátum. A másik csoportba tartoznak azok a file formátumok, amelyeket a nyomtatóprogramok nem tudnak használni, csak arra szolgálnak, hogy segítségükkel a PXL file-ok szállítására kevesebb mágneslemezt kelljen felhasználni. Ilyen pl. a MicroTEX PP (packed pixel) formátuma.

A felsoroltakon kívül még más file formátumok is léteznek (pl. GF, 'general format'), ezek azonban az IBM PC-n nem használatosak.

---

[11] A fényszedő gépek valójában nem fekete-fehér pontokból rakják össze a betű képét, hanem egy filmtekercsen levő betűkép mintákat „fényképeznek" — levilágítják — a megfelelő helyre.

1.5. Makrók, makrócsomagok, formátumfile-ok

A TEX program közvetlenül kb. 300 alapvető parancsot, ún. primitívet ért meg. Ezzekkel a parancsokkal mindent, amire a TEX egyáltalán képes, meg tudunk tenni, de rendkívül fáradságos, aprólékos munkával. A TEX talán legfontosabb lehetősége, hogy nem lezárt szövegszedő program, hanem sokkal inkább egy programnyelv, amelyben az alapvető parancsok, mint egy programozási nyelv utasításai, segítségével programokat, ún makrókat állíthatunk össze. Több szövegszedő, sőt még szövegformázó program is rendelkezik azzal a lehetőséggel, hogy néhány szövegformázó utasítást egyetlen utasítással fogjunk össze. Ez azonban még nem teszi alkalmassá a programot arra, hogy szövegformázó *programokat* írjunk benne. A TEX azonban rendelkezik a programozási nyelvek alapvető utasítástípusaival, vannak olyan parancsai, amellyel ciklust lehet szervezni; elágazást, esetszétválasztást lehet kezelni; file-okat lehet megnyitni, olvasni, írni; változókat, regisztereket lehet definiálni és azoknak értéket adni stb. Egy ilyen programot hívunk a TEX-ben *makrónak* és ezt a programot az általunk adott makrónévvel, mint szövegformázó paranccsal indíthatjuk el.

A makrók együttesét, ha azok egy bizonyos szövegformázási feladatkört fednek le, *makrócsomagnak* hívunk. A TEX tervezésekor sem gondoltak arra, hogy a szövegeket majd primitívek segítségével fogják formázni a felhasználók. Az alkotók mindjárt a program elkészítésekor megalkottak egy makrócsomagot, a *PLAIN*-t amely az alapvető, szedéskor leggyakrabban előforduló feladatokat könnyíti meg. A PLAIN kb. 600 parancsot tartalmaz. Jelentősége akkor tűnik ki igazán, ha a TEX-hel formázott szövegeket megvizsgálva rájövünk, hogy a parancsok túlnyomó többsége makró, primitívekből összeállított program.

Különleges formai igényekkel rendelkező szövegeknél kényelmes, ha további, PLAIN-ben nem létező makrókat definiálunk, vagy további makrócsomagokat használunk. A TEX felhasználók körében általánosan elterjedt makrócsomag a LATEX, az AMSTEX és a VANILLA. Ezeket ismertetjük a 9. fejezetben.

Amikor egy makrócsomagot használunk, a makrókat — primitívekből felépített programokat — a TEX lefordítja a számítógép nyelvére. Mivel ez viszonylag sok időt igényel, és mivel a makrócsomagokat nem — vagy csak nagyon ritkán — változtatjuk meg, lehetőség van arra, hogy a makrócsomagot lefordított formában, ún. *formátumfile* alakjában tároljuk. A TEX futtatásakor nem kell minden egyes alkalommal lefordítanunk a makrócsomagot, hanem a formátumfile-t közvetlenül betölthetjük, és használhatjuk. A formátumfile-okra az FMT rövidítést szokták használni.

2. fejezet

TEXnikai alapozás

A TEX-re különösen igaz, hogy könnyebb elsajátítani használatát különböző példákon keresztül, mint hosszas leíró magyarázat alapján. Ezért könyvünkben sok példa fog szerepelni, melynek formája a következő:

A példát két hasábban mutatjuk be: a jobboldali hasábban a forrásszöveg látható TEX parancsokkal és a begépelt szöveggel; a baloldali hasábban pedig az eredményként előálló szöveg.

```
A példát két hasábban mutatjuk
be: a jobboldali hasábban a
forrásszöveg látható \TeX\
parancsokkal és a begépelt
szöveggel; a baloldali hasábban
pedig az eredményként előálló
szöveg.
```

2.1. A TEX által használt betűk és jelek

A forrásszöveget tartalmazó file-ban előforduló karaktereket a következő csoportokba oszthatjuk:

Betűk, számok

Ide tartoznak az angol ábécé nagy és kis betűi a-tól z-ig, ill. A-tól Z-ig és a számok. A MATEX programcsomag használata esetén ide tartoznak még az angol ábécéből hiányzó ékezetes kis és nagy betűk, tehát ebben az esetben betűkön a magyar ábécé kis és nagy betűit is értjük.

Vezérlő karakterek

Ezek azok a karakterek, amelyek segítségével a TEX-nek formázó parancsokat adhatunk:\ # & % { } $ ^ _ ~.

Egyéb karakterek (jelek)

Ide tartoznak a forrásszövegben előforduló egyéb jelek, például a központozás jelei:. ; : ! " - és más jelek például:) (=.

A forrásszövegben lévő karaktereket a TEX feldolgozza. A feldolgozás során az egyes karakterekből, karaktersorozatokból a formázott szövegben különböző betűket, jeleket készít. A betűknek és a számoknak — ha nem egy TEX formázó parancs részei — betűk és számok felelnek meg. Például:

```
fiam                    fiam
```

Itt a TEX az 'f' és 'i' betűpárból az 'fi' ligatúrát készítette el, az 'a' betűből 'a'-t, az 'm' betűből 'm'-et rajzolt. A jelek egy része a betűkhöz hasonlóan viselkedik. A forrásszövegben levő központozási jelek központozási jelként jelennek meg a formázott szövegben is. Például:

```
Ezek — a mondat végén —     Ezek --- a mondat végén ---
„központozási" jelek:.;??   ,,központozási" jelek:.;??
```

MATEX A példából látszik, hogy míg két egymás mellett álló kérdőjel ugyanúgy, két egymás melletti kérdőjelként jelenik meg, addig a két egymás mellett álló kivonás („mínusz") jelből a TEX egy — kötőjelnél hosszabb, gondolatjelnél rövidebb — vonást készít. Két egymás mellett álló vesszőből pedig „magyar" idézőjel lesz. Ez a fajta idézőjel csak a MATEX-nél használható.

A vezérlőkarakterek általában nem jelennek meg közvetlenül a formázott szövegben. Hatásukra valamilyen különleges jel íródik ki, vagy más betűtípussal szedett szöveg kezdődik, vagy egyéb szövegformázási parancs hajtódik végre. A leggyakrabban használt vezérlő karakter a „backslash"[1]:\. A következő táblázatban bemutatjuk az összes — forrásszövegben megengedett — karaktert, és azt, hogy a feldolgozás után mi lesz belőle:

[1] A 'slash'(=ferde vonal,hasíték) a /-t jelenti, a 'back'(=vissza) a „visszafelé" álló ferde vonalra utal.

forrásszöveg karakterek	hogyan jelenik meg
a...z	a...z (kisbetűk)
A...Z	A...Z (nagybetűk)
1,2,3...9,0	1,2,3...9,0 (számok)
(szóköz)	általában szóköz
(sorvége karakter)	általában szóköz
-	- (kötőjel, divíz)
--	– (vonás, kismínusz)
---	— (gondolatjel, mínusz)
=	= (egyenlőségjel)
:	: (kettőspont)
;	; (pontosvessző)
,	, (vessző)
`	‘ (kezdő aposztróf jel)
'	’ (záró aposztróf jel)
``	“ (angol kezdő idézőjel)
''	” (záró idézőjel)
,,	„ (magyar kezdő idézőjel) MATEX
"	” (záró idézőjel)
/	/ (per jel)
<	¡ (a PLAIN-ben spanyol felkiáltójel)
	Í (a MATEX-ben) MATEX
>	¿ (a PLAIN-ben spanyol kérdőjel)
	Ú (a MATEX-ben) MATEX
!	! (a PLAIN-ben)
	í (a MATEX-ben) MATEX
?	? (kérdőjel)
!'	¡ (a PLAIN-ben spanyol felkiáltójel)
	í’ (a MATEX-ben) MATEX
?'	¿ (a PLAIN-ben spanyol kérdőjel)
	?’ (a MATEX-ben) MATEX
(,)	(,) (kezdő és záró gömbölyű zárójel)
+	+ (plusz jel, összeadás jel)
[,]	[,] (kezdő és zárószögletes zárójel)
*	* (csillag)
\, #, &, %, {, }, $, ^, _, ~	TEX vezérlőkarakterek

MATEX logoval jelöltük a táblázatban azokat az eseteket, amikor a MATEX használata megváltoztatja a begépelt karakterek hatását. Az eltérések összefoglalva:

- a < ill. a > karakterek hatására nem ¡ ill. ¿ jelenik meg, hanem Í ill. Ú.
- a ! ' ill. a ? ' ligaturák nem használhatók: hatásukra nem a ¡ ill. ¿ jelenik meg, hanem í' ill. ?'.
- a , , ligatura azonban csak a MATEX-ben használható; hatására „ jelenik meg.
- a ! karakter hatására nem !, hanem í jelenik meg. A ! karakter vezérlő karakter, ezért szövegben paranccsal állítjuk elő a felkiáltójelet. Erről a későbbiekben még lesz szó.

A \ % { } vezérlő karaktereket ebben a fejezetben, a $ ^ _ karaktereket az 5. fejezetben, az & karaktert a 6. fejezetben, és a # karaktert a 7. fejezetben ismertetjük részletesen.

Fontos tudni, hogy a TEX miként kezeli a forrásszövegben levő szóköz és sorvége karaktereket. Ha két szó között egy szóköz karakter áll, akkor ez a két szó a formázott szövegben is két külön szóként jelenik meg. Ha a két szó között egynél több szóköz karakter áll, ezt a TEX egyetlen szóköz karakternek veszi:

egy szóköz, két szóköz, tíz szóköz.	`egy szóköz, két szóköz, tíz szóköz.`

(Ha két szó közé nem teszünk szóköz karaktert, akkor természetesen egybeírva jelennek meg.)

A sorvége jelet a TEX egy *szóköz karakternek* veszi. Ha tehát az előbbi példában levő szavakat két külön sorba írjuk, ugyanazt a formázott szöveget kapjuk:

	egy
egy szóköz	szóköz

Ha a sor végén vagy a következő sor elején szóköz karakter (karakterek) áll(nak), a TEX a fenti szabály alkalmazásával a szóköz és sorvége jeleket egyetlen szóközzel helyettesíti. A TEX tehát *nem veszi figyelembe a forrásszöveg alakját,* azt, hogy hogyan van sorokra tördelve, milyen szavak vannak egymás alatt stb. A formázott szöveg alakja — egyetlen kivételtől eltekintve — csak a forrásszövegben elhelyezett parancsoktól függ. Ez az

egyetlen kivétel az *üres sor*. Ha a TEX a forrásfile-ban egy üres sort talál, új bekezdést nyit. Például:

```
Ez egy nagyon rövid bekezdés. Két mondatból
áll, ez a második és egyben az utolsó mondata.

Ez már a következő bekezdés.
```

A TEX ezt így szedi:

Ez egy nagyon rövid bekezdés. Két mondatból áll, ez a második és egyben az utolsó mondata.

Ez már a következő bekezdés.

A forrásszövegben elhelyezhetünk olyan szövegrészeket — megjegyzéseket, értelmezéseket — is, amelyeket nem akarunk megjeleníteni a formázott szövegben. Ezt, a programozási nyelvektől átvett szóhasználat szerint *kommentnek* nevezett szövegrészt, a % vezérlőkarakterrel különíthetjük el. Például:

```
Ez a látható rész,          Ez a látható rész, % és
                            % ez már láthatatlan.
```

A % jelet *komment-jelnek* nevezzük. Hatására az adott sorban a % után következő karaktereket a TEX nem dolgozza fel, hanem a következő sor feldolgozásába fog. Ebből következik, hogy a TEX a % karaktert tartalmazó és az azt követő sort egyetlen sornak tekinti, ugyanis a % karakter miatt a sorvége karaktert sem veszi figyelembe. A következő példában a % karakter alkalmazásával mutatjuk be, hogy a TEX két külön sorban levő szót is tud egybeírva szedni.

```
Öreg TEXnő nem vén teknő.   Öreg \TeX% ember, mégpedig fiatal
                            nő nem vén teknő.
```

Végül megemlítjük, hogy lehetőség van arra is, hogy a TEX vezérlő karaktereit átdefiniáljuk: például megtehetjük, hogy a komment-jel, a % jel helyett a ; jel legyen. A vezérlőkarakterek átdefiniálása azonban messzemenő következményekkel jár, mert a különböző makrócsomagok felhasználják a definícióikban. Ezért a vezérlőkarakterek átdefiniálását még a gyakorlott

TEX felhasználónak sem ajánljuk. A Függelékben megadjuk a billentyűzetről begépelhető betűk, jelek, vezérlőkarakterek kódját, és a MATEX által használt ékezetes betűkódokat.

Sokszor szükség lehet arra is, hogy például $, % jeleket jelenítsünk meg a formázott szövegben. Ezeket is elő lehet állítani, de nem úgy, ahogyan a betűket, mert a TEX ezeket, mint a szöveg formázását leíró karaktereket értelmezi, és a formázott szövegbe nem teszi bele. Ahhoz, hogy ezeket előállítsuk, TEX parancsokat kell használnunk. Hasonlóan, TEX paranccsal állítjuk elő a billentyűzetről be nem vihető különleges jeleket, mint például a görög betűket, matematikai jeleket, szimbólumokat, stb. A TEX használatakor rendelkezésre álló széles jelkészletből adunk ízelítőt az 5.fejezetben és a C. függelékben.

2.2. TEX parancsok

A TEX parancsok mindig \ jellel kezdődnek és alapvetően két csoportba sorolhatók: parancsjelek és parancsszavak.

2.2.1. Parancsjelek

Egy parancsjel két karakterből áll. A \ karakterből és egy olyan karakterből, amelyik nem betű és nem szám[2] Például a \$ parancsjellel állítjuk elő a $ jelet, a \% parancsjellel pedig a % jelet.

```
mennyi 110$ 25 %-a ?          mennyi 110 \$ 25 \%-a ?
```

Figyeljük meg, hogy a példában a forrásszövegnél a \$ parancsjel után szóköz következik. A formázott szövegben is ez történik, de a \% parancsjel esetében egyik szövegben sincs utána szóköz. Mivel a TEX „tudja", hogy a parancsjel összesen két karakterből áll, azt is „tudja", hogy hol van a parancs

[2] Az elnevezés kicsit megtévesztő, mert azt sugallja, hogy egyetlen darab jelről van szó. Valójában a két jel *együttesét* hívjuk parancsjelnek.

vége, ezért a parancsjelet egybeírhatjuk más betűkkel, jelekkel, nem kell közéjük szóközt tennünk. Ha például különleges betűket akarunk előállítani, a parancsjel a szó belsejében szerepel:

châpeau, voilà, español	`ch\^apeau, voil\'a, espa\~nol`

Bár a MATEX használatakor nincs szükség rá, parancsszavakkal előállíthatunk magyar ékezetes betűket is.

Így is lehetséges, de MATEX-hel sokkal könnyebb.	`\'Igy is lehets\'eges, de \maTeX-hel sokkal k\"onnyebb.`

Az általánosan használt parancsjeleket gyűjtöttük össze a következő táblázatban:

parancsszó	eredménye
\$	$
\#	#
\&	&
\%	%
_	-
\‘	„grave” ékezet, pl. è, à
\’	„acute” ékezet, pl. é, á
\"	„umlaut” ékezet, pl. ü, ö
\~	„tilda” ékezet, pl. ñ
\^	„sapka” ékezet, pl. â
\=	„vonás” ékezet, pl. ȳ
\.	„pont” ékezet, pl. ṗ
\}	} (csak matematikai módban)
\{	{ (csak matematikai módban)
\,	egy „kis” szóköz (csak matematikai módban)
\>	közepes szóköz (csak matematikai módban)
\;	nagy szóköz (csak matematikai módban)
!	negatív „kis” szóköz (csak matematikai módban)
*	(szorzásjel, csak matematikai módban)
\\|	|| (norma-jel, csak matematikai módban)
\+	táblázatkezelő parancsjel
\-	elválasztás

␣	szóköz
\/	kurzív kiegyenlítés
\?	! MATEX

MATEX A táblázat utolsó eleme szolgál arra, hogy az MPLAIN-ben felkiáltójelet tudjunk létrehozni.

A táblázatból hiányoznak a következő billentyűzetről bevihető jelek: @ ( ) \ [ ] : <. Az ezekhez tartozó parancsjelek (\@ \(\) \\ \[\] \: \<) a PLAIN-ben és az MPLAIN-ben nincsenek definiálva, de más makrócsomagokban létezhetnek. (A matematikai módban használható parancsjeleket az 5. fejezetben, a táblázatkezelést vezérlő parancsjeleket a 6. fejezetben részletesen ismertetjük.)

MATEX A \- parancsjelet arra használjuk, hogy megadjuk hol választható el a szó. Például: `el\-vá\-lasz\-tás`. A TEX automatikus elválasztási algoritmussal is rendelkezik, amely a MATEX programcsomag segítségével a magyar szavakat is helyesen választja el. Néhány különlegesen bonyolult szó esetén mégis szükség lehet a \- parancsjelre. Az elválasztásról részletesen szólunk a 4. fejezetben. A \␣ parancsjellel utasítjuk a TEX-het, hogy az adott helyre szóközt tegyen. Mivel a TEX a forrásfile-ban levő szóközöket sokszor lenyeli (pl. ha egymás után több van), ezért külön meg kell adnunk, ha két szó között nagyobb távolságot, több szóköznyi helyet szeretnénk:

W. Shakespeare: A vihar. `W. Shakespeare:\ \ A vihar.`

Különösen fontos a \␣ parancsjel akkor, ha a TEX minden szóközt lenyel. Erre a következő pontban látunk példát.

2.2.2. Parancsszavak

Láttuk, hogy parancsjelből összesen 33 db lehet. A TEX formázó parancsainak tehát a túlnyomó többsége parancsszó. A parancsszó első karaktere a \ karakter. Ezt tetszőleges számú betű követheti. A \ karakteren kívül a parancsszó nem tartalmazhat más vezérlő karaktert, központozási, vagy egyéb jeleket, még számokat sem. A TEX megkülönbözteti a kis és nagy betűket a parancsszavakban is. Így például a \TeX parancsszó hatására a formázott

szövegben megjelenik a TEX szimbólum (logo), de ha azt írjuk \Tex, akkor hibaüzenetet ad, mert nincs definiálva ilyen parancsszó.

A legalapvetőbb parancsszavak a betűtípusok megadására szolgálnak. A legfontosabb betűtípusokhoz tartozó parancsszavak:

MTEX

- \ant „antikva" – antikva betűk kiírására szolgál. A szövegek nagy része, a mi könyvünk „folyó" szövege is ezt a betűtípust használja. Az MPLAIN-ben ez az alapértelmezés: ha más paranccsal nem állítjuk át, a betűk ilyen típusúak lesznek.
- \rm magyar ékezetes betűket nem tartalmazó antikva típusú betűk. A PLAIN-ben ez az alapértelmezés.
- \it „italic type"–kurzív betűk kiírására szolgál. *Szövegekben általában kiemelésre használják.*
- \sl „slanted"–dőlt antikva betűk kiírására szolgál. *Csak 30–40 éve terjedt el a használata.*
- \bf „bold face"–félkövér betűk kiírására szolgál. **Címeket, kiemelt szövegrészeket szednek vele.**
- \tt „typewriter type"–írógép típusú betűk kiírására szolgál. `Például számítástechnikai könyveknél hasznos.`

Ezeket a parancsszavakat úgy tekinthetjük, mint a kapcsolókat. Ha a szövegben valahol kiadunk egy \it parancsot, akkor attól kezdve minden betű kurzívval lesz szedve egészen addig, amíg más paranccsal felül nem bíráljuk. Kezdjük az elején antikva betűkkel, *majd átváltunk kurzívra,* néhány antikva betű újra, majd *döntjük a betűket, és végül* **itt áll néhány szó félkövéren.**

A parancsszavakat — a parancsjelekkel ellentétben — nem írhatjuk egybe az utána álló szavakkal. Például, ha azt akarjuk megjeleníteni, hogy ez így *nem jó*, nem írhatjuk ezt: `ez így \itnem jó`. Ekkor ugyanis a TEX egy \itnem parancsszót alkalmazna, ha lenne ilyen. A parancsjeleknél a TEX mindig „tudta" hol végződik a parancs, hiszen a \ karakter után mindig pontosan egy jel állt. A parancsszavak azonban tetszőleges hosszúak lehetnek; a TEX ezért úgy tekinti, hogy a parancsszó a \ karaktertől egészen az első szóközig, vagy más, nem betű karakterig tart. Ezért a parancsszavak után feltétlenül szóközt kell tennünk, ha a következő „szó" betűvel kezdődik.

Ha a parancsszó után valamilyen nem betűvek kezdődő karaktersorozat áll, ezt egybeírhatjuk a parancsszóval. Mivel a parancsszóban csak betű fordulhat elő, a TEX „tudja", itt a vége a parancsszónak.

```
Ilyenek a kurzív számok \it 2,
3, 4 \ant és a kurzív kérdőjel
\it? \ant az \& jel pedig ilyen
\it\&.
```

Ilyenek a kurzív számok *2, 3, 4* és a kurzív kérdőjel *?* az *&* jel pedig ilyen *&*.

A parancsszavakat — ugyanúgy, mint a parancsjeleket — az *előttük álló* szavakkal, vagy más karaktersorozattal mindig egybeírhatjuk, nem kell közéjük szóközt tenni, mert a TEX a \ karakterről mindig felismeri, hogy valamilyen parancs következik. Így elérhetjük azt is, hogy egy adott szó belsejében emeljük ki a szükséges karaktereket.

```
\bf d\ant e\bf v\ant ice \bf i
\ant ndependent
```

device independent

Itt a \bf és a \ant parancsszavak után mindig szóköz áll, a TEX mégsem tesz szóközt. Írhattuk volna azt is: \bf i\ant ndependent A TEX tehát mindig figyelmen kívül hagyja, hány szóköz áll a parancsszó után, ezekből egyetlen egyet sem szed ki a formázott szövegben. Ha ez nem így volna, sehogyan sem tudnánk leírni, hogy középső mert a parancsszó után kötelezően előírt szóköz megjelenne, és a szó így nézne ki:

köz épső.

Sokszor arra is szükségünk van, hogy a parancsszó utáni szóköz a formázott szövegben is megjelenjen. Hiába írjuk, hogy:

```
a TEXegy szövegszedő prog-
ram.                        a \TeX egy szövegszedő program.
```

a \TeX parancsszó utáni szóköz eltűnik, és így egybe írja a TEX-het az utána következő szóval. Ahhoz, hogy ez ne történjék meg, a \␣ parancsjelet kell használni:

```
a TEX egy szövegszedő prog-
ram                         a \TeX\ egy szövegszedő program
```

Ezzel utasítjuk a programot, hogy tegyen egy szóközt a két megjelenő szó közé. Vannak olyan parancsszavak is, amelyeknél a \ karaktert egyet-

len betű követi; ilyen például a § paragrafus jel előállítására szolgáló \S parancsszó, vagy az angol ábécében elő nem forduló betűk szedésére vonatkozó parancsszavak. Például:

```
z\l oty, fran\c caise,
{\rm F\ae r\o er}
```

złoty, française, Færøer

Bár a MTEX használatakor nincs rá szükség, megemlítjük, hogy az összes magyar ékezetes betű előállítható a TEX-ben levő parancsjelek ill. parancsszavak használatával. A \' ékezet segítségével készíthetünk á, Á, é, É, ó, Ó, ú, Ú betűket, a \H ('Hungarian umlaut') segítségével pedig

```
hosszú \H o; \H u bet\H uket
```

hosszú ő; ű betűket

A hosszú í betű elkészítése egy kicsit bonyolultabb. Ha ugyanis azt írjuk, hogy

```
ez \'igy nem jó,
```

ez így nem jó,

akkor furcsa betűképet kapunk, hiszen az i-re egy pont és egy vessző kerül. A TEX-ben két külön parancsszó szolgál arra, hogy az i ill. j betűkről a pontot eltávolítsa.

```
ez a ponttalan i: \i, ez pedig a
ponttalan j: \j
```

ez a ponttalan i: ı, ez pedig a ponttalan j: ȷ

A vesszőt ez után tehetjük az i-re így:

```
ez \'\i gy már jó
```

ez így már jó

A különleges betűket, jeleket előállító parancsokat táblázatban foglaltuk össze (csak egyetlen példát adtunk azokra a parancsokra, amelyek több betűre is alkalmazhatók):

parancs	megjelenő betű	parancs	megjelenő betű
\ae	æ	\b k	k̲
\AE	Æ	\c c	ç
\oe	œ	\d h	ẖ
\OE	Œ	\l	ł
\aa	å	\L	Ł
\AA	Å	\dag	†

\ss	ß	\ddag	‡
\o	ø	\S	§
\O	Ø	\P	¶
\~\i	ĩ	\it\$	£
\H u	ű	\it\&	&
\t\i u	i͡u	\copyright	©
\dots	...		

A felsoroltakon kívül még sok jel, szimbólum jeleníthető meg a TEX-hel, de ezek csak matematikai módban használhatók, így az 5. fejezetben írjuk le őket.

Ha MATEX programcsomagot használunk, a billentyűzet átdefiniáló program segítségével begépelhető ékezetes betűket a TEX megérti, ezért nem kell parancsjeleket, parancsszavakat alkalmaznunk. A TEX az ékezetes betűket úgy állítja elő, hogy az alapbetűt és az ékezetet egymásra teszi. A MATEX PXL file-jai ezzel szemben minden ékezetes betűt mint önálló betűképet tartalmaznak. Ennek következtében néhány — az eredeti PXL file-ban meglévő — betűkép itt hiányzik. A MATEX használatakor például az \ae parancs hatására nem az æ, hanem az æ betű jelenik meg. Ennek az az oka, hogy az „ékezetesített" PXL file-ban az æ betű helyére az Ő került. Ez azonban nem jelenti azt, hogy MATEX-hel nem tudunk æ betűt nyomtatni. Ahhoz, hogy ezt megtegyük, a MATEX alapértelmezéseként használt \ant (magyar ékezetes antikva) betűtípust az \rm betűtípusra kell változtatnunk.

	a következő betűk \rm \o, \O, \ae,
a következő betűk ø, Ø, æ,	\AE, \OE \ant is elérhetők
Æ, Œ is elérhetők MATEX-hel	\maTeX-hel

A többi betűtípusban azonban (\it, \sl, \bf) ezeket a különleges betűket nem lehet ilyen könnyen elérni. Az eredeti PXL file-okhoz tartozó fontot kell definiálnunk, amelynek módját ebben a fejezetben részletesen ismertetjük. A MATEX a következő 19 betű képét írja felül a PXL file-okban:

ASCII kód (dec.)	Eredeti betű vagy jel	MATEX betű vagy jel
0	Γ	"
1	Δ	Á
2	Θ	á
3	Λ	é

4	Ξ	É
5	Π	í
6	Σ	ó
7	ϒ	Ó
8	Φ	ö
9	Ψ	Ö
10	Ω	ő
26	æ	Ő
27	œ	ú
28	ø	ü
29	Æ	Ü
30	Œ	ű
31	Ø	Ű
60	¡	Í
62	¿	Ú

2.3. Környezetek kijelölése

Bár szavak, szövegrészek kiemelésére általában kurzív jeleket használnak, szükség lehet egyes szavak, betűk aláhúzására is. Az aláhúzásra szolgáló parancsszó az \underbar. Nézzük, mi történik, ha több szót akarunk aláhúzni:

Ezek a szavak nagyon fontosak.

```
Ezek a szavak \underbar
nagyon fontosak.
```

Láthatjuk, hogy csak az \underbar parancsszó után álló első nem szóköz karakter van aláhúzva. Ez a parancsszó tehát alapvetően másképp működik mint pl. az \it parancs, hiszen az utóbbinál a bekapcsolástól kezdve minden betű kurziválódott. Ahhoz, hogy kijelöljük azt a tartományt, érvényességi kört, ahol a parancsot érvényesíteni szeretnénk, { } jeleket kell használnunk.

Ezek a szavak nagyon fontosak

```
Ezek a szavak \underbar
{nagyon fontosak}
```

A példában írhattuk volna ezt is: `\underbar{nagyon fontosak}` Hatása ugyanaz, hiszen akkor, ha a parancsszó utáni első karakter nem betű, egybe lehet írni a parancsszóval. Ha az aláhúzandó szövegrész több szóból áll, a TEX a köztük lévő szóközöket is aláhúzza, így táblázatok fejlécének elkészítésére is használható. Ha a szavakat különállóan akarjuk aláhúzni, minden egyes szóra külön kell alkalmaznunk az `\underbar` parancsszót.

Egy fontos megjegyzés az aláhúzásról: az aláhúzott szavakat a TEX nem választja el, sőt az aláhúzott szövegrészt akkor sem töri két sorra, ha több szóból áll.

A { } környezet kijelölő karaktereket a betűtípust változtató parancsszavak esetén is használhatjuk. Ha például csak egy szövegrészt akarunk kurzívval kiemelni ezt írhatjuk:

Minden *amit a TEX készít* csodálatos

```
Minden {\it amit a \TeX\ készít}
csodálatos
```

Az Olvasó most joggal kérdezheti, hogy miért nem így emeltük ki:

Minden *amit a TEX készít csodálatos*

```
Minden \it {amit a \TeX\ készít}
csodálatos
```

Az utóbbi esetet a TEX úgy éretelmezi, hogy átváltottunk kurzív betűtípusra, és ennek semmi köze sincs a { } környezetkijelölő jelekhez, így a további betűket, jeleket kurzívval szedi. Vannak tehát olyan parancsok, amelyek érvényességi körét úgy jelöljük ki, hogy a parancsot is a kapcsos zárójelek közé tesszük, és vannak olyan parancsok, amelyeknél a parancs után tett kapcsos zárójelekbe tesszük a parancs tárgyát. Például:

Ez itt *nagyon fontos*

```
Ez itt {\it nagyon fontos}
```

Ez itt nagyon fontos

```
Ez itt \underbar{nagyon fontos}
```

Korábban aTEX parancsait két csoportba osztottuk: parancsjelek és parancsszavak. Egy másik rendezőelv alapján feloszthatjuk őket aszerint, hogy a hatásuk csak a parancs után közvetlenül álló (egy vagy több, meghatározott számú) objektumra terjed ki, vagy a parancs után következő összes objektumra folyamatosan, egészen addig, amíg a parancsot fel nem oldjuk. Az objektum itt jelenthet egy karaktert, egy szót, vagy egy egész bekezdést. Ha

a környezetkijelölő kapcsos zárójelekkel több karaktert fogunk össze, egyetlen objektumnak számít. Például az \underbar parancsot — mint láttuk — két betűből álló objektumra is alkalmazhatjuk:

két o betű aláhúzva oo

```
két o betű aláhúzva \underbar{oo}
```

Miért mondtuk az előbb, hogy „egy vagy több"? Mert vannak olyan parancsok is, amelyek hatása csak a közvetlenül utánuk következő két vagy három stb., de mindenképpen a parancstól függő, meghatározott számú objektumra terjed ki. Például ilyen a „tie" ékezet, amely két betűt köt össze, két objektumra alkalmazódik.

o͡oh͡hí

```
\t oohhí
```

A hatásukban folyamatos parancsokat feloldhatja egy másik parancs, de legtöbbször hatásuk azért szűnik meg, mert valamilyen környezet véget ér.

A betűk méretét szabályozó parancsok hatásukban folyamatosak. Például a \hatant paranccsal azt adjuk meg, hogy mostantól fogva 6 pontos betűkkel írjunk. (Ez kissé nehezen olvasható, ezért térjünk vissza az eredeti betűnagysághoz.)

A szövegeket általában ritkán szedik hatpontos pincurka betűkkel

```
A szövegeket általában ritkán
{szedik \hatant hatpontos}
pincurka betűkkel
```

Az előbbi példában a { jellel jelöltük ki egy környezet kezdetét. Az első szóval 'szedik' bár a környezeten belül van, semmi sem történik, a \hatant parancs kiadása után levő szavakat viszont 6 pontos betűvel szedi a TEX. Mivel a parancsot a környezeten belül adtuk ki, a környezeten kívül levő szavakra nincs hatással.

Környezeteket tetszőleges mélységben egymásba is skatulyázhatunk.

Sok van mi *csodálatos*

```
Sok {\bf van \underbar
{mi {\it csodálatos}}}
```

(Vigyáznunk kell a nyitó- és zárójelek helyes párosítására, a TEX szövegeknél ez az egyik leggyakoribb hiba forrása.) A környezet kijelölő jeleket használhatjuk arra is, hogy letiltsuk a ligaturákat.

P. a maffia fia

```
P. a maf{f}ia {f}{i}a
```

Ekkor ugyanis az egymás mellett lévő f ill. i betűket külön szedi, nem készít belőlük ligaturát.

Láttuk, a parancsszavak után a TEX mindig „lenyeli" a szóközöket, ezért például a TEX szimbólum leírásánál is \␣ parancsot kell alkalmaznunk ha nem akarjuk, hogy a TEX-het egybeírja az utána álló szóval. A {} jelekkel ezt másképpen is megoldhatjuk:

...a TEX azért csak TEX marad

```
...a {\TeX} azért csak \TeX{}
marad
```

Fontos a { }, az úgynevezett „üres környezet" is, mivel a szedett szövegben nem jelenik meg, de minthogy ez is objektum, parancsok alkalmazhatók rá. A fenti példában az volt a szerepe, hogy leállítsa a TEX szóközöket lenyelő folyamatát. Kíváncsiak lehetünk egy parancs hatására akkor is, ha nincs megfelelő objektumunk amire alkalmazhatnánk. Például előállíthatunk egy magában álló ékezetet így: \'{}: ´

2.4. Paraméterek

A legegyszerűbb szövegformázó utasítás a TEX-ben az üres sor: hatására a TEX új bekezdést kezd. Azt azonban, hogy mekkora legyen a bekezdés mélysége, nyilván nem tudjuk ezzel a módszerrel megadni. A TEX a különböző formázásokhoz szükséges értékeket paraméterekben tárolja. A bekezdés mélységét a \parindent („paragraph indentation"=bekezdés mélysége) paraméter tartalmazza. Ezt az értéket, amelyet a PLAIN és az MPLAIN makrócsomagok kb. 7 mm-re állítanak be, a `\parindent=10 mm` paranccsal állíthatjuk át 10 mm-re. Ez formailag egy parancsszó: \ karakterrel kezdődik, utána áll közvetlenül a paraméter neve, majd az egyenlőségjel és az a mennyiség, amelyre a paramétert be akarjuk állítani. A paraméter után vagy az egyenlőség jel után, vagy akár a számérték után is hagyhatunk egy vagy több szóközt. Tehát a következő értékadatok is helyesek: `\parindent =10mm; \parindent= 10mm; \parindent = 10 mm` Ha viszont a mértékegységet — jelen esetben a mm-t — lehagyjuk, a TEX hiba-

üzenetet ad. Mértékegységként a TEX elfogad minden — a nyomdászatban használatos egységet. Ezek a következők:

nyomdászati mértékegység neve	rövidítése
pont	pt
pica	pc
inch (hüvelyk)	in
big point (nagypont)	bp
centiméter	cm
milliméter	mm
didot point	dd
cicero	cc
scaled point(=osztott pont)	sp

A mértékegységek egymáshoz való viszonyát egy táblázatban foglaltuk össze:

	1 pt	1 pc	1 in	1 bp	1 mm	1 dd	1 cc
pt	1	12	72,27	1,00375	2,84528	1,07	12,84
pc	0,08333	1	6,0225	0,08365	0,2371	0,0892	1,07
in	0,01384	0,16608	1	0,013889	0,03937	0,0148	0,17770
bp	0,99626	11,9552	72	1	2,8348	1,066	12,792
mm	0,35146	4,2175	25,4	0,3528	1	0,376	4,513
dd	0,93458	11,215	67,542	0,9381	2314,77	1	12
cc	0,07788	0,93456	5,6284	0,07817	0,2216	0,8333	1
sp	65536	786432	4736287	65781,76	186468	70124	841489

(A táblázatban nem tüntettük fel a cm-t, mert könnyen átváltható a mm-ből. Nem adtuk meg azt sem, hogy 1 sp hogyan váltható át a többi mértékegységre.)

A mértékegységek közül Amerikában a pontot és a hüvelyket használják, ezért a különböző makrócsomagokban pt vagy in jelekkel adják meg a paraméterek értékeit. (Magyarországon a ciceró terjedt el leginkább.) A PLAIN-ben és az MPLAIN-ben `\parindent=20pt` áll.

Másik két fontos paraméter a tükörméret megadására szolgáló `\hsize` és `\vsize` paraméterek. (A `\hsize` a „horizontal size"= vízszintes méret, a `\vsize` a „vertical size"= függőleges méret szavakból származnak.) Egy könyv B5 - ös formátumát tehát így adhatjuk meg: `\hsize=126mm`, `\vsize=189mm`. Egy paraméternek nem csak úgy adhatunk értéket, hogy

az egyenlőségjel után egy konkrét számot írunk. Ha egy paraméternek már van értéke, felhasználhatjuk más paraméterek értékadásánál, például a \vsize=\hsize értékadással egy négyzet alakú (126mmx126mm-es) tükörméretet kapunk.

Az értékadásnál az egyenlőség bal oldalán is ki kell írnunk a \ karaktert, ugyanis a TEX erről ismeri fel, hogy itt \hsize paraméterről, és nem pusztán hsize karaktersorozatról van szó. Fontos tudnunk, hogy ezek a paraméterek nem csak a számértéket, hanem a mértékegységet is „megjegyzik". A mértékegységet akkor is jelölnünk kell, ha 0-át adunk meg. Például: \parindent=0mm

Vannak olyan paraméterek, amelyekhez nem tartozik mértékgység. Ezek közül a legfontosabb a \magnification. Ezzel adhatjuk meg azt, hogy az egész szöveget ki akarjuk nagyítani. Például ha \magnification=1200 parancsot adunk ki a file elején, akkor az egész dokumentumot a TEX 1,2 szeresére nagyítja. A parancsot mindig a file legelején kell kiadnunk és egy file-ban csak egyszer adhatunk értéket a \magnification paraméternek. Az egyenlőségjel után mindig a kívánt nagyítás 1000 szeresét kell írni. Ennek megfelelően a TEX szokásos nagyításait az 1000, 1095, 1200, 1440, 1728, 2074, 2488 számokkal érhetjük el. Nem kell ezeket a számokat feltétlenül megjegyeznünk: a TEX-ben van hét parancs \magstep0, \magstephalf, \magstep1, \magstep2, \magstep3, \magstep4, \magstep5, amely ezeket a számértékeket tartalmazza. Ha például a \magnification=\magstep2 parancsot adjuk ki, a \magnification értéke 1440 lesz.

Tudjuk, hogy a parancsok nevében csak betűk fordulhatnak elő (egyébként ez ugyanúgy vonatkozik a paraméterekre is). A \magstep3 karaktersorozatból csak a \magstep tartozik a parancs nevéhez, az utána álló szám pedig egy változó, amely kijelöli, hogy a \magstep parancs melyik „rekeszére" vagyunk kíváncsiak. Mivel a \magstep egy parancs, és nem paraméter, értékét nem írhatjuk felül:

1728=1988 \magstep3=1988

A TEX úgy szedi ki a szöveget, mintha a \magstep3 helyén annak értéke (1728) állna, utána pedig kiszedi a begépelt = jelet és a számot. A \magnification paraméter beállításával elért nagyítás pontosan olyan hatású, mintha fényképészeti eljárással állítottuk volna elő: mindent (betűket, szóközöket, tükörméretet) arányosan kinagyít. Például ha a sor hosszát

126 mm-re állítottuk be és 1,2 szeresére nagyítunk, a sor hossza 1,2x126 mm=151,2 mm lesz.

Vannak olyan esetek, amikor a kiszedett szöveg valamilyen méretét minden nagyítás mellett meg szeretnénk tartani. Ilyen például az, ha adott méretű ábrát, képet szeretnénk a szövegben elhelyezni. Lehetséges, hogy még nem döntöttük el, mekkora betűméretet, sortávolságot, stb. szeretnénk, de tudjuk, hogy a sor hossza mindenképpen — mondjuk — 90 mm lesz. A TEX-ben lehetőségünk van arra, hogy ezeket a „valódi" méreteket megadjuk. Ha a `\hsize=90 true mm` beállítást használjuk[3] az alkalmazott nagyítástól függetlenül a sor hossza mindig 90 mm lesz. Ez természetesen azt eredményezi, hogy egyre nagyobb nagyításoknál egyre kevesebb betű fér majd el egy sorban: Például ha a `\hsize=90 true mm` beállítást használjuk, 10 pontos nagyságban ilyen a szedett szöveg:

Ez a példaszöveg, a sorhossza mindig ugyanannyi, 90 mm,
de a nagyítás változik, és így egyre több sorra van szükség.

Majd 1,093 szeresére nagyítva:

Ez a példaszöveg, a sorhossza mindig ugyanannyi, 90
mm, de a nagyítás változik, és így egyre több sorra
van szükség.

Minden mértékegységhez használhatjuk a `true` kiegészítést, tehát helyesek a következő parancsok: `\vsize=10 true in`, `\parindent=20 true pt`. A `true in`, `true cm`, `true cc` ...kifejezéseket mint külön mértékegységet tekinti a TEX.

A tükörméret vagy a bekezdési mélység például egy könyvön belül mindig pontosan ugyanakkora, hacsak ennek megváltoztatására külön utasítást nem adunk. Vannak azonban olyan jellemzői a dokumentumoknak, amelyek — ha nem is sokat — oldalról oldalra változhatnak. Ilyen például a sorok közötti távolság. Mivel a különböző oldalakon eltérő számú bekezdés, ábra, táblázat ...stb. lehetséges, ahhoz, hogy a megadott tükörméret kijöjjön, a sorokat néha közelebb, néha távolabb kell szedni egymástól. Alapvető paraméterként megadhatjuk a sorok közötti távolságot, de ez az érték nem rögzített — mint például a sorok hossza — hanem „rugékony"[4], bizonyos

[3] A „true"(=igaz) angol szó, a „valódi" méretre utal

[4] Az angol nyelvű TEX szakirodalomban a „glue"=ragasztó elnevezés honosodott meg az ilyen típusú paraméterekre.

értékek közt változhat. A TEX számára nem csak azt adhatjuk meg, hogy mennyi legyen a sorok közti távolság, hanem számértékekkel jelezhetjük, hogy mennyire rugalmasan lehet kezelni ezt az értéket. Ennek megadására egy külön paramétertípus szolgál. Láttunk már példát mértékegységgel rendelkező paraméterekre (például `\hsize`). A sortávolság megadására szolgáló `\baselineskip` paraméter megadásához 3 számértékre van szükség:

- a sorok közti távolságra
- a kinyújthatóság mértékére
- az összenyomhatóság mértékére

A paraméter megadásának formája a következő: `\baselineskip=2cc plus 1cc minus 1cc`. A sorok távolságát ciceróban adtuk meg: ez átlagosan 2 ciceró, de ha a lap tördelése megkívánja legfeljebb 1 ciceróval lehet több és 1 ciceróval lehet kevesebb. Az ilyen beállítással elég nagy szabadságot adtunk a TEX-nek ahhoz, hogy az oldal képe mindig szép legyen, az esetleges fattyúsor, ábrák, táblázatok elhelyezéséből adódó tördelési problémákat megoldja.

Ugyancsak „rugalmas" típusú paraméter a `\parskip`. Ezzel az új bekezdéseknél lévő soremelést szabályozhatjuk. A most kezdett új bekezdéshez tartozó paraméterbeállítás: `\parskip=...` A `\parskip` által megadott távolság hozzáadódik a `\baselineskip` paraméter értékéhez ha új bekezdés kezdődik. Ezeket a paramétereket megadhatjuk úgy is, hogy csak a távolságot — az első értéket — adjuk meg, a többit nem. Ekkor a TEX a nyújthatóság ill. összenyomhatóság mértékét 0-nak veszi, azaz úgy gondolja, hogy a megadott távolság nem változhat. Ha címek, szövegkiemelések céljára valamennyi helyet szeretnénk kihagyni, használhatjuk a `\vskip` paramétert. Ha mondjuk fontos, hogy egy sor külön álljon, akkor a következő parancsot kell kiadnunk:

```
... \vskip 3mm
A kiemelés rávilágít a szöveg mély
értelmére, vagy éppen a sekélységére. \vskip 3mm
```

Ez a többi szövegtől elkülönülten, így jelenik meg:

A kiemelés rávilágít a szöveg mély értelmére, vagy éppen a sekélységére.

A példában meghatározott 3–3mm nem rugalmas: sem összenyomni, sem széthúzni nem lehet.

Szövegrészek elkülönítésére, sorok közötti nagyobb helyek kijelölésére a PLAIN és az MPLAIN makrócsomagban 3 parancsszó van: a `\smallskip`, `\medskip` és a `\bigskip`[5]. Ezeknek a parancsszavaknak hatására kis, közepes és nagy helyek maradnak ki:

ez egy kis hely, `\smallskip` volt,

ez már közepes helykihagyás, `\medskip` volt,

ez pedig nagy helykihagyás, `\bigskip` volt.

Ezeket a parancsszavak által létrehozott távolságokat a PLAIN és az MPLAIN makrócsomag rugalmasnak definiálja, tehát például ha a `\medskip` parancsot különböző helyeken többször is kiadjuk, az általa létrehozott üres helyek nem lesznek feltétlenül egyformák. Az eddig tárgyalt „rugalmas" paraméterek ill. parancsszavak kivétel nélkül a függőleges távolságok mértékének megadására szolgáltak. A vízszintes távolságok megadása hasonló módon történik: A `\vskip` paraméter vízszintes változata a `\hskip` paraméter. Arra használhatjuk, hogy egy sorban lévő szavak között a normálistól eltérő szóközt szedjen a TEX. Például ebben a sorban egy szó elkülönül a többitől. Az előbbi mondat a forrásfile-ban ilyen alakú:

```
Például ebben a sorban
\hskip 1cm plus 0.5cm minus 0.5cm egy
\hskip 1cm plus 0.5cm minus 0.5cm szó elkülönül a többitől.
```

Láthatjuk, hogy a számok megadásánál tizedestört értékek is szerepelhetnek. Ez a megállapítás igaz mindenféle paraméter értékadására. Fontos, hogy a tizedestörtben *tizedespontot* használjunk — az amerikai szokásoknak megfelelően — és ne tizedesvesszőt. A tizedespont előtti jegyet vagy jegyeket, ha azok mind 0-k nem kell leírni. A `\hskip` paramétert folyamatos szövegben ritkán használjuk, jelentősége bonyolult matematikai képletek szedésénél van. A `\hskip` paraméternél jóval egyszerűbb az `\enskip` parancsszót használni. Ez a parancs szintén arra szolgál, hogy a szavak közt bizonyos mértékű távolságot adjunk meg vele, de ez a távolság rögzített és

[5] Az elnevezések a 'small skip'(=kis ugrás); 'medium skip'(=közepes ugrás); 'big skip'(=nagy ugrás) angol kifejezésekből származnak.

nem „rugalmas". Az \enskip parancsszó által létrehozott távolságot azonban nem tudjuk az eddig bemutatott mértékegységekkel megadni, ez ugyanis betűtípusonként változik.

Sok szerkesztési, tördelési probléma miatt szükséges, hogy ne rögzített mértékegységben előírt távolságokat tudjunk megadni, hanem relatív, a betűtípustól függő távolságokat. A nyomdai gyakorlatban meghonosodott, hogy egy adott betűkészlet kis x betűjének magasságát és a kis m betű szélességét mint az adott betűtípusra jellemző értéket mértékegységként használják. Így megadhatunk 2x magasságot vagy 3m szélességet. A TEX ezekre a mértékegységekre az ex és az em elnevezést használja. Az \enskip parancsszó egy 0,5 em szóköznek — körülbelül egy 'n' betű szélességének — megfelelő távolságot hagy ki az adott helyen:

Például ezen s z ó betűi között 0,5em távolság van	`Például ezen s\enskip z\enskip ó betűi között 0,5em távolság van`

Ez a mértékegység különösen akkor hasznos, ha még nem tudjuk, hogy a szöveget milyen betűtípussal állítjuk elő, vagy ha sokféle betűtípust használunk. A TEX az ex, em mértékegységekben megadott helyet úgy számítja ki, hogy az éppen érvényben lévő betűtípus x, m betűjét veszi alapul. Kényelmesen használható a \baselineskip, \parskip, \parindent paraméterek megadásánál is.

2.5. Számlálók

A paraméterek egy külön osztályát alkotják a számlálók. Értékük egy egyszerű egész szám (lehet negatív is). Jellemzőjük, hogy valamilyen automatikus szedési, tördelési eljárás számértékeinek tárolására, beállítására szolgálnak. Legfontosabb számláló a \pageno, amely az oldalszámot tartalmazza.[6] A TEX, ahogy egymás után szedi ki az oldalakat, mindig növeli eggyel ennek a számlálónak az értékét. A számlálót azonban a felhasználó is beállíthatja. Nagyobb dokumentumok, például könyvek szedésénél általában több részből — mondjuk fejezetből — állítjuk össze az egész könyvet. Ekkor meg kell

[6] A számláló neve az angol „page no.— number"(=oldalszám)-ból származik.

adnunk a TEX-nek, hogy a 3. file (3. fejezet) elkészítésekor honnan kezdje a lapszámozást. Erre szolgál a \pageno=52 paraméterbeállító parancs. A TEX a file-ban lévő oldalakat 52-től kezdve fogja számozni. A PLAIN és az MPLAIN makrócsomagok használatakor az oldalszámozás az oldal alján, középen jelenik meg. Ha nem akarjuk, hogy az oldalak számozottak legyenek, a file legelején ki kell adnunk a \nopagenumbers parancsot. Ha azt szeretnénk elérni, hogy az oldalszámozás máshová kerüljön, ne a lap aljára, akkor a \folio parancsot kell használnunk. A parancsszó hatására kiíródik az oldalszámozást tartalmazó számláló értéke oda, ahol a parancsot kiadtuk. Ez lehet a lap tetején, a jobb ill. bal oldalon vagy a lap alján stb. Erről részletesen a 4. fejezetben, a fejsorok ill. lábsorok ismertetésénél lesz szó.

2.6. Betűkészletek (fontok) megadása

A fejezet elején láttuk, hogy hat alapvető betűtípust érhetünk el az MPLAIN használatakor. Ezek: eredeti antikva, magyarított antikva, *dőlt*,*kurzív*, **félkövér**, írógép típusú betűk. Ha már installáltuk a TEX ill. MATEX programunkat láthatjuk, hogy ezenkívül sokféle TFM file áll rendelkezésünkre. Ezek jelentős része nem betűket, hanem jeleket, szimbólumokat tartalmaz, de a felsoroltakon kívül több betűtípus is elérhető. Ilyen például az AMSS „almost modern sans serif" , az ún. groteszk betűcsalád, amely az egyik leggyakrabban használatos betűtípus. Ahhoz, hogy használni tudjuk ezt a betűtípust is, először definiálnunk kell. Ezt a \font parancsszóval tehetjük meg:

```
\font\sanss=AMSS10
```

A \font parancsszó után áll a betűkészlet neve, ahogy majd hivatkozni akarunk rá, majd az egyenlőségjel másik oldalán az a file név, amelynek .TFM ill. .PXL kiterjesztésű file-jaiban a megfelelő betűtípus megtalálható. A font nevet szabadon választhatjuk meg — mi most a \sanss nevet választottuk — csak arra kell ügyelnünk, hogy ez a név ne essen egybe egy már korábban definiált parancsszóval. A \font paranccsal egy parancsszót definiáltunk, ezért a parancsszó általános megkötései a fontnevekre is érvényesek. Nem definiálhatjuk ezért a 10 pontos sanserifet így:\font\10pontossanss=AMSS10, mert

a parancsszavakban nem fordulhatnak elő számok. Mondhatjuk az előbbi helyett ezt: `\font\tizpontossanss=AMSS10` A `\font\` parancs kiadásakor a TEX program betölti a memóriába a fonthoz tartozó TFM file-ból a betűkre vonatkozó információkat. Az MPLAIN-ben összesen 16 TFM file töltődik be automatikusan anélkül, hogy parancsot adnánk rá. Ezek a következők:

parancs neve	a fontfile neve	a definiáló \font utasítás	megjegyzés
\tenrm	AMR10	\font\tenrm=AMR10	tíz pt, roman
\sevenrm	AMR7	\font\sevenrm=AMR7	hét pt, roman
\fiverm	AMR5	\font\fiverm=AMR5	öt pt, roman
\tenit	AMTI10	\font\tenit=AMTI10	tíz pt, kurzív
\tenbf	AMBX10	\font\tenbf=AMBX10	tíz pt, félkövér
\sevenbf	AMBX7	\font\sevenbf=AMBX7	hét pt, félkövér
\fivebf	AMBX5	\font\fivebf=AMBX5	öt pt, félkövér
\tensl	AMSL10	\font\tensl=AMSL10	tíz pt, dőlt
\tentt	AMTT10	\font\tentt=AMT10	tíz pt, írógép
\tizant	HAMR10	\font\tizan=HAMR10	tíz pt, antikva
\hetant	HAMR7	\font\hetant=HAMR10	hét pt, antikva
\otant	HAMR5	\font\otant=HAMR5	öt pt, antikva
\teni	AMMI10	\font\teni=AMMI10	tíz pt, mat. betűk
\seveni	AMMI7	\font\seveni=AMMI7	hét pt, mat. betűk
\fivei	AMMI5	\font\fivei=AMMI5	öt pt, mat. betűk
\tensy	AMSY10	\font\tensy=AMSY10	tíz pt, mat. szimb.
\sevensy	AMSY7	\font\sevensy=AMSY7	hét pt, mat. szimb.
\fivesy	AMSY5	\font\fivesy=AMSY5	öt pt, mat. szimb.
\tenex	AMEX10	\font\tenex=AMEX10	tíz pt, egyéb jelek

Az alapértelmezés minden betűtípusnál a 10 pontos változat. A korábban megismert `\ant`, `\bf`, `\it` parancsokkal a megfelelő, éppen érvényben lévő betűnagyságot „kapcsoljuk" be. Mielőtt továbbmennénk, egy kicsit pontosabban meg kell határoznunk, hogy mit jelent az, hogy „tízpontos betű". Ezt a kifejezést ugyanis kétféle értelemben használjuk.

- Az egyik értelme a betű fizikai méretére vonatkozik. Egy betűtípusra akkor mondjuk, hogy tízpontos, ha a gondolatjel hossza és a () magassága 10 pont(=3,5146 mm)

- A másik értelme a betű típusára vonatkozik. Láttuk az első fejezetben, hogy az AMR5 betűi nem azonosak az AMR10 betűivel még akkor sem, ha az előbbit a kétszeresére nagyítjuk. Ebben az esetben az a kifejezés, hogy 5 pontos ill. 10 pontos a betű tervezett nagyságát jelenti, tehát azt a méretet, amire a betűt tervező gondolt[7]. Valóban, például az AMR5 betűi 5 pont nagyságú betűk szedésére a legalkalmasabbak.

Az előzőekben láttuk, hogy van arra is lehetőség, hogy egy 10 pontos nagyságú betűből a `\magnification` parancs segítségével akár 20 pont nagyságú betűket szedjünk ki. Az ott leírt eljárás arra nem használható, hogy címeket, kiemelendő szövegrészeket nagyobb betűvel szedjünk, mert a `\magnification` hatására az egész szöveg „kinagyítódik". A `\font` paranccsal azonban definiálhatunk „nagy" betűket is a következőképpen: `\font\nagykurziv=AMTI10 at 20.74pt`. A parancsban egy `\nagykurziv` betűtípust definiáltunk, kb. 2-szeresére kinagyított 10 pontos kurzív betűkből áll. A font definiálásánál az `at`[8] a TEX egyik *kulcsszava* ugyanúgy, mint a `\vskip`, `\hskip` paraméterek megadásánál a `plus` és a `minus`. Az `at` után álló szám és mértékegység azt mondja meg, hogy a megjelölt betűtípust mekkora (fizikai) nagyságban akarjuk használni. Eszerint a `\nagykurziv` paranccsal *20. pt magasságú betűket* írhatunk. A magasság megadásánál a pt helyett használhatunk bármely más, korábban bemutatott mértékegységet az `em`, `ex` kivételével. Természetesen ahhoz, hogy a megfelelő betűket ki is tudjuk nyomtatni, rendelkeznünk kell a megfelelő nagyítást tartalmazó PXL file-lal.

A nagyítások megadására más lehetőségünk is van. Ha nem a betű méretét, hanem a nagyítás arányát akarjuk megadni a betűtípus definiálásakor, az `at` kulcsszó helyett a `scaled`[9] kulcsszót használjuk. Az előbbi betűtípust definiálhatjuk így is: `\font\nagykurziv=AMTI10 scaled\magstep4`. Láttuk, hogy a `\magstep4` változó értéke 2074, azaz a pontos nagyítás 2.074-szeres. Figyeljünk arra, hogy például a mátrixnyomtatók 240 pont/inch felbontsú pixelfile-jai esetén ezek a 20 pontos (pontosabban 20.74 pontos) betűk egy olyan file-ban vannak, amelyik nevében nem a 2074, hanem a 2488 számot tartalmazza. A 2074-es megjelölésű file ezzel szemben a 17 pontos be-

[7] Angolul „design size"= tervezési méret.

[8] Az at magyarul -nál,-nél-t jelent.

[9] scaled=skálázott

tűket tartalmazza. Ennek oka, hogy a tízpontos betűk is már egy, a nevében 1200-at viselő file-ba kerültek. Erről a kérdésről bővebben az A. függelékben írunk majd. Itt jegyezzük még meg, hogy ugyanazt a betűtípust több, más és más nagyságban is használhatjuk. Például ugyanazon szöveg előállítására a 10 pontos kurzív betűket 3 nagyságban is definiálhatjuk.

Gyakran előfordul, hogy egy szöveg szedéséhez néhány betűtípust nagyobb méretben is felhasználunk, és az egész szöveget egy `\magnification` paranccsal nagyítjuk is. Ekkor a kinyomtatott betűk nagyítása a kétféle nagyítás szorzata lesz. Ha például `\magnification=\magstep1` nagyítást alkalmazunk, az előbbi `\nagykurziv`-val szedett betűk akkorák lesznek, mintha `\magstep5`-tel definiáltuk volna őket. (Ilyen nagyítás a legtöbb esetben nem is áll rendelkezésre.) Gyakori hiba, hogy `\magnification` paranccsal nagyítunk olyan szöveget, amelyben `\magstephalf` nagyítással definiált font van. Ekkor a kinyomtatásnál a nyomtatóprogram nem találja meg a megfelelő PXL file-t, mivel azok, az egyetlen 1.095-szörös nagyítás kivételével, mind az 1.2 hatványainak megfelelő nagyításokat tartalmazzák. Ha tehát a következő nagyításokat alkalmazzuk:

```
\font\közepes kurziv=AMTI10 scaled \magstephalf
\magnification=\magstep1
```

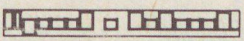
akkor a nyomtatóprogram az AMTI10 font 1.095x1.2=1.315-szörös nagyítását keresi, ami általában nem áll a felhasználó rendelkezésére.

2.7. Néhány szó a dobozokról

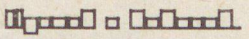
A TEX a forrásfile-ból annyit olvas be, amennyiből egy oldalt el tud készíteni. Ha az oldal elkészült, azon már nem változtat, még akkor sem, ha a később feldolgozott szöveget esetleg szebben lehetne tördelni, ha az előző oldalt pl. ritkábban szedte volna. Így tehát a TEX számára a legnagyobb tördelési egység az oldal. Az oldal azonban sok kisebb egységből, TEX elnevezéssel *dobozból* épül föl.

A legkisebb, „elemi" dobozok a betűket, jeleket befoglaló dobozok, amelyekről az 1. fejezetben már volt szó. Ha a betűket egymás mellé rakjuk,

alacsony, széles dobozt, egy sort kapunk; például az „Ilyenek a dobozok." mondat ahogy a TEX „látja":



Figyeljük meg, hogy a TEX a betűknek megfelelő dobozokat



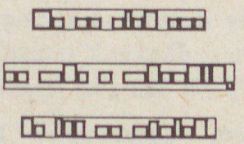
egy nagyobb, minden betűdobozt magában foglaló dobozba teszi.

Hosszabb, rövidebb sorokat egymás alá rakva fölépíthetünk nagyobb dobozokat, végül a legnagyobb dobozt, az oldalt. Dobozt alkotnak a bonyolult matematikai képletek, de az ábrák számára kihagyott üres hely is doboz.

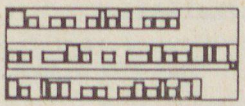
A TEX a feldolgozás során mindig a kisebb dobozoktól a nagyobbak felé halad. Először a sorokat, mint dobozokat készíti el, arányosan elosztva a betűket — dobozokat — és az üres helyeket a soron belül. Ha ezzel elkészült, minden sort egyetlen — belül már nem változtatható — doboznak tekintve felépíti belőlük az oldalt úgy, hogy arányosan elosztja a sorok közti üres helyeket. Ha ábrák, matematikai formulák, kiemelt szövegrészek is vannak, a TEX ezeket is mint dobozokat tekinti, és a sorok dobozaival együtt dolgozza fel őket. Nézzük például a következő, rövid bekezdést hogyan állítja össze a TEX:

Ez az első sor
ez már a második,
és itt az utolsó.

Először az első, második sort, majd a harmadik sort, mint dobozt állítja össze:



A soroknak megfelelő dobozokból azután összeállítja a szövegrészt:



A legtöbb esetben a TEX automatikusan állít össze egy dobozt alkotóelemeiből, de különleges esetekben a felhasználó TEX paranccsal is létrehozhat dobozokat, pontosan megadva, hogy milyen elemekből épüljön föl.

A TEX kétféleképpen tud kisebb dobozokból egy nagyobbat fölépíteni: vagy *vízszintesen* rakja a dobozokat egymás mellé, vagy *függőlegesen*, egymás tetejére helyezi a dobozokat. Ennek megfelelően két parancsot használhatunk:

- a \hbox parancsot, amellyel vízszintesen,
- és a \vbox parancsot, amellyel függőlegesen

építkezhetünk[10]. A parancs után kapcsos zárójelek között adjuk meg azokat az elemeket, amelyekből dobozt akarunk készíteni. Az elemek lehetnek betűk, jelek, de lehetnek más dobozok is; a parancsokat egymásba is skatulyázhatjuk. Az előbbi példát így adtuk meg:

```
\vbox{\hbox{Ez az első sor}
\hbox{ez már a második,}
\hbox{és itt az utolsó.}}
```

Az előbbi példánkban a TEX automatikusan számolja ki a dobozok magasságát és szélességét. Ha ábrát helyezünk el a szövegben, a kihagyandó üres helyet is ezekkel a parancsokkal adhatjuk meg. Ekkor azonban a doboz üres, ezért mindkét mérete 0 lesz. A feladatot nem oldhatjuk meg úgy sem, hogy a már megismert \hsize és \vsize parancsokat a dobozok belsejében helyezzük el. A doboz méretét a következőképpen adhatjuk meg:

```
\hbox to 34mm{ ... }
```

A parancs után álló to szó *kulcsszó*, a parancshoz tartozik. Ha a TEX a \hbox vagy a \vbox parancs után ezt találja, akkor egy mértékegységgel rendelkező számot vár, amely megmondja, hogy a doboz mekkora legyen. A kapcsos zárójelek között megadhatunk tetszőlegesen szövegrészeket, más parancsokat is használhatunk stb. Az első fejezetben bemutatott példát:

Túlzott alkalmazása csúnya nyomtatási képhez vezethet.

így állítottuk elő:

```
\hbox to 110mm{Túlzott alkalmazása
```

[10] Az elnevezések a 'horizontal box'(=vizszintes doboz) ill. a 'vertical box'(=függőleges doboz) angol kifejezésekből származnak.

```
csúnya nyomtatási képhez vezethet.}
```

A \hbox és a \vbox parancsokkal előállított dobozt a TEX ugyanúgy kezeli, mint egy betűt. A szöveg tördelésénél, a sorokfeltöltésénél, a sorok közti távolság megállapításánál ugyanúgy veszi figyelembe a doboz méreteit, mint egy betűhöz tartozó doboz esetén. Például a következő szövegben egy 10 mm-szer 10 mm-es üres helyet hagyunk ki úgy, hogy egy dobozt készítünk:

```
Szöveg szöveg {\it szöveg} szöveg
\underbar{szöveg} szöveg {\bf szöveg}
.
.
szöveg szöveg \vbox to 10mm{\hbox to 10mm{}}
.
.
szöveg szöveg szöveg szöveg {\it szöveg} szöveg.
```

A TEX ezt így szedi ki:

Szöveg szöveg *szöveg* szöveg szöveg szöveg **szöveg** szöveg szöveg szöveg szöveg *szöveg* szöveg szöveg szöveg szöveg szöveg szöveg *szöveg* szöveg szöveg *szöveg*

szöveg szöveg *szöveg* szöveg szöveg szöveg szöveg szöveg *szöveg* szöveg szöveg *szöveg* szöveg szöveg *szöveg* szöveg szöveg szöveg szöveg szöveg *szöveg* szöveg szöveg szöveg szöveg szöveg *szöveg* szöveg

A dobozokról itt leírtak bővebb kifejtésben szerepelnek majd a 6. fejezetben, ahol az ábrák készítéséhez használjuk őket.

3. fejezet

Az első TEX szöveg előállítása

Ebben a fejezetben alapvető formázási parancsokat mutatunk be. Végigkövetjük egy dokumentum előállítását az első leütéstől a hibátlan szöveg kinyomtatásáig.

Érdemes ezt a fejezetet a számítógép előtt ülve olvasni — elsősorban kezdő, TEX-hel először ismerkedő felhasználókra gondolunk — és a bemutatott parancsokat, és így a hibaüzeneteket is, azonnal kipróbálni. A fejezet forgatókönyve annak a munkának, amellyel a szerzők a könyv hátsó borítóján lévő szöveget előállították.

3.1. Az első lépések

Az első dolog, amit a szöveg előállításakor megadunk: a tükörméret. Ez általában a könyv, dokumentum tervezésének legelején eldől. Könyvünk esetében a hátsó borítón levő „fülszöveg" ideális tükörméretére 126 mm-szer (sorhossz) 210 mm (magasság) adódott. Az első két parancs ennek alapján:

```
\hsize=126mm
\vsize=210mm
```

A „fülszövegben" nem szokás számolni az oldalakat, a TEX viszont szenvedélyesen számoz minden oldalt. Hogy ebben az esetben mégse tegye egy parancsot adunk ki:

```
\nopagenumbers
```

Az oldal tetejére, középre helyezve ki akarjuk írni, hogy „Magiszter Számítástechnikai Szerkesztség". Ehhez a \centerline parancsot használjuk, kapcsos zárójelek között a középre szedendő szöveget adjuk meg:

```
\centerline{Magiszter Számítástechnikai Szerkesztöség}
```

Azt szeretnénk, hogy a „Magiszter" név hasonlítson egy kicsit az embléma betűihez, így az „M" betűt egy kis „m" betűvel helyettesítjük, ám jelentősen felnagyítva. Ehhez először definiáljuk a nagy antikva betűket tartalmazó fontot:

```
\font\nnagyant=HAMR10 scaled\magstep3 %nagyon nagy antikva
%(1.728 szeresére nagyítva)
```

A „Magiszter" többi betűjét is egy kicsit nagyobb betűvel szedjük, ehhez definiáljuk az 1.095-szörös nagyítású fontot:

```
\font\felesant=HAMR10 scaled \magstephalf
```

Most már ki tudjuk írni úgy, ahogy a borítón szerepel:

```
\centerline{{\nnagyant m}{\felesant agiszter}
Számítástechnikai Szerkesztöség}
```

A sor után kb. 1 cm helyet szeretnénk kihagyni, hogy a főcím kellőképpen elkülönüljön. Ezért az előző parancs után a \bigskip parancsot adjuk ki:

```
.
.
\centerline{...}
\bigskip
```

A főcímet középre szedjük \centerline paranccsal, de előbb pontosan, betűről betűre meg kell adnunk a „logo"-t, a betűkből összerakott MTEX szimbólumot. Az M és A betűk 7 pontos antikva betűk 1.2^4=2.074 szeresre nagyítva, a TEX betűi 10 pontos antikva betűk ugyanekkora nagyításban:

```
\font\hatalmasant=HAMR10 scaled \magstep4 %a logo
\font\hetant=HAMR7 scaled \magstep4        % betűihez
```

A MATEX logoban az M és A betűket egy kissé közelebb húzzuk egymáshoz: 0.1 em[1] távolsággal, és a „MA" betűpárt 0.37 em távolsággal közelebb húzzuk a TEX logohoz, mint ahogyan a betűk normálisan egymás mellett állnának. A betűk közötti távolság vezérlése a \kern parancsot használjuk. A logot előállító parancsok:

```
{\hetant M\kern -.1em A}\hatalmasant\kern -.37em\TeX}
```

A „programcsomagot" a már definiált \nnagyant betűkkel szedjük, így a főcím:

```
\centerline{{\hhetant M\kern -.1em A}\hatalmasant\kern -.37em
\TeX\ \nnagyant programcsomag}
```

Könyvünkben sokszor szerepelnek logok. Ha minden alkalommal ilyen sok munkával kellene előállítani őket bizonyos, hogy mi sem használnánk a MATEX szimbólumot. Szerencsére a TEX-ben lehetőség van arra, hogy ezeket a szimbólumokat egyetlen paranccsal állítsuk elő. Az előbb a főcím megadásánál mi is felhasználtuk, hogy minden TEX programban létezik a \TeX parancsszó, amely ezt a szimbólumot előállítja. (A parancsszó \TeX; mi azért írtunk az előbb \TeX\␣-et, hogy a \␣ parancs egy szóközt iktasson a TEX és a „programcsomag" szó közé, ld. 2. fejezet) A fülszövegben többször is szerepel a MATEX logo, így kényelmesebb, ha most definiáljuk az ezt előállító \maTeX parancsot:

```
\def\maTeX{{\hetant M\kern -.1em A}\kern -.37em\TeX}
```

A definíció első parancsa a \def, amely jelzi a TEX-nek, hogy egy parancs definíciója következik. Utána áll a parancs neve \maTeX és azután kapcsos zárójelek közé zárva az a parancssorozat, amelyet a \maTeX parancs kiadásával végre kell hajtani. (A TEX logo definícióját a 2. fejezetben mutattuk be.) A forrásszövegben bárhol definiálhatunk parancsokat, de ezekre csak a definiálásuk után következő szövegrészekben hivatkozhatunk. A főcím után ismét kihagyunk kb 1 cm helyet, majd kezdődik a „folyószöveg". Ezt 'sans serif' betűtípussal szeretnénk szedni, ezért definiáljuk a \sanss fontot:

[1] Az em az érvényben lévő betűtípus kis m betűjének szélessége. Mértékegységekről részletesen a 2. fejezetben szóltunk.

```
.
.
\centerline{{\hhetant M\kern .....}
\def\maTeX{...}
\bigskip
\font\sanss=AMSS10
```

A TEX minden üres sor — a \bigskip is annak számít — után új bekezdést kezd. A bekezdés mértéke beállítható, és ha nem akarjuk, hogy beljebb kezdjen, használhatjuk a \noindent parancsot (mint a jelen esetben is). Ezután átváltunk a 'sans serif' betűtípusra, és kezdődhet a „folyószöveg":

```
.
.
.
\noindent
\sanss
A \maTeX\ programcsomag az IBM PC/XT/AT
kompatibilis számítógépeken      % a százalékjel a komment-jel.
futó PC\TeX\ és Micro\TeX,       % A jel után következő
valamint a Preview programokat   % karaktereket a \TeX\
teszi alkalmassá arra,           % nem veszi figyelembe, és
hogy magyar nyelvű szövegeket    % nem szedi ki, ezért
lehessen könnyen,                % ide írhatunk jegyzeteket.
gyorsan, szépen --- a \TeX\      % A --- jelekből lesz a
szövegszedő program minden       % gondlatjel.
lehetőségét felhasználva --- előállítani.
A programcsomag egyszerűen implementálható; futtatható
minden olyan számítógépen, amelyen a PC\TeX, a
Micro\TeX\ vagy a Preview program fut, tehát
semmilyen külön hardware igénye nincs. A \maTeX\
programcsomag az alábbi szolgáltatásokat tartalmazza:
```

Mielőtt továbbmennénk meg kell oldanunk egy fontos problémát. A fenti szövegrészben említettünk néhány törvény által védett elnevezést. Ezeknél a neveknél jeleznünk kell, hogy melyik cég, társulat védjegye. Tekintve, hogy

a lap alján, a többi szövegtől elkülönítve szokták megadni kézenfekvő, hogy erre a TEX lábjegyzetkészítő parancsát, a \footnote parancsot használjuk. A parancsnak két változója van; ezeket a parancs után { } zárójelek közé zártan kell megadni. Az első kapcsos zárójel közé azt a jelet kell tenni, amelyet a lábjegyzet jelölésére használunk, a második zárójelpár közé a lábjegyzet teljes szövegét. A \footnote parancsot közvetlenül a jelölt szó mögé kell írni. Ennek alapján a fenti szöveg:

```
.
.
.
A \maTeX\footnote{\dag}{\copyright \maTeX\ az
Akadémiai Kiadó védjegye} programcsomag az IBM
\footnote{\dag}{\copyright IBM az International
Business Machines Corp. jegyzett védjegye}
PC/XT/AT számítógépeken futó PC\TeX\footnote {\dag}{\copyright
PC\TeX\ a Personal...
.
.
```

A lábjegyzet jelölésére a †-jelet használjuk, amelyet a \dag parancs állít elő. Ha a fenti szöveget a TEX kiszedi, a lap alján a következő fog megjelenni:

† ©MATEX az Akadémiai Kiadó védjegye +
† ©IBM az International Business Machines Corp. jegyzett védjegye
† ©PCTEX a Personal...

Ez, az összes védjegyet felsorolva, túl sok helyet foglal el, másrészt a †-jel inkább zavaró mint hasznos. Ezért a borítón lévő szöveg készítésénél egyetlen lábjegyzetet használunk. Ez a lábjegyzet „jeltelen", vagyis a szó, amelyre a lábjegyzet vonatkozik nincs semmilyen jellel megjelölve, és ebben az egyetlen lábjegyzet szövegben az összes védjegyet felsoroltuk:

```
.
.
.
gyorsan, szépen,  ---  a \TeX\footnote{}{\copyright
```

```
IBM az IBM az International
Business Machines Corp. jegyzett védjegye;
\copyright\TeX\ az AMS (Amerikai
Matematikai Társulat védjegye; \copyright Micro\TeX\
az Addison-Wesley Publishing Company védjegye;
\copyright\maTeX\ az Akadémiai Kiadó védjegye}
```

Ezután a kitérő után folytassuk a szöveget. Most jön a felsorolás. Azt szeretnénk, hogy az egyes szolgáltatások beljebb kezdve, egymástól kicsit elkülönülve álljanak. A felsorolásra az \item parancs a legkényelmesebb eszköz. A TEX a parancs után kapcsos zárójelekben megadott jelet tesz ki minden egyes egység elé. Mi a ‡-szimbólumot választottuk, ezt a \ddag parancs állítja elő:

```
.
.
. ... az alábbi
szolgáltatásokat tartalmazza:
\item{\ddag} segítségével új, ékezetes betűket is
tartalmazó betűcsaládokat lehet készíteni
minden betűtípusban és nagyságban.
```

A TEX ezt így szedi ki:

az alábbi szolgáltatásokat tartalmazza:

‡ segítségével új, ékezetes betűket is tartalmazó betűcsaládokat lehet készíteni minden betűtípusban és nagyságban.

Láthatjuk, hogy a felsorolás nem különül el eléggé az előtte lévő szövegtől. A \parskip paranccsal beállíthatjuk, hogy minden új bekezdéskor a szokásos sortávolságot mennyivel növelje meg a TEX. A felsorolás első eleme előtt tehát ezt kell megadnunk:

```
.
.
.    ....az alábbi
szolgáltatásokat tartalmazza:
\parskip 4pt plus 1pt minus 1pt
```

```
\item{\ddag} Segítségével új, ...
```

A parskip parancs tehát 4 pt-tal megnöveli a sortávolságot, ha új bekezdés vagy felsorolási egység kezdődik. Az utána álló plus 1pt minus 1pt kifejezés arra ad lehetőséget a TEX-nek, hogyha az oldal tördelése úgy kívánja, ezt a helyet egy kicsit összenyomhatja vagy széthúzhatja.

A felsorolás első elemének második mondatát írógépszerű, nem proporcionális, tehát azonos szélességű betűkkel szeretnénk szedni. Ezért ezt a mondatot kapcsos zárójelek közé zárjuk, és ezen a környezeten belül a \tt paranccsal bekapcsoljuk az írógép típust.

```
...az alábbi
szolgáltatásokat tartalmazza:
\parskip 4pt plus 1pt minus 1pt
\item{\ddag} Segítségével új, ékezetes betűket is
tartalmazó betűcsaládokat lehet készíteni minden
betűtípusban és nagyságban. {\tt(Például
ékezetes, nem proporcionális betűtípust is, amely
a számítástechnikai könyvekhez szükséges.)}
\item{\ddag} Használatával...
```

Az újabb \item parancs kiadásakor a TEX tudja, hogy újabb felsorolási egység következik. Az előzőt leírja és sort emel, az első sort beljebb kezdi. Ebben a szövegrészben két új dolog bukkan fel: az aláhúzás és a felkiáltójel. Ha a forrásszövegben !-et gépelünk, akkor a kiszedett szövegben nem felkiáltójel, hanem í betű lesz. Ahhoz, hogy felkiáltójelet tudjon a TEX szedni a \? parancsot kell kiadnunk (nem tévedés, valóban a \? paranccsal állítjuk elő a felkiáltójelet ugyanis a \! parancs matematikai módban más jelentésű.)

Szövegrészt aláhúzni az \underbar paranccsal lehet. A parancs után kapcsos zárójelek közé kell tenni az aláhúzandó szavakat:

```
.
.
az ékezetes betűket közvetlenül
megérti. \underbar{nem kell hozzá preprocesszor\?}
\item{\ddag} Tartalmaz egy...
```

.
.
.

A felsorolás utolsó egységének „alpontjai" is vannak. Ez a következőképpen készül:

.
.
.

```
rendezett tárgymutatót készít.
\item{\ddag} Más fontos magyar szolgáltatásokat
is biztosít, mint például:
\item{--} az automatikus keltezés, fejezetcím
készítés és számozás négy címfokozatig, \item{--} a
magyar szokásoknak megfelelő \?idézőjelek" és
fejlécek készítése

és még sok egyéb hasznos dolog
```

Ha \itemitem parancsot adunk ki, akkor a TEX a meglévő felsorolási egységhez tartozó felsorolási egységeket szed ki. Ennél a „második" felsorolási szintnél a – jelet, a „fület" használtuk. Az üres sor a TEX számára mindig a bekezdés végét, új bekezdés nyitását jelenti. Az \itemitem és az \item parancsok hatásának végét jelenti az előbbi szövegrészben kihagyott sor. Az utána következő szavakat a TEX a „rendes" bekezdési mélységgel szedi, ami most nem felel meg nekünk. Azt szeretnénk, ha a tükör szélénél kezdődne az „és még sok egyéb hasznos dolog" szövegrész. Ezzel jeleznénk azt, hogy itt a felsorolás vége. Ezért a szövegrész elé egy \noindent parancsot beiktatunk. A szövegrész után is megismételjük ezt a parancsot, mert a következő bekezdést is a bal margónál szeretnénk kezdeni. Mivel fontos dologról van szó, a szöveget félkövér betűkkel kiemeljük, az ehhez szükséges fontot definiáljuk:

.
.
.

```
és még sok egyéb hasznos dolog.
```

```
\font\sanssbf=AMSSBX10
\noindent
A \maTeX\ programcsomag {\sanssbf megörzi a
rendszer \TeX\ kompatibilitását.} Igy például ...
.
.
```

A matematikai formulákat előállító parancsokat az 5. fejezetben részletesen tárgyaljuk, most nem térünk ki rá. A formulák után még egy bal margón kezdődő bekezdés van, és végül a forrásszöveg végére írjuk: \end ill. ehelyett azt is írhatjuk: \bye. A két parancs között az a különbség, hogy míg az \end parancs csak jelzi a TEX programnak a file végét, a \bye parancsnak ezenkívül más hatása is van: az utolsó oldalt kitölti üres sorokkal, az oldal formázását lezárja, és ezután jelzi a programnak a file végét. A két parancs ennek ellenére szinte mindig egyenértékű, biztonságosabb azonban — különösen bonyolultabb szövegek esetén — , ha a \bye parancsot használjuk:

```
.
.
.
és emellett a \maTeX\ programcsomag
teljes dokumentációja.
\bye
```

3.2. Az első fut(tat)ás

A forrásszöveg elkészült, tároljuk el egy file-ba amelynek a neve — mondjuk — BORITO.TEX. Itt fontos, hogy a kiterjesztés TEX legyen. Bár a TEX program más kiterjesztésű file-okat is fel tud dolgozni, mégis kevesebb hibalehetőséggel jár, ha igazodunk a konvencióhoz.

Tegyük fel, hogy a TEX rendszerünk már installálva van, és rendelkezésünkre áll az MPLAIN.FMT formátumfile. (A TEX ill. MATEX installálását a

függelékben ismertetjük.) Töltsük be a TEX programot a TEX.EXE file-ból; ez a program készíti el a forrásszövegünk (BORITO.TEX) alapján a szedett szöveget tartalmazó BORITO.DVI file-t. Mivel MATEX parancsokat, ékezetes betűket is használunk a forrásszövegben, meg kell adnunk a TEX-nek a definícióikat tartalmazó ill. beolvasó MPLAIN.FMT file-t is:

```
tex &mplain borito
```

A DOS számára mellékes, hogy kis- vagy nagybetűket használunk. Az & jellel tudatjuk a TEX-hel, hogy egy formátumfile neve következik, amelyere szükségünk van. A formátumfile kiterjesztését *nem szabad megadni* mert akkor a TEX nem találja meg. Ha több formátumfile-t is be akarunk tölteni, ezeket szóközökkel elválasztva egymás mellé írjuk:

```
tex &formategy &formatketto ....
```

A formátumfile-ok megadása után következik a forrás file. Ennél megadhatjuk a kiterjesztést is (BORITO.TEX), el is hagyhatjuk (BORITO), de a BORITO. rossz megoldás.

A program elindítása után a következő jelenik meg a képernyőn:

```
This is TeX, Version 2.0 (preloaded format=mplain 88.3.1) ...
**&mplain borito
--- MaTeX Ver. 0.6, (c)Magiszter 1988 ---
(borito.tex [1]
Output written on borito.dvi (1 page, 5100 bytes).
```

Az első sorban bejelentkezik a TEX, itt a bemutatottak mellett még egyéb üzenetek is szerepelhetnek pl. PCTeX Version ..., vagy MicroTeX .... Az utána következő zárójelben találjuk a betöltött formátumfile-ok nevét és azok létrehozásának dátumát. A sor végén a betöltéskori dátumot és időpontot láthatjuk. A második sorban levő '**' után a bemenő adatokat tartalmazó file-okat sorolja fel, esetünkben ez az mplain és a borito. A harmadik sorban jelentkezik be a MATEX makrócsomag. A munka elkezdésekor a TEX kiírja a „(borito.tex" üzenetet, és azután szögletes zárójelben az oldal számát, amellyel éppen foglalkozik; ha az oldalszám után bezárta a zárójelet, akkor azzal az oldallal már nem foglalkozik többet, az kész van. Ha az utolsó oldal is elkészült, kiírja a képernyőre az „Output written ..." üzenetet, amelyből megtudhatjuk hány oldalas lett a szöveg, ill. hogy

mekkora DVI file-t készített. A DVI file-al egyidőben a TEX egy LOG file-t (BORITO.LOG) is készít, ez a képernyőn megjelenő üzeneteket tartalmazza és a hibajavításnál komoly segítséget nyújt.

A következő lépésben az elkészült DVI file-t — esetünkben a előállított BORITO.DVI file-t — kinyomtatjuk papírra. A különböző TEX implementációknál más és más a nyomtatóprogramok neve, és használata. Itt a PCTEX dvieps mátrixnyomtató programját mutatjuk be, de hasonlóan működik a MicroTEX dvi-eps nyomtatóprogramja is, illetve ezen implementációk lézernyomtató programjai is.

Kezdő felhasználónak úgy a legkényelmesebb a programot használni, ha először csak a program nevét gépeli be:

```
>dvieps
```

Ekkor ugyanis a program egyenként rákérdez a szükséges adatokra:

- a nyomtatandó DVI file nevére,
- az output egységre,
- a felbontásra,
- arra, hogy hányadik oldallal kezdje a nyomtatást
- és hány oldalt nyomtasson.

A DVI file név megadásakor a kiterjesztés nélküli nevet adjuk meg. Az output egységnél általában elég, ha lenyomjuk a „return" (vagy „enter") billentyűt, ugyanis az alapértelmezés az, hogy a géphez kapcsolt printer nyomtatja ki a szöveget. Megadhatunk itt egy másik egységet, vagy egyszerüen egy file-nevet is. Az így létrejövő file azonban hatalmas méretű, a DVI file méretét több nagyságrenddel meghaladhatja! Ez a file ugyanis az egész szöveget, mint fekete—fehér pontokból álló grafikus információt tartalmazza. A BORITO egy oldalas szövegének esetében például a BORITO.TEX file 3496 byte-os, a BORITO.DVI file 5120 byte-os, az outputot tartalmazó file viszont 459777 byte nagyságú! A felbontás megadásánál a PCTEX esetében két, a MicroTEX-nél 5 lehetőség közül választhatunk. A kisebb felbontás, a 'draft' csúnyább, viszont gyorsabban kinyomtatja a program, mint a 'final'[2] -t. Érdemes javításokhoz, munkapéldányokhoz a 'draf' módot használni, mert ez a nyomtatót és a festékszalagot (festékkazettát) sokkal jobban kíméli. Ha

[2] 'draft' magyarul 'durva, nyers, csiszolatlant' jelent; a 'final' pedig 'végsőt'.

csak egy oldalt javítunk, nem kell az egész szöveget újra kinyomtatani; a nyomtatóprogram képes egyedi oldalakat is nyomtatni. Az alapértelmezés szerint (ha csak „return"-t nyomunk) a program az első oldaltól az utolsóig mindent kinyomtat.

```
DVIFNAME: borito
DVIEPS version 1.1     (c) Personal TeX, Inc. 1985

Printer file name [prn]:
Draft mode (1/120" resolution) or Final mode (1/240") [D or F]:
Starting page (default=*):
Maximum number of pages (default=1000000):
Options selected:
Starting page = *
Maximum number of pages = 1000000
magnification=1000;          0.00005067 pixels per DVI unit
' TeX output 1988.05.01:1311'
Font 57: amr10 --- loaded at size 1359216 DVI units
(this font is magnified 207%)
Font 56: hamr10 --- loaded at size 717619 DVI units
(this font is magnified 109%)
Font 55: hamr10 --- loaded at size 1132462 DVI units
(this font is magnified 173%)
.
.
.
Font 0: hamr10 --- loaded at size 655360 DVI units

beginning of page 1
[ magiszter Szmtstechnikai Szerkesztsg MA T E X . . . progra]
[mcsomag IBM PC/XT/AT kompatibilis . . .sMicroT E X valamint]
[ a Preview programokat teszi alkalmass . . . . . lehes]
[sen knnyen, gyorsan, szpen | a T E X . . . . . felhasz]
.
.
```

```
[onal Business Machines . . . . T E X azAMS (Amerikai Matem]
[atikai Trsulat) vdjegye; . . . . vllalatvdjegye; c MicroT ]
[E X az Addison{Wesley . . . . . c Preview aTextset, Inc. v]
```

A program kiolvassa és kiírja a képernyőre, hogy mekkora nagyítású, és mikor készült a DVI file, majd betölti a betűképeket tartalmazó PXL file-okat. Az oldalak nyomtatásakor a képernyőre is kiírja a szöveget. Mivel a képernyőn sem ékezetes betűket, sem különleges jeleket nem tud megjeleníteni ez kevéssé hasonlít a papíron levő szöveghez, pusztán arra jó, hogy követhessük, hol tart a kiíratás.

3.3. Az első pofonok

Valószerűtlen, hogy minden olyan simán, zökkenőmentesen menjen, mint ahogy az előző szakaszban leírtuk. A valóságban, a borító készítésekor sem volt így. A szerzők az első futtatáskor a következőket tapasztalták:

```
This is TeX, Version 2.0 (...) (no format preloaded)
**&mplain borito
                --- MaTeX Ver. 0.6, (c)Magiszter 1988 ---
(borito.tex
! Illegal unit of measure (pt inserted).
<to be read again>
                   m
<to be read again>

1.2 \vsize=210m

?
```

A TEX a hibát mindig a sor elején levő ! jellel jelzi. A hibaüzenet szerint a TEX által nem ismert mértékegységben, „m"-ben adtuk meg az oldal magasságát. Külön sorba kiírja az „elfogadhatatlan" karakter(eke)t, majd egy újabb sorban azt a szövegkörnyezetet, ahol a hibát észlelte:`\vsize=210m`.

Ennek a sornak az eleje a leginformatívabb: az 1.2 ugyanis azt jelenti hogy 2. sor[3]. A következő sorban álló kérdőjelre a program választ vár. Lehetőségeink:

- a h és a „return" billentyűt lenyomva „help"-et[4] kapunk:

```
? h
Dimensions can be in units of em, ex, in, pt, pc,
cm, mm, dd, cc, bp, or sp; but yours is a new one!
I'll assume that you meant to say pt, for printers' points.
To recover gracefully from this error, it's best to
delete the erroneous units; e.g., type '2' to delete
two letters. (See Chapter 27 of The TeXbook.)

?
```

A képernyőn megjelenő szövegből megtudhatjuk, hogy a TEX milyen mértékegységet fogad el, és azt hogy az általunk megadott m-et pt (pont)-tal helyettesítette. Ez most nekünk nem felel meg, hiszen a 210 pt csak kb. harmada a kívánt 210 mm távolságnak.

- A második lehetőség, hogy x-et és (returnt) nyomunk. Ekkor a TEX program futása befejeződik, lehetőséget adva nekünk arra, hogy az észlelt hibát szövegszerkesztővel kijavítsuk. A hibát kijavítva (a mm mértékegység hiányzó m betűjét pótolva) újra elindítjuk a TEX-het:

```
.
.
(borito.tex
! Undefined control sequence.
1.3 \nopagenumber

?
! Undefined control sequence.
1.50 seg!ts^^Bg^^Bvel \Tex
                          \ parancsokk^^F alak!totta ...
```

[3] sor angolul: 'line', ezért áll a sorszám előtt l-betű

[4] segítséget

```
? r
OK, entering \nonstopmode...
)
! Emergency stop.
<*> &mplain borito

No pages of output.
```

Mint láthatjuk, a TEX furcsa módon adja vissza az üzeneteiben a magyar ékezetes betűket. Ezt a kis kényelmetlenséget, amiről később még lesz szó, sajnos meg kell szoknunk.

- A harmadik lehetőség, hogy a kérdőjelre válaszul egy „return"-t nyomunk. Ekkor a TEX figyelmen kívül hagyja, hogy hibát talált, és a szöveg szedését folytatja. Ha a hiba nem katasztrofális a későbbi szövegrészekre — mint például esetünkben a `\nopagenumbers` parancs utolsó, lemaradt s betűje —, szerencsénk van. Érdemes ugyanis egy futtatásra minél több hibát felderíteni, mert az újabb futtatásokkor a TEX mindig előről kezdi a szöveg szedését. Ezt szolgálja a következő:
- negyedik lehetőség: `r` betűt és egy „return"-t nyomva a TEX az egész forrás file-t feldolgozza, egyetlen hibánál sem áll meg. Ha ezt választjuk, a program lefutása után a LOG file-ból kideríthetjük mi is történt.
- A szöveg első feldolgozásakor hasznos a 'scroll' lehetőség — ehhez `s` „return"-t kell nyomnunk —, mert ekkor a TEX nem szedi a szöveget csak szintaktikus ellenőrzést végez, igy sokkal gyorsabban feldolgozza a file-t.

A felsoroltakon kívül lehetőség van arra is, hogy futtatás közben beszúrjunk, vagy töröljünk karaktereket. Ez azonban sokszor több munkát okoz, mint amennyi előnnyel jár. A futtatáskor elvégzett javítások ugyanis nem kerülnek bele a forrásszövegbe, ezért ezeket kétszer kell elvégeznünk; és mindig emlékeznünk kell arra, hol, mit, mire javítottunk.

4. fejezet

Hogyan tördel a TEX?

4.1. Hogyan adjuk meg az oldal formáját?

Amikor létre akarunk hozni egy dokumentumot a TEX segítségével, nem elég azt tudni, hogyan vigyük be a szöveget, hogy azt a TEX egy adott formában hozza létre. Be kell tudnunk állítani az oldalnak a szövegtől független paramétereit is, mint pl. az oldal méreteit, margók nagyságát, élőfej vagy lábsor tartalmát, stb. Az itt szereplő parancsok egy jelentős részét a korábbi fejezetek már vázlatosan érintették, itt részletes használatukat írjuk le.

4.1.1. Az oldal paraméterei

Az oldal szélességét a `\hsize`, magasságát pedig a `\vsize` paraméter beállításával kapjuk meg. Ez a könyv például a `\hsize=127mm` és `\vsize=175mm` beállításokkal készült, míg az alapértelmezés szerint ezek `\hsize=6.5in` és `\vsize=8.9in`. Mint a hasonló esetekben mindig, itt is használhatjuk a `cm` (centiméter), `mm` (milliméter), `in` (inch), `pc` (pica), vagy `pt` (pont) mértékegységek bármelyikét. Az oldal szélességén itt a normál sorokká tördelt szöveg — tehát nem a beljebb kezdett vagy valamely egyéb módon megváltoztatott sorok — szélességét értjük, míg az oldal magassága az élőfej és lábsor[1] nélküli, de a lábjegyzeteket tartalmazó oldalrész magasságát jelenti.

[1] A lábsor másik lehetséges elnevezése lehetne az élőfej mintájára az élőláb. Ezen a fogalmon a könyvben azt a sort értjük, amely az oldalszámot tartalmazza, amennyiben

Ezeket a változókat lehetőleg csak egyszer, a megírandó dokumentum legelején állítsuk be, és később ne változtassuk. Ha valakinek szüksége lenne arra, hogy a lap szélességét vagy magasságát a dokumentumon belül megváltoztassa, először próbálja meg ezt a rendelkezésre álló (főleg ebben a fejezetben leírt) egyéb utasításokkal elérni. Ha ezek nem elegendők a kitűzött cél eléréséhez és feltétlenül meg kell változtatni a `\hsize` és `\vsize` paramétereket, akkor vegyük figyelembe, hogy a TEX — miként azt később részletesebben is leírjuk — oldalanként hozza létre az outputot. Egy adott oldal létrehozásakor — ami mindig akkor történik, amikor az oldal megtöltéséhez elegendő anyagot olvasott be — mindig az éppen érvényben levő paramétereket veszi figyelembe.

A nyomtatott szövegnek az aktuális lapon való elhelyezkedését a bal és a felső margó beállításával szabályozhatjuk. Ez a gyakorlatban egy kicsit nehézkesebb, mint az oldal méreteinek a beállítása, mivel az output helyzetét a nyomtató beállítása is befolyásolhatja. Tanácsos tehát először egy–két oldalt kinyomtatni, megnézni a nyomtatott rész elhelyezkedését az oldalakon és utána megváltoztatni az alapértelmezéseket. Elméletileg a bal margó éppen egy inch-csel nagyobb, mint a `\hoffset` paraméter értéke, ami az alapértelmezés szerint `0pt` (azaz nulla pont). Hasonlóan, a felső margó egy inch-csel nagyobb, mint az alapértelmezés szerint `0pt`-ra beállított `\voffset` paraméter. Ezek az értékek azonban hangsúlyozottan elméletiek, mindenkinek magának kell kitapasztalnia, hogy a saját számítógépén és nyomtatóján mik a valóságos értékek. Ha például egy adott konfigurációban az alapértelmezés szerinti beállításoknál (tehát, ha nem adjuk meg a `\hoffset` és `\voffset` változók értékeit) a felső margó 3 cm, de mi 4 cm-re akarjuk beállítani, akkor (az alapértelmezés szerinti `\voffset=0pt` helyett) a `\voffset=1cm` beállítást kell alkalmaznunk. Hasonlóan, ha a bal margó 2,8 cm, de mi csak 2,3 cm-es bal margót szeretnénk, akkor a `\hoffset=-.5cm` utasítást adjuk meg az inputban. A `\hoffset` és `\voffset` paramétereket már sokkal valószínűbb, hogy meg kell változtatni többször is egy adott szövegen belül — gondoljunk csak arra, hogy a könyvek esetében általában különbözik a páros és a páratlan oldalakon alkalmazott margó. Ezekre is ugyanaz a szabály

az az oldal aljára kerül. A magyar könyvekben a lábsorban nagyon ritkán szerepel az oldalszámtól eltérő szövegrészlet, de mivel más országok tipográfiai szokásai a lábsor adta lehetőséget is felhasználják információ közlésére, a programot felkészítették ennek kezelésére is.

érvényes, mint a többi, az oldal külalakját szabályozó paraméterre, azaz az adott oldalra az a beállítás lesz hatással, ami az oldal létrehozásakor, tehát az oldal alján van érvényben.

Többek között a fenti típusú paraméterek pontos beállításában lehet segítségünkre az `\advance` parancs. Ez arra szolgál, hogy akkor is be tudjunk állítani a kívánt értékre egy paramétert, ha annak nem ismerjük az alapértelmezés szerinti beállítását. Például az `\advance\vsize by 2cm` parancs hatására bármilyen magas is volt eddig egy oldal, mostantól két centiméterrel magasabb lesz. A bal margó fél centiméterrel való csökkentését pedig, amire az előző példában volt szükségünk, az `\advance\hoffset by -.5cm` paranccsal érhetjük el.

4.1.2. Oldalszámozás, élőfej és lábsor

Mint már említettük, az élőfej és a lábsor az oldal szövegétől független része, tehát külön kell őket beállítani. Ez természetesen vonatkozik az oldalszámra is, mivel azt általában ezen két extra sor valamelyike tartalmazza. Az alapértelmezés szerint az élőfej üres és a lábsor csak az oldalszámot tartalmazza középre zárva. A `\nopagenumbers` parancs hatására ez is megszűnik, nem lesz oldalszámozás és mind az élőfej, mind a lábsor üres lesz az oldalakon. Ha valamely okból nem az első oldallal kívánjuk elkezdeni a dokumentumot (például egy könyvet írunk fejezetekre tagolva, és a harmadik fejezet az 52. oldalon kezdődik), vagy valahol a dokumentumon belül meg szeretnénk változtatni a TEX által automatikusan adott oldalszámozást, ezt a `\pageno` paraméter beállításával tehetjük meg. Például a dokumentum legelején alkalmazott `\pageno=52` utasítás hatására az első oldal száma 52 lesz. A `\pageno` paraméter esetén is, miként mindig ebben a fejezetben, az oldal számát az annak a létrehozásakor érvényes `\pageno` érték határozza meg. A TEX természetesen minden oldal létrehozása után emeli a `\pageno` értékét eggyel, és így automatikusan követi az oldalszámozást.

Még egy fontos szolgáltatást nyújt az oldalszámozással kapcsolatban a TEX. Előfordulhat, hogy valaki, például egy könyv írása esetén, az első néhány oldalt római számokkal óhajtja megszámozni a i, ii, iii, iv stb. mintájára, majd a könyv fő részeitől már az arab számozást szeretné használni, egytől kezdődően. Ha a `\pageno` változónak az x negatív értéket adjuk,

akkor az azt követő első oldal száma $|x|$ lesz római számmal megadva. Ha például a könyv számozását a 3. oldalon akarjuk elkezdeni római számokkal (tehát a iii oldalszámmal), akkor a dokumentum elején a \pageno=-3 parancsot adjuk ki, majd azon az oldalon, ahonnan el akarjuk kezdeni az arab számozást (ha azt az oldalt 1 számúnak szeretnénk), kiadjuk a \pageno=1 parancsot. Az éppen aktuális oldalszámot az érvényben lévő formában tárolja a \folio változó. Ha tehát a \pageno értéke 34, akkor a \folio (ami már nem egy szám, hanem egy karaktersorozat a TEX szempontjából) tartalma 34, míg ha a \pageno értéke -4, akkor a \folio tartalma „iv".

Az \advance parancsot a \pageno és hasonló típusú (tehát számértékű) változók megváltoztatására is felhasználhatjuk. Az

```
\advance\pageno by 3
```

utasítás például hárommal megnöveli az oldalszámot. Erre akkor lehet szükség, ha valamiért néhány üres oldalt akarunk kihagyni a dokumentumban. Ezt ugyan, ahogy később majd látjuk, üres oldalak kinyomtatásával is megtehetjük, de ennek természetesen nem sok értelme van, hasznosabb, ha egyszerűen az oldalszámot növeljük a megfelelő mértékben. Természetesen ezen parancs használatakor figyelnünk kell, hiszen ha az oldalakat éppen római számokkal számozzuk, akkor a nagyobb római számok eléréséhez a \pageno értékét éppenhogy csökkentenünk kell. Például ha azt szeretnénk, hogy a iii oldal után (amikor a \pageno értéke -3) a vii oldal következzen, azaz a \pageno a következő -4 érték helyett a -7 értéket vegye fel, akkor az \advance\pageno by -3 parancsot kell kiadnunk. Ezt a nehézséget oldja meg nekünk az \advancepageno parancs, amely mindig a kívánt irányban változtatja meg a \pageno értékét. Így, ha pl. az 5. oldalon járunk, az \advancepageno hatására a 6. oldalra kerülünk, a iii oldalt viszont a iv oldal követi. Természetesen a TEX automatikája is ezt az utasítást használja, hogy — mint azt már korábban említettük — egy–egy oldal létrehozása után az oldalszámot megnövelje eggyel.

Az élőfej és a lábsor természetesen nemcsak az oldalszámot, hanem egyéb fontos információt is tartalmazhat. A \headline változó tartalmazza az élőfej tartalmát, a \footline pedig a lábsorét. Az alapértelmezés szerinti beállítás tehát a következő:

```
\headline={\hfil}
\footline={\hss\tenrm\folio\hss}
```

Az élőfej kitöltése itt azt jelenti, hogy nincsen látható tartalma, azaz üres. A lábsor viszont egy kicsit több magyarázatot kíván. A `\folio` paraméter, mint már tudjuk, az oldalszámot tartalmazza az éppen aktuális formában. A `\tenrm` utasítás hatására ez a tízpontos antikva betűtípusból kerül szedésre, mint azt várjuk is[2]. Ha ez az utasítás nem szerepelne itt, akkor az oldalszám mindig abból a betűtípusból lenne kiszedve, amely az oldal tördelésekor éppen érvényben van, azaz jó eséllyel oldalról oldalra változna. A `\hss` utasítás pedig egy végtelenül nyújtható és végtelenül összenyomható (a negatív irányba is) szóközt jelöl. Ennek a `\folio` két oldalán való alkalmazása tehát azt eredményezi, hogy az oldalszám alul középre lesz zárva. Hatása annyival több a szintén végtelenül nyújtható `\hfil` hatásánál, hogy ha itt a `\folio` helyén egy olyan szövegrészlet állna, amely kiszedve szélesebb lenne az oldalszélességnél, akkor azt a TEX a két `\hss` egyenlő mértékű negatív összenyomásával továbbra is az oldal közepére szedi oly módon, hogy a lábsor mindkét irányban ugyanolyan mértékben nyúlik túl a margókon. A `\hfil` alkalmazása esetén a TEX ezt nem tudná megtenni, és hibaüzenetet adna.

A `\headline` és `\footline` paraméterek megismerése után nem meglepő, hogy a `\nopagenumbers` utasítás nem más, mint a `\footline={\hfil}` rövidítése. Az élőfej és lábsor tartalmát nemcsak előre lehet beállítani, mi is beállíthatjuk magunknak bármikor az ismertetett paraméterek megváltoztatásával. Például a

```
\headline={\tenrm\folio\hfil}
```

hatására az oldalszám az élőfejben (is) megjelenik a bal oldalra zárva, vagy a

```
\headline={\hrulefill}
```

hatására az élőfej egyszerűen egy vízszintes vonal lesz az oldal teljes szélességében. Mint már említettük, a TEX az oldalak méretének a beállításakor az élőfejet és a lábsort nem veszi figyelembe, így ha élőfejet is használunk, és még e fölött további egy inch margót is szeretnénk, akkor (feltéve, hogy az alapértelmezés szerinti `\voffset=0pt` beállításra valóban egy inch-nyi

[2] Mivel itt csak egy számot szed ki a program, mindegy, hogy a `\tizant` magyar vagy `\tenrm` angol antikva betűtípust használjuk.

felső margó adódik) a `\voffset=2\baselineskip` parancsot kell kiadnunk, aminek hatására a felső margó az élőfej által elfoglalt sor és az oldalon lévő szöveg között kihagyott egy üres sor magasságának összegével lejjeb kerül.

Az egyes dokumentumok létrehozásakor az oldalak jellemzői attól is függhetnek, hogy a dokumentumnak éppen milyen típusú oldalán járunk éppen. Elképzelhető például, hogy egy cikk esetén az első oldalra nem szeretnénk oldalszámot tenni, vagy hogy csak a második oldaltól szeretnénk az élőfej tartalmát valamivel feltölteni. Még gyakoribb példa — ahogy azt már említettük — a könyvek jobb és bal oldalai, amelyeket különböző alakúakra szoktak szedni. Általában eltérő a lapszámozás ezeken az oldalakon,[3] vagy ami még általánosabb, a teljes élőfej és lábsor különböző módon kerül kiszedésre az oldalszám paritásától függően. A lapszámozás a magyar irodalmi művekben általában a jobboldali oldalakon a lábsor jobb szélén, míg a baloldali oldalakon a lábsor bal szélén van, az élőfej pedig mindkét esetben üres. A technikai jellegű művekben inkább a lábsor marad üresen, az élőfejek pedig az oldalszámon kívül valamely egyéb információt is tartalmaznak. Általában a jobboldali oldalak élőfeje a jobb oldalon tartalmazza az oldalszámot, középreÓ zárva pedig az éppen futó fejezet címét. A baloldali oldala élőfeje balra zárva tartalmazza az oldalszámot és úgyszintén középre zárva a fejezeten belüli szakasz címét. A következő fejezetben, ahol megtanuljuk hogyan definiáljunk saját utasításokat, példákat találunk arra, miként lehet a különböző szokásoknak megfelelő élőfejet és lábsort létrehozni.

4.2. Hogyan tördeli a TEX a sorokat?

Első pillanatban azt gondolhatnánk, a felhasználónak nincsen szüksége a címben feltett kérdés válaszára, hiszen a TEX azért egy szövegszedő rendszer, hogy önállóan meg tudja oldani a tördelést. Ez természetesen részben igaz is. Mindössze azt nem szabad elfelejtenünk, hogy a TEX is „csak" egy számítógépes program, ami teljes pontossággal végrehajtja azt, amit előírtak neki, de semmi többet. Többek között nem képes gondolatolvasásra, így nem

[3] A magyar könyvekben a baloldali oldalak páros számúak, a jobboldaliak pedig páratlan számúak.

fog csak azért új sort kezdeni, mert mi azt úgy szeretnénk. Így tehát először is azért van szükség a címben megadott algoritmus megértésére, hogy azt szabályozni tudjuk. Továbbá a TEX is egy „véges" program, lehetetlenség megtanítani mindenre, így néha előfordulhat hogy egy-egy sort nem a legalkalmasabb helyen próbál megtörni. Például nem túl szerencsés dolog, ha a 6 órakor szövegrészletet úgy próbálja elválasztani, hogy a 6 marad egy sor végén és az órakor a következő sor elejére kerül (ilyen esetekben nem mindig lehet megmondani, hogy mely kérdés tartozik a „gondolatolvasás" fogalmába és melyik az, amelynek eldöntésére esetleg valamely módon meg lehetne tanítani a TEX-et). Meg kell értenünk tehát a sorelválasztó algoritmust azért is, hogy adott esetben kijavíthassuk a TEX által elkövetett hibákat.

4.2.1. Az algoritmus

A válasz az első megközelítésben nagyon egyszerű. A program sorra veszi be a szavakat, egyiket a másik után, elhelyezi őket az adott sorban mígcsak a sor végére nem ér, akkor pedig új sort kezd[4]. Csakhogy mivel itt könyv minőségű nyomtatvány készül és így a sorok jobbra is ki vannak zárva, a valóságban ez soha nem ilyen egyszerű — a sor végére szinte soha nem egy szó végén érünk. Tehát az utolsó beolvasott szó már olyan, ami nélkül a sor túl rövid lenne, vele együtt viszont a sor már túl hosszú. Ilyenkor van szerepe a TEX sorelválasztó algoritmusának, amely több lehetséges megoldás közül igyekszik kiválasztani a legjobbat, a legszebbet. Egyik ilyen lehetőség a sorok széthúzása vagy összenyomása (attól függően, hogy a kritikus szót a TEX már átviszi-e a következő sorba, vagy még az előzőben hagyja) a szóközök megnövelésével vagy összenyomásával. A legtöbb szövegszerkesztő rendszer csak ezt a megoldást ismeri (ha egyáltalán kizárja a sorokat). A szövegszedő rendszerek számára — éppen mivel nagyon szép, könyv minőségű nyomtatványt kívánnak létrehozni — ez a megoldás csak az esetek egy kis részében megfelelő, amikor a széthúzás vagy összenyomás nem túl nagy mértékű és így nem rontja a szövegoldal esztétikai megjelenését. Természetesen a TEX

[4] Noha ennél a nagyon egyszerű leírásnál bonyolultabbat adunk meg, az is csak egy egyszerűsítése lesz a TEX igen bonyolult algoritmusának. Arra azonban elegendő, hogy az alapvető működését megértsük és befolyásolni tudjuk.

nem szóközöknek a szavak közé illesztésével zárja ki a sorokat. A legkisebb mérték, amit felhasznál, az ún. „scaled point" a pontnak (ami már maga is jóval kisebb, mint egy milliméter) az 1/65536-od része; a sorkiegyenlítés ilyen „scaled point"-onként történik.

Ha a sor széthúzásával vagy összenyomásával nem kapnánk elég szép összképet, akkor a program a sor végére eső szó elválasztásával próbálkozik, egy ötletes szóelválasztó algoritmus segítségével. A MAGYAR TEX magyar változat megtanulta a magyar elválasztási szabályokat és a kivételek nagy részét is. Így tehát nagy valószínűséggel a TEX talál olyan pontokat a sorvégi szavakban, ahol azokat el tudja választani[5]. Anélkül, hogy statisztikai vizsgálódásokba bonyolódnánk, kijelenthetjük, hogy a magyar változat megtalálja a lehetséges magyar elválasztási pontok legalább 95%-át és csak egészen kivételes esetekben talál rossz elválasztási helyeket. Másképpen fogalmazva, a MAGYAR TEX ismeri a magyar elválasztás alapvető szabályait és a leggyakoribb kivételeket (összetett szavakat, igekötős igéket). A TEX-nek magának is vannak — az esztétikai követelményekből fakadó, és az adott nyelvtől független — szabályai a lehetséges elválasztási helyekre. Így például nem hagy egy sor végén egy egyetlen betűből álló szótagot és nem visz át új sorba egy vagy két betűből állót. Ezek a szabályok azonban főleg csak a rövid szavakat érintik, amelyek esetében viszont nagy valószínűséggel működik az előző bekezdésben tárgyalt sorszéthúzás vagy összenyomás. Egyébként a tipográfiai szabályok sem engedik egyetlen betűből álló szótag leválasztását a szóról.

Tegyük most fel, hogy a TEX talál egy vagy több jó elválasztási pontot. Ekkor egy igen bonyolult számítás és előre beállított „büntetőpontok" alkalmazásával kiszámítja, hogy a szóba jövő lehetséges megoldások közül (a sor széthúzása vagy összenyomása, illetve a különböző lehetséges pontoknál történő szóelválasztások) melyik az, amelyik az esztétikai igényeknek legjobban megfelel[6], és ha az így kapott sor még elég szép — azaz egy bizonyos

---

[5] Természetesen, ha valaki a TEX angol változatát használja, akkor a program az angol elválasztás szabályai szerint fogja a szavakat elválasztani — noha az angol elválasztási szabályok bonyolultabb volta miatt a magyar elválasztásnál kicsit nagyobb hibaszázalékkal.

[6] Tapasztalt TEX felhasználók részletesebb könyvekben megtalálhatják az ilyen esetekben alkalmazott számítás és paraméterek pontosabb leírását, és így a saját igényeik szerint változtathatják az oldalak külalakját. Például aki inkább szeretné, hogy a sorok jobban össze legyenek nyomva vagy szét legyenek húzva, mint hogy sok elválasztás

előre adott tűréshatáron belül marad, — akkor ezt alkalmazva töri meg a sort.

A dolog neheze akkor következik, ha a TEX nem talál a magas esztétikai igényeinek megfelelő sorelválasztási pontot. Ekkor nem próbálkozik tovább (nem is tud), hanem a sort az általa rossznak ítélt módozatok egyikét használva helyezi az outputba, és figyelmeztető üzenetet ad. Ez az eset két különböző módon valósulhat meg. Vagy túl rövid a sor, így a sorkizárás miatt túlságosan szét kell azt húzni . Ilyenkor az alábbi figyelmezető üzenethez hasonló jelenik meg a képernyőn:

```
Underfull \hbox (badness 1546) in paragraph at lines 85--93
[]\tenrm A do-log ak-kor lesz ne-he-zebb, ami-kor az adott[7]
```

A 8. fejezetben pontosan leírjuk hogyan kell a hibákat azonosítani és kijavítani, így itt most csak a soremelésre figyelünk. Az üzenetből kiolvashatjuk, hogy a rosszul elválasztott sor a

```
A dolog akkor lesz nehezebb, amikor az adott
```

szövegrészletet tartalmazza és az inputban a 85–93. sorok által meghatározott bekezdésben van valahol (nem feltétlenül egy sor elejétől a végéig terjedő részen). Feltehetően azért lett túlságosan széthúzva, mert egy viszonylag hosszú szó következik utána, amelynek az elején valamely okból a TEX nem talált szóelválasztási pontot. A TEX által kiírt figyelmeztetésből azt is kiolvashatjuk, hogy mely helyeken választaná el a sorban szereplő szavakat, ha erre valamiért sor kerülne. A szövegben kikeresve a megfelelő szót eldönthetjük, hogy egyetértünk-e a TEX-hel és számunkra is túlságosan nagy mértékben vannak széthúzva a szóközök vagy pedig a sor kielégíti az igényeinket. A második esetben semmi teendőnk nincsen, míg az elsőben ki kell valahogy javítanunk a hibát, amelynek módozataira később térünk ki.

A másik lehetséges eset és figyelmeztető üzenet akkor következik be, ha a hibás sor túl hosszú lesz, belenyúlik az oldal jobb margójába. Ebben az esetben a következő típusú üzenetet kapjuk:

történjék, az megnövelheti az elválasztásokért kapott büntetőpontokat és csökkentheti a széthúzásért adottakat.

[7] Ne lepődjünk meg, ha néha a kiírt sor nem ilyen könnyedén olvasható. A TEX csak az angol ábécé betűit képes az eredeti alakjukban a képernyőre kiírni, minden más jel vagy csak nagyon furcsán jelenik meg, vagy egyáltalán meg sem jelenik.

```
Overfull \hbox (8.934pt too wide) in paragraph at lines 85--93
[]\tenrm A do-log ak-kor lesz ne-he-zebb, ami-kor az adott
```

Ez azt jelenti számunkra, hogy a benne megjelölt sor az itt adott formában került az outputba és 8,934 ponttal szélesebb az oldal adott szélességénél. Természetesen ebből az üzenetből is kiolvashatjuk, hogy az inputban a 85. és a 93. sorok között helyezkedik el a problémás szövegrészlet. A hiba forrása itt még közvetlenebbül kiolvasható. A sor nagyon rövid lenne az adott szó nélkül, vele viszont már túl hosszú, elválasztani pedig nem tudja a program. Itt az okozza a problémát, hogy habár a TEX megtalálja a lehetséges a-dott szótagolást, egybetűs szótagokat nem választ le a szóról. Hogy meg tudjuk becsülni, mennyivel is hosszabb a sor a kelleténél, 1 pontot durván 0,35 milliméternek számíthatunk. A példában szereplő sor tehát durván 9 ponttal, azaz 2,15 milliméterrel hosszabb a kelleténél. Ez természetesen tényleg csúnya sort eredményez, tennünk kell valamit. Előfordulhat azonban, hogy a program mondjuk csak 0.285pt-tal, azaz körülbelül egy tized milliméterrel hosszabb sort jelez, ami szabad szemmel egyáltalán nem észlelhető és így nem is kellene kijavítani. Figyelmen kívül is hagyhatnánk így a figyelmeztető üzenetet, azonban a program, az implementálástól függően, egy fekete téglalapot tesz az ilyen figyelmeztető üzenetben szereplő sorok végére, hogy a javításkor könnyebben észrevehető legyen. Ha ez a téglalap megjelenik a sor végén, akkor a figyelmeztető üzenet így módosul:

```
Overfull \hbox (8.934pt too wide) in paragraph at lines 85--93
[]\tenrm A do-log ak-kor lesz ne-he-zebb, ami-kor az adott|
```

A különbség a második sor végén megjelenő, a hibás output sort jelképező sor végén megjelenő függőleges vonásban van, ami a téglalapot jelzi. A fenti jelenség elkerülésére két dolgot tehetünk. A TEX is fel van készítve arra, hogy ne vegyen túlságosan szigorúan minden hosszú sort. Nevetséges lenne, ha egy sort azért tekintenénk nem elfogadhatónak, mert 0.002pt-tal hosszabb a kelleténél. A programba tehát be van építve egy \hfuzz paraméter, amely megmondja, hogy mi az a mérték, amíg egy túl hosszú sort nem kell hibásnak tekinteni, és így nem kell figyelmeztető üzenetet sem kiadni rá. Az alapértelmezés szerint ez az érték 0.1pt, de ha például a \hfuzz=1pt parancsot kiadjuk valahol az input elején, akkor a TEX a jobb margón egy pontnál nem többel túllogó sorokat nem tekinti hibásnak és nem látja el a

fekete téglalappal a végén[8]. Ily módon elérhetjük, hogy a program csak azokat a sorokat jelölje meg az outputban, amelyeket a véleményünk szerint is ki kell javítani. Ha viszont valamely okból az összes hosszú sort jelző téglalapot el szeretnénk tüntetni, akkor az ezek szélességét kijelölő `\overfullrule` paramétert kell beállítanunk nullára az `\overfullrule=0pt` paranccsal. A paraméter alapértelmezés szerinti beállítása egyébként `5pt`.

Itt most be kell vallanunk, hogy az algoritmusról idáig elmondottakban van egy kis hazugság. Ha kicsit elnézőbbek akarunk lenni önmagunkhoz, akkor azt is mondhatjuk, túlegyszerűsitettük a dolgot. Most azonban egy kicsit pontosabban mondjuk el. Az igazság az, hogy a program nem akkor dönt a soremelésről, amikor már mindenképpen elegendő anyagot olvasott be ahhoz, hogy kitöltse a sort, hanem csak sokkal később, a bekezdés végén. Ennek az az oka, hogy az egymás utáni soremelések igen szoros kölcsönhatásban vannak egymással, előfordulhat, hogy egy soremelést kicsit megváltoztatva az ugyan „csúnyább" lesz, de csak így tudunk a következő sorra egyáltalán elfogadható emelési pontot találni. Így tehát az algoritmus valójában a következők szerint működik: először a TEX beolvassa a teljes bekezdést, majd elkezdi a soremelési pontokat keresni. Ezeket azonban mindig együtt keresi és együtt értékeli ki. Ha ilyen módon megtalálja a soremelési pontok „legjobb" halmazát, azaz azt a rendszert, amelynek együttes alkalmazása a legkevésbé rontja a bekezdés külalakját, és amely természetesen még az esztétikai „tűréshatárai" közé esik, azt fogja alkalmazni. Ha ilyet nem talál, akkor részben feladja a próbálkozást és az előbbiekben leírt típusú „rossz" sorokat hoz létre. Elismerjük, ez így egy kicsit bonyolult. Szerencsére azonban nem kell teljesen megértenünk, csak azt jegyezzük meg jól — amire később szükségünk lesz — hogy a soremelési pontok kiválasztása bekezdéseként egyszerre történik, mégpedig a bekezdések legvégén.

4.2.2. Hogyan befolyásolhatjuk az elválasztást?

Mi tehát a teendő akkor, ha a TEX egy olyan sort talált, amely szerinte már nem nyomtatható ki a helytelen soremelés miatt és mi is egyetértünk ezzel a

[8] A tapasztalatok azt mutatják, hogy a `\hfuzz` értékét körülbelül egy-két pontra érdemes beállítani.

döntéssel. Az első és legfontosabb teendő annak megvizsgálása, hogy nincs-e egy olyan elválasztási pont, amely alkalmazásával jobban meg lehetne oldani a soremelést, azaz amely a TEX által megtalált két lehetséges pont közé esik és amelyet valamely okból a program nem vett észre. Általános szabályt nem tudunk megadni ezekre az esetekre, csak a talán legtipikusabbat említjük: a program esetleg nem talál meg egy lehetséges szóelválasztási pontot. A MATEX programot használva ez egészen ritka esetektől eltekintve akkor fordulhat elő, ha a szó a magyar nyelvben ritka összetett szó, ha a szóban kötőjel szerepel, vagy pedig ha kettőzött két betűvel jelölt mássalhangzót kellene elválasztania a programnak, mint amilyen az asszony szóban az ssz. Az első esetben egyébként egészen ritkán az is megeshet, hogy a MATEX olyan elválasztási pontot talál, ahol a szót a szabályok szerint nem lehet elválasztani. Sajnos ezt csak az output olvasásakor lehet észrevenni és kijavítani, de ez olyan ritka eset, hogy nem okozhat komolyabb problémát. Mindenesetre a most következő módszer mindkét hiba kijavítására alkalmas.

Ha tehát egy olyan szó kerül az inputba, amelynek a helyes elválasztásával a MATEX nincs tisztában, de amelynek az elválasztására a helyes soremelési pont megtalálása miatt szükség van, akkor meg kell mondanunk a programnak, hogyan válassza el ezt a szót vagy ezeket a szavakat. Ezt elsősorban úgy tehetjük meg, hogy a szóban (szavakban) kijelöljük a TEX számára az elválasztási helyeket a \- parancsok segítségével. Például, ha a MATEX a cigányélet szót ci-gá-nyé-let-ként próbálja elválasztani, akkor ha a szót ci\-gány\-é\-let-ként adjuk meg az inputban, a helyes elválasztási pontokat fogja alkalmazni. Ha egy szóban megadunk ilyen parancsot, akkor hatására a program nem vizsgálja, hogy a saját algoritmusa szerint találna-e más elválasztási helyeket. A \- parancs értelemszerűen csak arra a szóra érvényes, amelyikre alkalmaztuk. Ha tehát a cigányélet szó többször előfordul a szövegben, sőt többször kerül olyan pozícióba, hogy a TEX hibásan akarja elválasztani, akkor minden esetben külön-külön meg kell adni a helyes elválasztását. Ez a parancs természetesen nemcsak a hibás elválasztási pontok kijavítására szolgál, hanem a TEX által valamiért nem alkalmazott elválasztási pontokat is ki lehet vele jelölni. Ha például az előző szakaszban tárgyalt hibás sorban az adott szót hajlandók vagyunk a-dott-ként is elválasztani, akkor az inputban a\-dott-nak kell megadnunk.

Előfordulhat, főleg speciális szakszövegek TEX-hel való szedésekor, hogy sűrűn előforduló (szak-) szavakat a programnak többször el kellene válasz-

tani, de a szó azon kevés kivételek közé esik, ahol a program rossz elválasztási pontokat talál. Ilyenkor nem kell minden esetben külön megadni a helyes elválasztást, hanem alkalmazhatjuk a `\hyphenation` parancsot, amelynek a változójaként kell felsorolnunk a kivételes módon elválasztandó szavakat — kijelölve a helyes elválasztási pontokat — egy-egy szóközzel elválasztva. Ha például kiadjuk az input elején a `\hyphenation{ci-gány-é-let a-dott}`, akkor az így megadott szavakat az adott input file-ban mindenhol csak a helyes módon fogja elválasztani a TEX. Vigyáznunk kell azonban arra, hogy ez csak a teljesen azonos formában megjelenő szavakra vonatkozik. Így a `cigányéletet` vagy `adottnak` szavakat már nem lesz képes a program helyesen elválasztani, csak akkor, ha ezek is szerepelnek ilyen formában a `\hyphenation` parancs változójában.

A fenti módszer sem oldja meg azonban az `asszony` típusú szavak helyes elválasztását teljesen. Ha biztosak vagyunk benne, hogy a TEX el fogja választani a kérdéses helyeken a szavakat amennyiben megadjuk neki a helyes elválasztási módot, akkor a problémás helyeken az `asszony` helyett megadhatjuk az `asz\-szony` alakot is. De ha ekkor mégsem választaná el a program a szót, akkor az `aszszony`-ként jelenik meg az outputban. Ugyanezen okból nem alkalmazhatjuk a `\hyphenation{asz-szony}` megoldást sem, mivel ennek hatására a program csak az `aszszony` alakú szavakat keresné és választaná el a kívánt formában. A TEX program ezen nehéz feladat megoldására is szolgál egy paranccsal. Ha egy szóban a `\discretionary` utasítás szerepel, akkor annak helyén a program elválaszthatja a szót. A parancs három egymás utáni változójában megadott szövegrészleteket a következő módon kezeli. Ha a soremelési algoritmus úgy dönt, hogy az adott helyen elválasztja a szót, akkor a parancs első változójában szereplő szövegrészlet marad a régi sorban és a második változóban szereplő részlet az új sorba kerül. Amennyiben úgy adódna, hogy a szót nem kell elválasztani, vagy nem a `\discretionary` parancs által kijelölt helyen kell elválasztani, akkor a parancs helyére az outputban a harmadik változó kerül. Talán jobban megérthető az utasítás, ha egy példával szemléltetjük. Az

```
a\discretionary{sz-}{sz}{ssz}ony
```

példában a szó elválasztása esetén a régi sorban marad az a betű és az első változóban adott `sz-` szövegrészlet, ami összesen tehát `asz-`, míg a szóból

az új sorba kerül a második változóban adott sz és a maradék ony. Így, ha el kell választani a szót, akkor az az asz-szony módon történik. Ezzel szemben, ha nem kell elválasztani, akkor az a és az ony közé a harmadik változó kerül, asszony-t eredményezve.

Ezek tehát azok a lehetőségek, amelyeket megpróbálhatunk alkalmazni, amennyiben a TEX valamely okból nem tudna egy soremelést helyesen megoldani. Sajnos azonban az is előfordulhat, hogy a program minden lehetőséget megtalált a soremelésre, mégsem tudja azt megfelelően végrehajtani, mivel nincs olyan pont, amely az összes igényt kielégítené. Ekkor már csak az marad, hogy vagy átírjuk a szóban forgó szövegrészletet, vagy pedig — ha ezt nem akarjuk megtenni — arra kényszerítjük a programot, hogy egy általunk még elfogadhatónak talált helyen törje meg a sort. Mivel túl hosszú sorokat semmiképp sem engedhetünk meg, így az utóbbi megoldás gyakorlatilag azt jelenti, hogy a `Overfull \hbox` figyelmeztető üzenetek esetén a TEX által kijelölt soremelési pontnál eggyel korábban törjük meg a sort. Ezek a megoldások minden esetben `Underfull \hbox` figyelmeztetéseket és túlságosan széthúzott sorokat eredményeznek, de erre a kompromisszumra ilyenkor szükség van.

Ha jobban szeretjük a nem kizárt sorokat, mint az esetlegesen túl nagy szóközöket, a TEX erre is lehetőséget ad. A `\raggedright` utasítás hatására a sorok tördelése nagyjából ugyanazon algoritmus szerint történik, mint eddig, de a szöveg nem lesz jobbra zárva. Értelemszerűen, ekkor nem kaphatunk `\Underfull \hbox` hibaüzenetet. Vigyázzunk azonban, ha nem szeretnénk az egész szöveget így szedni, ne felejtsük el a megfelelő részt, és természetesen a `raggedright` parancsot is két kapcsos zárójel közé tenni, hogy az érvényességi területet behatároljuk.

```
{\raggedright Ezeket a
sorokat nem akarjuk kizárni
Így nem kapunk túlságosan
nagy, vagy túlságosan kicsi
szóközöket és az ilyenekre
figyelmeztetőüzeneteket sem.}
```

Ezeket a sorokat nem akarjuk kizárni. Így nem kapunk túlságosan nagy, vagy túlságosan kicsi szóközöket és az ilyenekre figyelmeztető üzeneteket sem.

Ha viszont valamely okból az egész szöveget nem sorkizártan szeretnénk szedni, akkor az input legelején adjuk ki a parancsot, az automatikusan végig érvényben marad.

Ha több sort, esetleg egész bekezdéseket vagy az egész szöveget írógép típusú betűkből szedünk, akkor a szöveg ezen részén le kell mondanunk a kizárásról, ahogy az írógép esetében is. Ilyenkor a `\raggedright` parancs speciális változatát, a `\ttraggedright` parancsot kell kiadnunk, ami ebben a stílusban utasítja a programot, hogy adja fel a kizárást, sőt egyben a megfelelő (írógép) stílusra is vált. Ennek elmulasztása viszont feltétlenül hibás sorokat eredményez, rengeteg `Overfull \hbox` és `Underfull \hbox` üzenetet, mivel írógép stílusban a program nem proporcionálisan szed, és így nincs módja a szóközökkel játszva kiegyenlíteni a sorokat. Ezt a stílust alkalmazva a szóelválasztási algoritmus is le van tiltva, tehát a sorok még egyenetlenebbek lesznek. Habár a `\ traggedright` parancs az írógép stílushoz van tervezve, hatással van az egyéb stílusú betűkre is. *Ezért ne felejtsük el ezt a parancsot és tervezett érvényességi körét is kapcsos zárójelek közé zárni.*

Ez egy antikva stílusú sor.
Innentől kezdve írógép stílusban szedjük a szöveget. Figyeljük meg, hogy a sorok nincsenek kiegyenlítve, valamint hogy nem adtuk ki külön a \tt utasítást.

```
Ez egy ... sor.\hfil\break
{\ttraggedright Innentől kezdve
írógép stílusban szedjük a
szöveget. Figyeljük ...

... ki külön a \tt utasítást.}
```

A példában szereplő `\hfil\break` utasítás új sort kezd, leírását a következő szakaszban találhatjuk.

4.2.3. Hogyan befolyásolhatjuk a soremelést?

Ebben a szakaszban két kérdést kell megválaszolnunk: Hogyan kényszeríthetjük a TEX-et arra, hogy egy általa automatikusan ki nem választott helyen sort emeljen, és hogyan adhatjuk meg neki azokat a helyeket, ahol akkor sem emelhet sort, ha egyébként a „véleménye" szerint az lenne a legalkalmasabb hely erre. A második eset szükségességére volt példa a fejezet elején adott 6 órakor. Itt lényegében két megoldás adódik. Az első, az ál-

talában inkább ajánlott a speciális ~ jel használata az inputban[9]. Ez a jel nem kerül ilyen formában az outputba, hanem egy szóköz kerül a helyére, de ezen a helyen a TEX nem fogja megemelni a sort. Tehát a `6~órakor` vagy `12.~fejezet` input szövegrészletek olyan szóközöket jelölnek ki, ahol soremelés nem hajtható végre. Ezzel szemben a szóköz körüli szavakat a program továbbra is elválaszthatja, ha szükséges. Például a második input részletből az outputban elképzelhető, hogy `12. fe-` kerül egy sorba és `jezet` a következőbe.

Előfordulhat azonban, hogy nem egy szóköznél kívánjuk megtiltani a programnak az elválasztást. Mint már említettük, a TEX nem választja el a kötőjeles szavakat. Ezt természetesen úgy kell érteni, hogy az `északi-sarki` szóból nem választja el a `sarki`-t például `sar-ki`-ként, de a kötőjelnél elválasztja a teljes szót. Előfordulhat azonban, hogy bizonyos kötőjeleknél meg szeretnénk tiltani az elválasztást. Ha B–3 szakasz-t szeretnénk írni, ahol a B–3 rész `B--3`-ként kerül megadásra, nyilvánvalóan nem kívánatos a B–3 megtörése (mint ahogy a B–3 és a szakasz külön sorba szedése sem). Ezt úgy érhetjük el, hogy a B–3-at a korábbról már ismert `\hbox` parancsba szedjük, mivel ezeket a TEX nem törheti meg. Így a B–3 szakasz helyes szedése `\hbox{B--3}~szakasz`. Ezt természetesen nem kell minden hasonló esetben ilyen bonyolultan megadni, mivel csak nagyon ritkán fordul elő, hogy a program épp egy kötőjelnél kívánja megemelni a sort. Elegendő, ha csak ezekben az esetekben javítjuk ki a kisebb tipográfiai hibát a fent jelzett módon. Habár a `\hbox` alkalmazása első pillanatban megfelelőnek tünik a szóközöknél alkalmazott elválasztás letiltására is, és az esetek többségében használható is, mi mégis a másik, korábban ismertetett megoldást javasoljuk. Több előnye mellett a legfontosabb, hogy míg a `12.~fejezet` forma megengedi a `fejezet` szó elválasztását, addig ez a `\hbox{12. fejezet}` esetében ez nem megengedett. Ez pedig komoly bajokat okozhat ha a szó a sor végére kerül, azaz éppen akkor, amikor az egész óvatosságra éppen szükségünk van.

Eddig elmondtuk, hogyan akadályozhatjuk meg a programot abban, hogy bizonyos helyeken sort emeljen. Ha viszont arra szeretnénk kényszeríteni, hogy az adott bekezdésen belül egy bizonyos ponton sort emeljen, a `\break` parancsot kell kiadnunk. Ennek hatására a TEX nemcsak sort emel,

[9] Figyeljük meg, hogy itt nem a `\~` ékezetet definiáló parancsról van szó, hanem a ~ önállóan történő alkalmazásáról.

hanem ki is zárja az előző sort. Rossz helyen történő alkalmazása tehát azt eredményezheti, hogy nagyon „laza" sorokat kapunk, amelyek természetesen együtt járnak a megfelelő \Underfull \hbox figyelmeztető üzenetekkel.

```
Ez egy laza sor.\break
Ilyet eredményez a rossz
helyen kiadott utasítás.
```

Ez egy laza sor.
Ilyet eredményez a rossz helyen kiadott utasítás.

Ezt a parancsot leginkább akkor használjuk, amikor túl hosszúra szedett sorok megtörésére akarjuk kényszeríteni a programot. A korábban már említett

```
Overfull \hbox (8.934pt too wide) in paragraph at lines 85--93
[]\tenrm A do-log ak-kor lesz ne-he-zebb, ami-kor az adott|
```

példában megoldhatjuk a problémát, ha az az és az adott szavak közé teszünk egy \break parancsot. Valószínűleg kicsit széthúzott lesz az így kapott

```
A dolog akkor lesz nehezebb, amikor az adott
```

sor és kapunk egy \Underfull \hbox figyelmeztető üzenetet, de ez még mindig jobb megoldás, mint a túl hosszú sor. A \break parancs minden esetben megtöri az adott sort — akkor is ha az üres. Így a \break\break parancsok együttes hatására az éppen adott sor befejeződik (az első \break hatására) és még egy üres sor is keletkezik (a második \break hatására). Persze éppen ilyen hatásos a megfelelő paraméterrel kiadott \vskip parancs is.

A \break parancshoz hasonló a \nobreak parancs, csak éppen ellentétes hatással. Azon a ponton, ahol a bekezdésen belül ez a parancs található, a program semmiképp sem fog sort emelni. Használhatnánk természetesen ezt a parancsot is az előző bekezdés ~ jele helyett, csak éppen egy kicsit nehézkes lenne. Igazi jelentősége a következő fejezetben tárgyalt makró utasítások írása közben, valamint abban az esetben van, ha lapváltás megakadályozására használjuk, ahogy azt a következő szakaszban leírjuk. A \break és \nobreak parancsok két „testvére" az \allowbreak és \goodbreak parancsok. Az első olyan helyen is megengedi a soremelést, ahol az egyébként tilos (például egy szón belül — a vele megtört szó viszont nem lesz elválasztva, az előző sorban hagyott része nem kap kötőjelet), de nem kényszeríti

ki. Azaz az ezzel a paranccsal megjelölt helyet a soremelés szempontjából éppen olyannak tekinti majd a program, mintha az egy szóköz lenne. A \goodbreak parancs pedig egy „viszonylag jó" helyet jelöl ki a soremeléshez, azaz a vele megjelölt ponton nagyobb valószínűséggel történik soremelés, mint máshol.

Megtörténhet, hogy nem akarjuk a sort kizárni soremeléskor. Ha a \break parancs előtt kiadjuk a \hfil parancsot, akkor az kitölti a sor maradék részét üres hellyel és így a \break parancs nem kizárva fejezi be a sort. Az előző példára alkalmazva ezeket, már „hamis" állítást kapunk.

```
Ez egy laza sor.                Ez egy laza sor.\hfil\break
Ilyet eredményez a rossz helyen Ilyet eredményez a rossz
kiadott utasítás                helyen kiadott utasítás
```

Ha több sort akarunk egymás után olyan formában kiíratni, ahogy az az inputban szerepel, kicsit unalmas lehet a sok \hfil\break utasítás kiadása. Helyette használhatjuk azt a megoldást, hogy minden sor után egy üres sort adunk, miáltal a sorok egyenként új bekezdéseket alkotnak, vagy használjuk a \par utasítást, ami nem más, mint ennek az üres sornak a „rövidítése". Ez azonban még mindig túl fáradságos megoldás, nem beszélve arról, hogy így — egyéb utasítások híján — minden sor beljebb kezdődik, mint a bal margó. A TEX-nek van egy speciális parancsa, az \obeylines, amelynek hatására a sorok olyan módon kerülnek törésre, ahogy az az inputban adott. Vigyázat, ez a parancs is — a \raggedright-hoz hasonlóan — a file végéig érvényben marad, ha egyszer kiadjuk és elfelejtjük zárójelek közé zárni. Tehát helyes használatát a következő példa szemlélteti:

```
Ez az első sor.                 {\obeylines Ez az első sor.
Ez a második sor.               Ez a második sor.
Ez pedig a harmadik sor.        Ez pedig a harmadik sor.}
```

4.2.4. Speciális sorok

Néhány esetben egyes sorok a többitől eltérő, speciális bánásmódot igényelnek, akár természetes, akár általunk kijelölt különleges szerepük miatt. A

legjellemzőbb ilyen sor a bekezdés első sora, amely általában beljebb kezdődik, mint a többi. Ennek a mértékét a \parindent paraméter határozza meg a TEX-ben. Az alapértelmezés szerinti értéke 20pt, amely az inputban tetszés szerinti helyen megváltoztatható. Ha azt szeretnénk, hogy valamely bekezdés első sora ne kezdődjön beljebb, akkor a bekezdés előtt adjuk ki a \noindent parancsot. Ennek érvénye csak arra a bekezdésre vonatkozik amelyikben kiadtuk — *a legjobb, ha közvetlenül a bekezdés első betűje előtt, de már a bekezdés kezdetét jelentő üres sor után adjuk ki,* mint ahogy a következő példában:

```
Ez a bekezdés első sora. Közvetlenül az első szó előtt adtuk ki a \noindent parancsot.
```

```
\noindent Ez a bekezdés ...
... parancsot.
```

Amennyiben a fenti hatás ellenkezőjét szeretnénk kiváltani, azaz egy olyan sort szeretnénk kezdeni, amelyik a bekezdés elejét jelölőkhöz hasonlóan beljebb kezdődik, de nem (feltétlenül) kezd egy új bekezdést, az \indent parancsot adjuk ki a sor első szava előtt. Ez a módszer természetesen csak akkor jelent különbséget a szokásos bekezdés-kezdéshez képest, ha a bekezdéseket az első soruk megjelölése mellet valamely más módon is elkülönítjük (erre a következő szakaszban látunk majd példákat). Még annyi haszna lehet az \indent utasításnak, hogy kétszeri használata kétszeres beljebb kezdést jelent — akár a \par\indent akár pedig az \indent\indent kombinációval használjuk.

Ha több egymás utáni bekezdésnél szeretnénk elérni, hogy az első sor ne kezdődjön beljebb, akkor jobb megoldás az első ilyen előtt kiadni a \parindent=0pt parancsot, és az utolsó után — ha ilyen van — de még a rákövetkező előtt az alapértelmezést visszaállító \parindent=20pt parancsot. Az előző szakaszban leírtak szerint a bekezdés sorokká tördelése akkor kezdődik, amikor a bekezdést teljes egészében beolvasta a TEX, tehát logikusan a bekezdésben levő sorokra is a bekezdés végén levő beállítások érvényesek. Ezen szabály alól viszont kivétel a \parindent paraméter értéke. Ezt ugyanis akkor veszi figyelembe a program, amikor a beljebb kezdésnek megfelelő mértéket a legelső sor elé, azaz a bekezdés *legelejére* illeszti, tehát a bekezdés legelején. Így egy bekezdésre az első szava előtt megadott utolsó \parindent paraméter az érvényes.

Előfordulhat, hogy a folyamatos szövegen belül keskenyebb sorokat szeretnénk szedni, akár azért, hogy kiemeljük a mondanivalót, akár valamely más okból. A TEX `\leftskip` és `\rightskip` paraméterei azt az extra mértéket határozzák meg, amennyivel egy-egy sor beljebb kezdődik az oldal bal szélénél, illetve előbb befejeződik az oldal jobb szélénél. Ezek az értékek természetesen általában nullára vannak beállítva, néhány ritka kivételtől eltekinve. Ezek közé tartozik a `\narrower` parancs, amelynek hatására mindkét paraméter a `\parindent` aktuális értéke lesz, és így a program mindkét irányban keskenyebb sorokat fog szedni. A parancs és a paraméterek természetéből következik, hogy egy teljes bekezdésre érvényben vannak, és a bekezdés végén érvényes értékük lesz hatással a tördelésre.

Az ebben a bekezdésben szereplő sorok most keskenyebbek a körülöttük levőknél. De hogy ezt a hatást tényleg elérjük, ki kell adni a `\par` parancsot a `\narrower` parancs érvényességi köre lezárása előtt, hogy még a bekezdés végén is érvényben legyen az utóbbi, és így tényleg keskenyebbek legyenek a sorok.

Az előző bekezdést tehát a következő módon szedtük:

```
{\narrower\noindent Az ebben a bekezdésben szereplő sorok most
...
így tényleg keskenyebbek legyenek a sorok.\par}
```

Figyeljük meg, hogy ez esetben is kapcsos zárójelek közé raktuk a kiszemelt szövegrészletet, jelen esetben a teljes bekezdést. Ez azért történt így, mert nem létezik a `\narrower` parancs hatását közvetlenül megszüntető utasítás, és kétségtelenül kényelmesebb zárójeleket használni, mint az összes átállított paramétert egyenként visszaállítani. A `\narrower` parancsnak megvan az a jó tulajdonsága, hogy kétszeri használata megkétszerezi a hatását. A `\narrower\narrower` utasítások hatására a bekezdésben szereplő sorok kétszer annyival kezdődnek beljebb, mint egy normál bekezdés első sora — a legelső sor háromszor annyival, ha nem szerepel a `\noindent` parancs — és ugyanilyen mértékben előbb fejeződnek be.

A `\leftskip` paramétert használó másik utasítás az `\item`, amelyik olyan bekezdéseket hoz létre, amelyben minden sor — beleértve a legelső sort is — ugyanannyival kezdődik beljebb, mint egy normál bekezdés első sora, de kiér a jobb margóig. Mivel az ilyen típusú szedésre általában akkor van szükség, amikor egy felsorolás elemeit szedjük, az `\item` utasításnak van

egy változója is, amely az egység cimkéjét adja meg. Hasonlóan működik az `\itemitem` utasítás, csak hatására a bal oldalon kimaradó hely a duplájára növekszik. Így a most következő output az utána levő input hatására áll elő.

1. Ez a lista első eleme. A felsorolás több pontot tartalmaz, amelyek közül néhánynak alpontja is van.
   a) Ez a fentebbi eset egyik alpontja. Figyeljük meg, hogy ebben az esetben kétszer akkora a bal margó.
   b) Ez egy másik alpont.
2. A lista második eleme.

```
\item{1.} Ez egy lista első eleme. A felsorolás több
pontot tartalmaz, amelyek közül néhánynak alpontja is van.
\itemitem{a)} Ez a fentebbi eset egyik alpontja. Figyeljük
meg, hogy ebben az esetben kétszer akkora a bal margó.
\itemitem{b)} Ez egy másik alpont.
\item{2.} A lista második eleme.
```

Vigyázzunk arra, hogy az itt említett parancsok mindenképp kiszednek valamit az egységek cimkéjeként. Ha nem adunk meg változót, akkor a kiszedendő szöveg első betűjét fogják lecsípni és külön szedni, valahogy így:

E lső elem. `\item Első elem.`

Tehát, ha nem szeretnénk cimkét hozzárendelni a felsorolás egy eleméhez, vagy felsoroláson kívül használjuk az `\item` vagy `\itemitem` utasítás valamelyikét, akkor jelöljünk ki a számára egy üres változót az `\item{}` mintájára.

A következő, `\line` mintájú parancsok különböző módon elhelyezett egyes sorok kiszedésére szolgálnak. A legegyszerűbb közülük maga a `\line` parancs, amely lényegében nem csinál mást, mint a változójában megadott szöveget tartalmazó sort kiszedi. Fontossága nem is önmagában rejlik (elég nehéz olyan helyzeteket elképzelni, amikor használatára valóban szükségünk van), hanem különböző változataiban. Ezek a `\centerline`, `\leftline` és `rightline` parancsok, amelyek a hosszuktól függetlenül középre, balra, illetve jobbra zárt sorokat szednek ki. A sorok tartalma természetesen a parancsok változójában megadott szövegrészlet lesz. Ezek a sorok a szokásosnál hosszabbak vagy rövidebbek is lehetnek, a TEX ezt nem veszi

figyelembe és nem ad hiba vagy figyelmeztető üzeneteket sem. A hosszabb sorok a `\centerline` parancs esetében a sor mindkét szélén túlnyúlnak, a `\leftline` parancs esetén a jobb margóba érnek, a `\rightline` esetén pedig a bal margóba.

Ez egy rövid, balra zárt sor.

Ez egy jobbra zárt.

Középre zárt.

Ez pedig egy, az előzőeknél hosszabb, balra zárt sor.

Hasonló, mint a fenti, de ezt jobbra szeretnénk zárni.

Ha ez a sor túl hosszú lenne, akkor mindkét oldalt a margóba érne.

Ezekben a példákban a sorokat rendre a `\leftline`, `\rightline`, `\centerline` és megint a `\leftline`, `\rightline` és végül a `\centerline` parancsok alkalmazásával szedtük ki.

4.3. Hogyan tördeli a TEX a bekezdéseket?

Némely paraméter vagy a tördelést szabályozó parancs esetén nehéz eldönteni, hogy az a sorok bekezdéssé való tördelését, vagy a bekezdések (oldalakká való) tördelését szabályozza. Így elképzelhető, hogy ebben a szakaszban több olyan téma is szerepel majd, amit az olvasó az előző szakaszba osztott volna be, és esetleg az előző szakasz néhány fogalmát jobban szeretné itt látni. Reméljük azonban, hogy mindezek ellenére nem lesz nehéz megtalálni a megfelelő fogalmakat.

4.3.1. Sortávolság

Amikor a TEX elkészül az egy bekezdésbe kerülő sorok tördelésével, kiírja azokat az outputba. Ilyenkor megnyitja a megfelelő `.dvi` file-t és kiegészíti a sorokat tartalmazó, „dvi-nyelven" írt parancsokkal. De ehhez nem elég a sorok ismerete, tudnia kell azt is, hogyan szervezze ezeket bekezdéssé. Az azonos bekezdésbe kerülő sorok esetén ez csak annyit jelent, hogy ismernie kell a sorok közötti távolságot. Ennek meghatározása viszont nem olyan egyszerű, amilyennek az első látszatra tünik. Természetesen addig semmi

komoly probléma nem merülhet fel, amíg az egymás utáni sorok azonos nagyságú (és az esetek többségében típusú) betűkből, és csak ezekből állnak. De ha valamely okból a betűnagyság változik, vagy — főleg technikai jellegű, matematikai szövegek esetén — képletek és egyéb, nem szövegjellegű részletek is megjelennek a sorokban, akkor mindez nehezebbé válik. A TEX igyekszik olyan algoritmusokat használni, amelyek a lehetőségekhez képest automatikusan határozzák meg a sortávolságot, és csak akkor válnak bonyolultabbá, ha a szöveg jellege is bonyolult lesz.

Ahhoz, hogy megértsük a sortávolságot vezérlő algoritmust, a sort kell közelebbről megvizsgálnunk. A sorhoz szorosan hozzátartozik három vízszintes vonal, amelyek közül talán a legfontosabb az alapvonal, amelyiken a szokványos betűk, mint amilyen például az `a,m,o,x` „ülnek". A két másik vonal a sor fizikai alját és tetejét jelöli ki. Ha csak betűk vannak a sorban, és nincs köztük olyan, amelyik az alapvonal alá nyúlna — mint amilyen az `y,g,j`, akkor a sor alját jelölő vonal megegyezik az alapvonallal. Az esetek többségében azonban ez nem áll fenn, és lejjebb lesz az alapvonalnál. Azt a távolságot, ami a szokványos esetekben a két egymás alá kerülő sor alapvonalát elválasztja egymástól a `\baselineskip` paraméter jelöli ki. Ennek alapértelmezés szerinti beállítása (amely azt tételezi fel, hogy tízpontos betűnagyságot használunk) `12pt`. Amíg csak tízpontos betűk vannak egy sorban, és normál szöveget szedünk, valóban ez a beállítás adja a legszebb megjelenést a soroknak. Ha viszont valamely okból egy sorban mondjuk tizenkét pontos betűk is szerepelnek, akkor ezek közül néhányan (a magasabbak) maguk is kitöltik a tizenkét pontnyi helyet a két sor között. Ha a felettük levő sorba olyan betűk is kerülnek, amelyek az alapvonal alá nyúlnak, akkor ettől a beállítástól akár részben egymásra is nyomtatódhatna két egymás fölötti betű. Ennek megelőzésére szolgálnak a `\lineskiplimit` és `\lineskip` paraméterek. Ha a `\baselineskip` paraméter alkalmazásával egy sor fizikai alja közelebb kerülne az alatta levő sor fizikai tetejéhez, mint a `\lineskiplimit` által meghatározott érték, akkor a két sor távolságát nem a `\baselineskip` paraméter fogja meghatározni. Ehelyett oly módon kerülnek elhelyezésre, hogy a fölső alja és az alsó teteje között a `\lineskip` paraméter által meghatározott távolság maradjon ki. Az alapértelmezés szerinti beállítás `\lineskiplimit=0pt` és `\lineskip=1pt`, azaz két sor semmiképp sem kerülhet egymásra, ez esetben plusz egy pontnyi távolság adódik a kettő közé.

Az eddigi beállítások tökéletesen működnek mindaddig, amíg tízpontos betűkkel és szokványos szöveggel dolgozunk. Minden egyéb esetben (nagy valószínűséggel) meg kell változtatni az alapértelmezés szerinti beállításokat. Ezekre az esetekre természetesen nem térhetünk ki egyenként, csak egyet említünk meg példaként. Tegyük fel, hogy a szövegben egy bekezdést a szokásos tízpontos betűnagyság helyett tizenkét pontossal szeretnénk szedni. Nem teljesen véletlen, hogy ezt a példát választottuk ki az addig szereplő paraméterek szemléltetésére. Ha a teljes szöveget szeretnénk tizenkét pontos betűnagysággal szedni, akkor sokkal egyszerűbb, ha tízpontos betűkből kiszedjük — a megfelelő paramétereket a true lehetőséggel beállítva — és utána a `\magnification=\magstep1` parancsot az input elején kiadva az outputot 1,2-szeresére nagyítva nyomtatjuk ki. Ha pedig nem egy teljes bekezdésben adódik problémánk a sortávolsággal, csak néhány sorában, akkor annak megoldására nem ezek a paraméterek szolgálnak. Ezek ugyanis jellegükből adódóan a teljes bekezdésben érvényben vannak.[10] Az ilyen problémás sorok kezelését vagy automatikusan megoldja a TEX — mint például a matematikai szövegben a kiemelt képletek esetében — vagy pedig sokkal nehezebb a kezelésük és más, részben még ebben a szakaszban ismertetett módszereket kíván.

Térjünk tehát vissza a kiválasztott példánkhoz, a tizenkét pontos betűkből álló bekezdés alkalmazásához egy olyan szövegben, amelyben jórészt tízpontos betűket használunk. A jelen példa nem tökéletes, mivel esetünkben a teljes könyv tizenegy pontos betűnagysággal készül, de ez a lényeget nem érinti. Mindenesetre a bekezdés elkészítéséhez be kell állítani a nmegfelelő paramétereket a tizenkét pontos betűnagysághoz. Mivel ezen paraméter-értékeknek csak az adott bekezdésre kell vonatkozniuk, vagy kapcsos zárójellel jelöljük ki ezt a területet és azon belül adjuk meg a paraméterek új értékeit (ez az egyszerűbb megoldás), vagy pedig a bekezdés lezárása után újabb parancsokkal visszaállítjuk a régi értékeket. Ami pedig a paramétereket illeti, közülük legalább a betűnagyságot (és vele együtt a betűtípust) kijelölő paramétert és az alapvonalak távolságát meghatározó `\baselineskip` paramétert át kell állítani. A mostani példában a tizenkét

[10] Még pontosabban a teljes bekezdésre azon értékeik lesznek hatással, amelyek a bekezdés végén érvényben vannak.

MAGYAR TUDOMÁNYOS AKADÉMIA

pontos betűnagyságnál az alapvonalak távolságát célszerű körülbelül tizennégy pontra beállítani. Az általános szabályok megadására itt nincs helyünk és módunk — ez már különben sem a program ismertetéséhez tartozik, hanem a tipográfia hatáskörébe. Ha valakinek csak egyszer-egyszer van szüksége ilyen típusú problémák megoldására, néhány kísérlet után mindig megtalálhatja az esztétikus sortávolságot. Ha viszont rendszeresen kell ilyen problémákat megoldani egy tipográfiához nem értő személynek, akkor ott a TEX program nem tud segíteni. A megoldás ilyen esetben a tipográfia megtanulása vagy a feladat megoldásához egy tipográfus alkalmazása.

Az újabb kitérő után megint tekintsük a példánkat, a tizennégy pontos sortávolságot. Mostantól azonban már kilenc pontos betűnagyságot használunk, hogy több példán szemléltethessük az eljárást. Az előző bekezdés megadásához tehát beállítjuk a betűnagyságot és a `\baselineskip` paramétert. Beállíthatnánk még a `\lineskiplimit` és `lineskip` paramétereket is, de erre nincs szükségünk valójában. Mint említettük, ezek csak a kivételes esetek kezelésére szolgálnak — arra pedig alkalmas az alapértelmezés szerinti értékük is. Bonyolultabb feladatokat az egyes sorok figyelembevételével kell megoldani kézi módszerekkel, ahogy majd a továbbiakban látjuk. Így tehát a jelen és a megelőző bekezdést a következő formában adtuk meg a könyv inputjában:

```
{\font\tketant=hamr10 scaled \magstep1
\tkeant\baselineskip=14pt
Térjünk tehát vissza a kiválasztott ... megoldásához egy
tipográfus alkalmazása. \par}
{\font\kilant=hamr9
\nylant\baselineskip=11pt
Az újabb kitérő után megint tekintsük ...a következő
formában adtuk meg a könyv inputjában: \par}
```

Amennyiben csak egyes esetekben szeretnénk a sorok közötti távolságot megnövelni, az előzőektől eltérő megoldást kell alkalmaznunk, és sokkal óvatosabbnak is kell lennünk. Általában, ahogy azt már láttuk, a `\vskip` paranccsal és annak különböző változataival definiálhatunk függőleges kihagyandó beosztástt. A `\vskip` parancsnak meg kell adnunk a változóját, de ennek egyre növekvő fokozatú előre definiált változatai is léteznek, amelyek már magukban működnek. A `smallskip`, `\medskip` és `\bigskip` parancsok az adott sorrendben 3, 6, illetve 12 pontos függőleges ugrásokat jelölnek ki, amelyek mértékük harmadáig még

rugalmasak is. Nincs is semmi probléma addig, amíg ezeket két bekezdés illetve bekezdés „értékű" szövegrészlet között használjuk. Ha viszont egy bekezdésen belül adjuk ki valahol ezt a parancsot, akkor elég nehéz kiszámítani a hatását. Most például az előző sorokban a `harmadáig még` szövegrészlet után közvetlenül kiadtunk egy `\medskip` utasítást. Mindenki látja a hatását, ami azonban nehezen számítható ki előre. Jelen bekezdés befejezése után is szerepel egy `\medskip` parancs, annak hatása viszont már most megmondható: a bekezdés után a szokásosnál körülbelül 6 ponttal több hely marad ki a következő bekezdés előtt. Egy bekezdésen belül általában az a szabály, hogy azon sor után marad ki a `skip` típusú paranccsal kiadott hely, amely sor a parancsot tartalmazza — csak éppen ezt nehéz kiszámítani.

Amennyiben a bekezdések között a szokásos `\baselineskip` — és esetleg a hozzá tartozó `\lineskip` — parancs(ok) által meghatározott távolságnál nagyobbat szeretnénk kihagyni, akkor azt nem érdemes minden egyes alkalommal külön-külön a `\vskip` vagy hozzá hasonló parancsokkal beállítani. Helyettük használhatjuk a `\parskip` paramétert, amelynek értéke az a *plusz* távolság, amennyivel több hely marad ki egy bekezdés záró és a következő bekezdés nyitó sora között a szokásosnál. A TEX alapértelmezése szerint ez 0 pont — pontosabban 0 pont, amely szükség esetén egy pontra kinyúlhat, ha az oldal kiegyenlítése ezt kívánja —, de más beállítása is lehetséges. A `parskip` mértékét tanácsos az egész szövegre terjedően egyöntetűen definiálni, mi mégis megemlítjük, megváltoztatása először arra a két bekezdés közötti helyre vonatkozik, amely az új érték után következik. Más szóval ez azt jelenti, hogy értéke a bekezdés *előtti* plusz távolságot befolyásolja, nem pedig a bekezdés utánit. Még egy fontos dolgot kell megemlítenünk a fenti `skip` típusú parancsokról: Az általuk meghatározott extra függőleges távolság mindig hozzáadódik a `\baselineskip` és társai által meghatározott sorok közötti távolsághoz.

4.3.2. Lapemelés

A TEX-nek van egy — a soremeléshez hasonló — lapemelő algoritmusa is. Ez annyival egyszerűbb, hogy az oldalra egy — a soremelő algoritmus által már kiszedett — sor mindig vagy teljes egészében ráfér, vagy egyáltalán

nem. Így a lapemeléskor csak a sorok közötti távolsággal lehet „játszani", hogy beállítsuk az oldal előírt magasságát, a sorokat azonban függőlegesen (természetesen) nem lehet megtörni. Ugyanakkor, miként a soremelés esetén is, lapemeléskor a program igyekszik mindazokat az igényeket figyelembe venni, amelyeket az átlag tipográfus figyelembe vesz. Így például csúnyának tekinti, és a sormeléskor már ismertetett pontozási skálán sok „büntetőponttal jutalmazza" a fattyúsorokat[11], illetve az oldalvégi elválasztott szavakat.

A lapemelés esetében nem ismertetjük a TEX algoritmusát, mivel a program használatához erre kevésbé van szükség, mint a soremelési algoritmus ismeretére. Mindössze az algoritmus befolyásolásához szükséges tudnivalókat adjuk meg. Az első fontos tudnivaló a bekezdések hosszára vonatkozik. Mint már írtuk, a program bekezdésekként állítja elő a szöveget, aztán pedig ezekből a bekezdésekből az oldalt. Ha tehát a teljes szöveget egyetlen bekezdésbe szedjük, akkor csak a teljes input feldolgozása után lesz képes az oldalakra való tördelést elvégezni. A számítógép memóriája viszont egyszerre csak korlátozott mennyiségű szöveget képes befogadni a TEX által használt formában. Gyakorlatilag ez azt jelenti, hogy ha körülbelül 20000 betűt adunk meg a programnak egyetlen bekezdésben, akkor azt már nem képes tárolni, és egy

```
! TeX capacity exceeded, sorry.
```

hibaüzenet kiséretében leáll. Szemléletesen ez a 20000 betű körülbelül 7 olyan oldalnak felel meg, mint ez a könyv. Mivel általában nem írunk ilyen hosszú bekezdéseket, ettől a problémától a gyakorlatban nem kell tartani. Ha valakinél mégis jelentkezne, akkor a bekezdések sűrűbb használatával megszüntethető.

Ha egy adott oldal elegendő rugalmasságot mutat függőlegesen, akkor nem nehéz a programnak megfelelő helyet találni a lapemelésre. Ez esetben ugyanis egy-két sort még átvihet a következő oldalra, illetve áthozhat onnan az éppen aktuális oldalra a rugalmas helyek egyenként kicsi, de összességében már viszonylag nagy változtatása árán, és ezen az áron megtalálhatja a legjobb lapemelési pontot. Ez azonban sok esetben — főleg különleges szö-

[11] Fattyúsornak nevezzük az oldal elején egy bekezdés utolsó sorát, amely így egyedül jelenik meg a bekezdésből az oldalon. Az alsó fattyúsor az oldal alján az új bekezdés első sora.

vegek esetében, amelyek sok kiemelést, táblázatot tartalmaznak — nincs így, aminek következtében a lapemelés az ilyen esetekben a soremelésnél is nehezebb feladat a TEX számára.

A nem megfelelő lapemelés „eredménye" a soremelési algoritmus elemzéséből már ismert `Overfull` és `\Underfull` figyelmeztető üzenetek, amelyekhez ekkor a `\vbox` kiegészítés járul. Ha egy oldalra a TEX nem tud annak kitöltésére elegendő anyagot tenni, illetve csak úgy tudja az oldalon szereplő rugalmas távolságokat megnyújtani az oldal teljes egészében való kitöltésére, hogy ezáltal azok túlságosan megnyúlnak, akkor egy

```
Underfull \vbox (badness 9897) has occured while output is ...
```

típusú figyelmeztető üzenetet ad. Ilyenkor biztosak lehetünk benne, hogy az így megadott oldalon túlságosan nagy szabad helyeket találunk. Az esetek többségében kicsit talán kedvezőbb megoldás, ha a program egy-egy oldalt a szokásosnál valamelyest hosszabbra tördel. [12] Ez ugyanis nem nagyon tűnik fel, feltéve persze, hogy nem nagy mértékű az eltérés. Ezt az esetet, amint azt várjuk is már, egy

```
Overfull \vbox
```

típusú figyelmeztető üzenet hozza a tudtunkra.

Ha tehát túlságosan technikai szövegeket szedünk, vagy egyszerűen az átlagnál kevésbé szerencsések vagyunk, akkor előbb-utóbb meg kell ismerkednünk a lapemelési algoritmust befolyásoló parancsokkal is, amelyek nagyon hasonlóak a soremelést befolyásolókhoz, sőt néha megegyeznek azokkal. A `\raggedright` parancshoz hasonló a `\raggedbottom`, amelyik arra utasítja a programot, hogy ne törekedjen az oldalak egyforma magasra szedésére, csak arra vigyázzon, hogy ne legyenek hosszabbak, mint a megadott `\vsize` érték. Persze ennek érvényessége esetén is törekszik a program minél teljesebben kitölteni az oldalt, nem fog csak néhány sorból álló oldalakat létrehozni. Hasonlóan a `\raggedright` parancshoz, ezt is kapcsos zárójelek közé kell zárni, ha nem kívánjuk az érvényességét a teljes szövegre kiterjeszteni. De, a `\raggedright` paranccsal ellentétben, ennek megvan az inverz

[12] Sajnos a TEX-nek is megvannak a maga „Murphy törvényei". Ezek egyike pedig azt mondja ki, hogy ha a TEX két rossz megoldása közül valamelyik kedvezőbb a számunkra, akkor az sokkal ritkábban történik.

utasítása is. A \normalbottom hatására a TEX visszaállítja az alapértelmezést, azaz igyekszik az oldalakat mindig úgy kitölteni, hogy azok magassága megegyezzék a \vsize által előírt magassággal.

Amennyiben kifejezetten ki akarunk jelölni egy pontot a lapemelésre, vagy meg akarjuk tiltani egy ponton a lapemelést, két eset lehetséges. Ha két bekezdés között, vagy hasonló jellegű helyen vagyunk (például egy \centerline-nal szedett címsor és az utána következő bekezdés között), akkor az \eject parancs hatására a TEX itt oldalt vált és a soremelésből már ismert \nobreak parancs hatására semmi esetre sem történik lapemelés.[13] Az \eject paranccsal éppúgy vigyáznunk kell, mint a \break paranccsal a soremeléskor: Amennyiben túl kevés anyag van még az adott oldalon és kiadjuk az \eject utasítást, a program „széthúzza" a sorokat és természetesen egy \Underfull \vbox figyelmeztető üzenetet ad. Ha tehát nem a rossz oldalemelés javítására kívánjuk használni, hanem egy, a szokásosnál kevesebb szöveget tartalmazó, és így rövidebb oldal létrehozására, akkor helyesen így kell kiadni: \vfil\eject.

Nehezebb, habár sokkal ritkábban előforduló feladat ha az oldal emelését egy bekezdés belül kívánjuk a fenti módon befolyásolni. Ilyenkor ugyanis a \nobreak parancs ilyen egyszerűen nem használható, hiszen csak a soremelést tiltaná le. Az \eject parancs pedig először befejezi az éppen beolvasott bekezdés szedését, és csak aztán érezteti hatását. Anélkül, hogy bővebben kitérnénk rájuk, vagy akár használatukat különösebben javasolnánk, elmondjuk, hogy ezekben az esetekben a \vadjust{\nobreak} és \vadjust{\eject} parancsok teszik meg a kívánt hatást. Persze még ezen parancsok hatására is teljes egészében szedésre kerül az a sor, amely őket tartalmazza, és csak azután hajtja végre illetve tiltja meg az oldaltörést a program. A soremelés és a lapemelés kombinációjából kapott

```
\hfil\break\vadjust{\vfil\eject}
```

parancs hatására az adott helyen befejeződik az éppen szedett sor és az éppen szedett oldal is. Persze ezt a lehetőséget csak illusztrációnak szántuk, anélkül, hogy használatára bátorítanánk.

Végül kitérünk azokra az utasításokra, amelyek az oldal megemelését nem adják ugyan ki egyértelműen, de olyan helyeket jelölnek ki, ahol valame-

[13] Pontosabban szinte egyetlen esetben sem. De ha ugyanott kiadunk valamely olyan parancsot, amelyik kierőlteti az oldalemelést, akkor a \nobreak hatástalan lehet.

lyest szívesebben látnánk az oldalemelést. A bekezdéseket alkotó folyamatos szövegen kívül kiadott \allowbreak és \goodbreak parancsok hasonló hatással vannak az oldalemelésre, mint az egyéb helyen a soremelésre, azaz az előző az egyébként tiltott helyeken is megengedi az oldalemelést, míg az utóbbi viszonylag nagy mértékben megemeli annak a valószínűségét, hogy az általa megadott helyen történjék az oldalemelés. Ennél kisebb mértékben, a megadott sorrendjük szerint növelik ezt a valószínűséget a \smallbreak, \medbreak és \bigbreak parancsok, amelyek az oldalemelés bátorításán kívül még a fokozatuknak megfelelő skip távolságot is megadják. Hármuk közül a \smallbreak növeli a legkevésbé annak a valószínűségét, hogy az adott helyen oldalemelés következzék be, és ugyanakkor kijelöl egy \smallskip-nyi távolságot is. Ezen parancsoknak a sajátossága még, hogy a megadott távolságokat nem adják hozzá a más parancsok által ugyanazon a helyen kijelölt távolsághoz, hanem — ha ilyen van — annak és az általuk megadottnak a maximumát veszik.

4.4. A TEX különböző módjai

Ez a szakasz nem feltétlenül szükséges a TEX szokványos használatához, a kevésbé érdeklődők ki is hagyhatják (ezért is került a fejezet végére). Az általános tördelési algoritmus megértéséhez és bizonyos ritkán előforduló hibák kijavításához viszont gyakran hasznos lehet, annak ellenére, hogy itt sem törekedünk teljességre.

Ahhoz, hogy a program hatékonyan tudja végrehajtani az előző szakaszokban leírt algoritmusokat, létrehozói különböző „állapotokra" osztották működését. Ezek az állapotok a következők:

- Vízszintes állapot, amikor a program a bekezdésekben található vízszintes objektumokat, azaz az esetek többségében a sorokat dolgozza fel.
- Belső vízszintes állapot, amikor a program a \hbox parancsok változóit dolgozza fel.
- Függőleges állapot. Ebben a TEX az oldalt alkotó függőlegesen egymásra helyezett összetevőket dolgozza fel.

- Belső függőleges állapot, amikor a program a \vbox parancsok változóit dolgozza fel.
- Matematikai állapot. Ilyenkor a sorokon belüli matemetikai képletek kiszedése történik. Ennek részletes tárgyalása a következő fejezetben történik.
- Kiemelt matematikai állapot. Ebben az állapotban a program a külön sorba kiemelt matematikai képleteket szedi ideiglenesen megszakítva az ezen kívül éppen érvényben levő állapotot.

Mint mondottuk, általában nincs szükségünk arra, hogy megértsük a TEX milyen állapotban van éppen. De ha kíváncsiak vagyunk, hogy miként is működik a \nobreak parancs, amely egyes esetekben a soremelést más esetekben pedig az oldalemelést tiltja meg, akkor az állapotok közötti különbségeket kell megértenünk. Súlyosabb eset, ha egy, a következő mintájú hibaüzenetet kapunk.

```
! You can't use .... in .... mode
```

Az üzenet pontos jelentése szerint nem használhatjuk az első kipontozott helyen szereplő parancsot vagy egyéb „TEX-szerű" objektumot a második kipontozott helyen meghatározott állapotban. Természetes, hogy az ilyen hibaüzenet megértéséhez ismernünk kell az állapotok legalább vázlatos leírását.

Kicsit jobban megfigyelve a fentebbi hibaüzenetet azonnal világossá válik mindenki számára, hogy bizonyos parancsok csak bizonyos állapotokban adhatók ki. Talán a legjellemzőbb példa a matematikai és vízszintes állapotok közötti különbség: Az 5. fejezetből kiderül, hogy többek között az ékezeteket és az aláhúzást más és más parancs adja meg a két állapoton belül. Egy másik, a felhasználó számára is fontos jellemzője az állapotoknak, hogy a TEX különböző állapotokban különböző módokon hajthatja végre ugyanazt a parancsot, mint a már sokszor említett \nobreak példája mutatja.

Az állapotok használatának alapvető szabályait az állapotok neve adja meg. A program általában valamely vízszintes állapotban van, ha vízszintesen „építkezik", valamely függőleges állapotban, amikor függőlegesen „építkezik" és valamelyik matematikai állapotban, amikor matematikai képleteket szed. De ez a szabály néha fordítva is igaz, azaz a vízszintes állapotokban a program „vízszintesen" fog cselekedni, míg a függőleges állapo-

tokban „függőlegesen", hacsak az éppen aktuális parancs nem tartalmazza, hogy végrehajtásához a programnak milyen állapotban kell lennie, illetve milyen állapotba kell váltania. Így például ha a TEX egy \nobreak utasítást, vagy ehhez hasonló olyan utasítást talál, amely a sor- és oldalemelési algoritmusokhoz tartozó „büntetőpontozással" kapcsolatos, akkor attól függően alkalmazza azt, hogy milyen állapotban van. Azaz vízszintes állapotban a soremelést befolyásolja a parancs, míg függőleges állapotban az oldalemelést. A \kern utasítás is aszerint járul hozzá vízszintesen vagy függőlegesen az oldal kiterjedéséhez az általa megadott dimenzióval, hogy a program vízszintes vagy függőleges állapotban van. Ha a TEX egy \hbox{dots} utasítást talál, miközben vízszintes állapotban van, akkor a kiszedett tartalmát a vízszintes listához, azaz az éppen épített sorhoz adja hozzá — és mivel a \hbox tartalmát nem tördelheti több sorba, hibát jelez, ha az túlságosan a sor végére kerül és belenyúlik a jobb margóba — míg függőleges állapotban a függőleges listához teszi, azaz új sorban helyezi el.

Természetesen mivel direkt módon csak ritkán adjuk meg, hogy milyen állapotban legyen a TEX[14], létezik egy algoritmus, amely ezeket az állapotokat váltogatja a program számára. Általánosságban azt mondhatjuk, hogy vízszintes állapotba kerül a program, ha valami olyan dolgot talál, ami egyértelműen a vízszintes listában dolgozandó fel, mint például ha bekezdés kezdetét találja, a \hskip parancsot vagy annak valamely változatát, amilyen a \quad, vagy például a \hrule parancsot. Hasonlóan ehhez a szabályhoz, a TEX függőleges állapotba vált — és egyben be is fejezi az éppen szedett bekezdést — amennyiben olyan parancsot talál, aminek „semmi köze" a vízszintes listához. Ilyen parancs többek között az üres sor vagy az annak megfelelő \par parancs vagy például a \vskip utasítás. Ezekhez hasonló a \vbox is, amely szintén lezárja a megkezdett bekezdést és a benne megadottak — értelemszerűen — függőlegesen adódnak az oldal tartalmához.

A program időnként megszakítja az éppen használt állapotot, egy időre másik állapotba vált, de nem felejti el a megelőző álapotot sem, és a megfelelő utasításra visszaváltó. Ilyen eset, amikor a két bekezdéssel ezelőtt ismer-

[14] Általában a $ és $$ parancsokkal megadjuk, hogy a matematikai állapotok egyikébe, illetve abból vissza váltson és a \hbox illetve \vbox parancsokkal és azok változójával, hogy valamelyik belső állapotba és abból vissza váltson. Azt azonban általában nem adjuk meg, hogy mikor váltson vízszintes illetve függőleges állapotba.

tetett módon a \hbox parancsot hajtja végre. De talán ennél is jellemzőbb a matematikai állapotba váltás, ami általában vízszintes állapotból történik. A $ jel hatására a TEX megszakítja a vízszintes állapotot, feldolgozza az utána következő matematikai képletet, majd az újabb $ jel hatására visszatér a vízszintes állapotba és a képletet hozzáadja a sorokat alkotó vízszintes listához.

5. fejezet

Matematikai szövegek szedése

Cikkek, könyvek, dokumentációk szedésének legnehezebb és legköltségesebb része a formulák, matematikai, műszaki képletek szedése. Különlegesen nagy gyakorlat és hozzáértés kell ahhoz, hogy valaki matematikai képleteket tudjon gyorsan, szépen szedni. A matematikai formulák szedésének ugyanis külön tudománya van: bonyolult elhelyezési szabályok, ill. konvenciók alakultak ki, amelyek be nem tartása nehezíti, esetleg lehetetlenné teszi a szöveg megértését. A TEX a legnagyobb segítséget éppen az ilyen szövegek előállításánál jelenti a felhasználónak: lehetővé teszi azt, hogy ezen konvenciók ismerete nélkül is lehessen helyesen matematikai szöveget írni még akkor is, ha a formulák, jelölések tartalmáról — nem matematikus lévén — a felhasználó szinte semmit sem tud. A TEX ugyanis különösen matematikai szövegek szedésében „szakértő", a formulák szedésénél bátran hagyatkozhatunk a TEX egyedülálló szakértelmére, az elhelyezést, betűtípust stb. illetően. Ahhoz, hogy matematikai szöveget szedjünk matematikai módba kell váltanunk. A matematikai mód a TEX-en belül külön világ. Itt a TEX nagyobb szabadságot élvez a jelek, betűk elhelyezésében mint a „rendes" szövegmódban. A könyvben eddig elmondott szabályok csak részben igazak a matematikai módon belül. Lássuk hát, mit jelent a matematikai mód !

- Matematikai módba a $ jellel lehet átkapcsolni. Ha például azt akarjuk leírni, hogy $a \cdot x + b$, akkor azt kell begépelnünk, hogy `$a\cdotx+b$`. A második $ jel a matematikai mód kikapcsolására szolgál. Ha egyszer a $ jellel bekapcsoltuk a matematikai módot, akkor maradunk is ebben a módban egészen addig, míg egy $ jellel ki nem kapcsoljuk.
- Matematikai módban a TEX minden karaktert ún. matematikai kurzívban szed. Ez különbözik a szövegben használatos kurzívtól: *Ez nem*

matematikai, hanem szövegkurzív, ez pedig a matematikai kurzív betűtípus.

- Matematikai módban a TEX nem veszi figyelembe a forrásszövegben lévő szóköz és sorvége karaktereket. Láttuk, hogy „normál" szövegmódban csak egy szóközt vett figyelembe a TEX, a második, harmadik stb. szóközöket lenyelte. Matematikai módban egyetlen szóközt sem vesz figyelembe. Tehát ha a következőt gépeljük be: `$x+y=z$`, vagy `$x + y= z$`, vagy `$x+ y = z$` mindig ugyanazt, az $x+y=z$ egyenletet kapjuk. A sorvége karakter ugyanúgy mint szöveg módban, szóköznek számít, tehát matematikai módban lenyeli. A szóköz karakter egyetlen helyen fontos a matematikai módban: a parancsszó végén. Ugyanúgy, mint szöveg módban, most is innen tudja a TEX, hogy vége van a parancsszónak.

- Matematikai módban részben megváltoznak a parancsjelek és parancsszavak is. Vannak olyan parancsok, amelyek csak matematikai módban, és vannak olyanok, amelyek csak szöveg módban használhatók. Egyes parancsok jelentése megváltozik. Matematikai módban *nem használhatók* a MATEX programcsomag által biztosított ékezetes betűk! Ekkor ugyanis a karaktereket más módon dolgozza fel a TEX.

Használható viszont a *felkiáltójel* `!`, amelyet szövegmódban a `\?` parancsjellel tudtunk előállítani. A `\?` parancsjellel matematikai módon belül is készíthetünk felkiáltójeleket, másrészt szövegközben is előállíthatunk felkiáltójelet ha ezt gépeljük be: `$!$`

Ékezetekes betűk készítésére *nem használhatók a 2. fejezetben leírt parancsok sem,* hanem a következőket alkalmazhatjuk:

`$\hat a$`	$\hat{a}$
`$\check a$`	$\check{a}$
`$\tilde a$`	$\tilde{a}$
`$\acute a$`	$\acute{a}$
`$\grave a$`	$\grave{a}$
`$\dot a$`	$\dot{a}$
`$\ddot a$`	$\ddot{a}$
`$\breve a$`	$\breve{a}$
`$\bar a$`	$\bar{a}$
`$\vec a$`	$\vec{a}$

A matematikai kifejezésekben — és csak ott — tehát csak ezekkel a parancsokkal tudunk ékezetes betűket szedni. Ezek használata meglehetősen kényelmetlen, de ha meggondoljuk, hogy a képletekben milyen gyakran (pontosabban milyen ritkán) fordulnak elő ékezetes betűk, meggyőződünk arról, hogy ezek szerepe, és az okozott kényelmetlenség elenyésző.

- Matematikai módban néhány jel értelmezése is megváltozik:
  1. A kötőjel (-) jelentése matematikai módban kivonás jel: $-$.
  2. A / jel jelentése törtjel: $/$.
  3. A függőleges vonal az abszolútérték jel lesz, például `$|x|$`-ből $|x|$ lesz.

Ennél kissé bonyolultabb a zárójelek értelmezése. A () zárójeleket a TEX továbbra is mint gömbölyű zárójeleket (()) jeleníti meg, de a kapcsos {} zárójelek két feladatot is ellátnak: egyrészt a TEX szokásos környezetkijelölő jelei, és mint ilyenek nem jelennek meg a formázott szövegben; másrészt a matematikai szövegek tartozékai, ilyenkor megjelennek a formázott szövegben. A {} jelek általában az első szempont szerint értelmeződnek; pl. a `${x+y}-z$` kifejezésből $x+y-z$ lesz. Ha azt akatjuk, hogy megjelenjenek, a \{ \} parancsjeleket kell használnunk: `$\{x+y\}-z$`, ${x+y}-z$.

Meg kell jegyeznünk, hogy a felesleges zárójelezés hibához vezethet. A TEX a matematikai szedési-konvenciók szerint helyezi el a betűket, jeleket. Ha például azt írjuk, hogy `$x+y$`, akkor ez így jelenik meg: $x+y$. Ha viszont ezt írjuk: `$x{+}y$`, akkor ez $x{+}y$ formában jelentkezik. Az utóbbi ugyan nem jelentős, de hiba, mivel a jelek között nem a megfelelő távolság van.

Matematikai szövegek szedésénél az a legfontosabb, hogy szedni tudjuk a matematikára jellemző sokféle szimbólumot. Ezek egy a része billentyűzetről is bevihető.Az = + < > | jelekből a TEX = + <> | jeleket készít. Nagyobb részük azonban olyan, amit csak parancsokkal tudunk előállítani. Talán a legfontosabb ilyen jel a szorzásjel · , amelyet a `$\cdot$` paranccsal érhetünk el. Például, a $n! = 1 \cdot 2 \cdots n$. kifejezéshez ezt kell megadnunk: `$n!=1\cdot2\cdots n$`.

A TEX rendelkezik az összes szimbólummal, jellel, amely matematikai szövegek nyomtatásában általánosan használatos. Az egyik legfontosabb csoportot a görög kis- és nagy-, valamint a kaligrafikus betűk alkotják:

Görög kisbetűk:

\alpha	α	\beta	β	\gamma	γ	\delta	δ
\epsilon	ϵ	\varepsilon	ε	\zeta	ζ	\eta	η
\theta	θ	\vartheta	ϑ	\iota	ι	\kappa	κ
\lambda	λ	\mu	μ	\nu	ν	\xi	ξ
\pi	π	\varpi	ϖ	\rho	ρ	\varrho	ϱ
\sigma	σ	\varsigma	ς	\tau	τ	\upsilon	υ
\phi	ϕ	\varphi	φ	\chi	χ	\psi	ψ
\omega	ω						

Görög nagybetűk:

\Gamma	Γ	\Delta	Δ	\Theta	Θ	\Lambda	Λ
\Xi	Ξ	\Pi	Π	\Sigma	Σ	\Upsilon	Υ
\Phi	Φ	\Psi	Ψ	\Omega	Ω		

Kalligrafikus nagybetűk:

\cal A	$\mathcal{A}$	\cal B	$\mathcal{B}$
\cal C	$\mathcal{C}$	\cal D	$\mathcal{D}$
\cal E	$\mathcal{E}$	\cal F	$\mathcal{F}$
\cal G	$\mathcal{G}$	\cal H	$\mathcal{H}$
\cal I	$\mathcal{I}$	\cal J	$\mathcal{J}$
\cal K	$\mathcal{K}$	\cal L	$\mathcal{L}$
\cal M	$\mathcal{M}$	\cal N	$\mathcal{N}$
\cal O	$\mathcal{O}$	\cal P	$\mathcal{P}$
\cal Q	$\mathcal{Q}$	\cal R	$\mathcal{R}$
\cal S	$\mathcal{S}$	\cal T	$\mathcal{T}$
\cal U	$\mathcal{U}$	\cal V	$\mathcal{V}$
\cal W	$\mathcal{W}$	\cal X	$\mathcal{X}$
\cal Y	$\mathcal{Y}$	\cal Z	$\mathcal{Z}$

A következő fontos csoport a műveleti jelekből, operátorokból áll. Először a bináris, 'kétműveletes' jeleket adjuk meg, majd a 'nagy' operátorokat:

\pm	$\pm$	\cap	$\cap$	\vee, \lor	$\vee$
\mp	$\mp$	\cup	$\cup$	\wedge, \land	$\wedge$
\setminus	$\setminus$	\uplus	$\uplus$	\oplus	$\oplus$
\cdot	$\cdot$	\sqcap	$\sqcap$	\ominus	$\ominus$
\times	$\times$	\sqcup	$\sqcup$	\otimes	$\otimes$
\ast	$\ast$	\triangleleft	$\triangleleft$	\oslash	$\oslash$
\star	$\star$	\triangleright	$\triangleright$	\odot	$\odot$
\diamond	$\diamond$	\wr	$\wr$	\dagger	$\dagger$
\circ	$\circ$	\bigcirc	$\bigcirc$	\ddagger	$\ddagger$
\bullet	$\bullet$	\bigtriangleup	$\bigtriangleup$	\amalg	$\amalg$
\div	$\div$	\bigtriangledown	$\bigtriangledown$		

Operátorok:

\sum	$\sum$	\bigcap	$\bigcap$	\bigodot	$\bigodot$
\prod	$\prod$	\bigcup	$\bigcup$	\bigotimes	$\bigotimes$
\coprod	$\coprod$	\bigsqcup	$\bigsqcup$	\bigoplus	$\bigoplus$
\bigvee	$\bigvee$	\biguplus	$\biguplus$	\biwedge	$\bigwedge$

Egyéb matematikai szimbólumok:

\aleph	$\aleph$	\prime	$\prime$	\forall	$\forall$
\hbar	$\hbar$	\emtyset	$\emptyset$	\exists	$\exists$
\imath	$\imath$	\nabla	∇	\neg, \lnot	$\neg$
\jmath	$\jmath$	\surd	$\surd$	\flat	$\flat$
\ell	ℓ	\top	$\top$	\natural	$\natural$
\wp	$\wp$	\bot	$\bot$	\sharp	$\sharp$
\Re	$\Re$	\\|, \Vert	$\Vert$	\clubsuit	$\clubsuit$
\Im	$\Im$	\angle	$\angle$	\diamondsuit	$\diamondsuit$
\partial	∂	\triangle	$\triangle$	\heartsuit	$\heartsuit$
\infty	∞	\basklash	$\backslash$	\spedesiut	$\spadesuit$
\smallint	$\smallint$				

Az alábbi jelek nagyrésze elérhető „áthúzott" alakban is, vagyis olyan alakú jelek is vannak amelyekkel az előbbi relációkat tagadni lehet. Pél-

Nyilak, következtetési jelek:

\leftarrow, \gets	$\leftarrow$	\longleftarrow	$\longleftarrow$
\Leftarrow	$\Leftarrow$	\Longleftarrow	$\Longleftarrow$
\rightarrow, \to	$\rightarrow$	\longrightarrow	$\longrightarrow$
\Rightarrow	$\Rightarrow$	\Longrightarrow	$\Longrightarrow$
\leftrightarrow	$\leftrightarrow$	\longleftrightarrow	$\longleftrightarrow$
\Leftrightarrow	$\Leftrightarrow$	\Longleftrightarrow	$\Longleftrightarrow$
\uparrow	$\uparrow$	\Uparrow	$\Uparrow$
\downarrow	$\downarrow$	\Downarrow	$\Downarrow$
\updownarrow	$\updownarrow$	\Updownarrow	$\Updownarrow$
\nearrow	$\nearrow$	\searrow	$\searrow$
\swarrow	$\swarrow$	\nwarrow	$\nwarrow$
\mapsto	$\mapsto$	\longmapsto	$\longmapsto$
\hookleftarrow	$\hookleftarrow$	\hookrightarrow	$\hookrightarrow$
\leftharpoonup	$\leftharpoonup$	\leftharpoondown	$\leftharpoondown$
\rightharpoonup	$\rightharpoonup$	\rightharpoondown	$\rightharpoondown$
\rightleftharpoons	$\rightleftharpoons$		

Relációs jelek:

\leq, \le	$\leq$	\geq, \ge	$\geq$	\equiv	$\equiv$
\prec	$\prec$	\succ	$\succ$	\sim	$\sim$
\preceq	$\preceq$	\succeq	$\succeq$	\simeq	$\simeq$
\ll	$\ll$	\gg	$\gg$	\asymp	$\asymp$
\subset	$\subset$	\supset	$\supset$	\approx	$\approx$
\subseteq	$\subseteq$	\supseteq	$\supseteq$	\cong	$\cong$
\sqsubseteq	$\sqsubseteq$	\sqsupseteq	$\sqsupseteq$	\bowtie	$\bowtie$
\in	$\in$	\ni, \owns	$\ni$	\propto	$\propto$
\vdash	$\vdash$	\dashv	$\dashv$	\models	$\models$
\smile	$\smile$	\mid	$\mid$	\doteq	$\doteq$
\frown	$\frown$	\parallel	$\parallel$	\perp	$\perp$

dául:\neq $\neq$,\notin $\notin$. Bármelyi relációs jelet áthúzhatjuk, ha a \not parancsot tesszük elé: \not\prec $\not\prec$, \not\approx $\not\approx$.

Néhány példa a fenti szimbólumok használatára:

```
$\alpha (x,y,z)-|\beta(x)|\leq|223x|<+\infty$
```

$$\alpha(x,y,z) - |\beta(x)| \leq |223x| < +\infty$$

```
$x\in A, y\not\in B\Rightarrow\forall z\in C,
\exists u: u\in(a\uplus B)x(C\setminus\{z\})$
```

$$x \in A, y \notin B \Rightarrow \forall z \in C, \exists u : u \in (a \uplus B)x(C \setminus \{z\})$$

```
$\vert f+g \vert \leq\vert f\vert +\vert g\vert$
```

$$|f + g| \leq |f| + |g|$$

```
$d(xdy)=dx\wedge dy\neq 0$
```

$$d(xdy) = dx \wedge dy \neq 0$$

A fejezet elején említettük, hogy a matematikai mód a TEX-en belül külön világot alkot. A matematikai mód azonban két külön világra oszlik. Eddig csak az egyik módról, a „szövegközi" matematikai kifejezésekről volt szó. A másik lehetőség matematikai kifejezések szedésére, hogy nem a szöveg soraiban, hanem kiemelve helyezzük el őket. Ezt a másik matematikai módot kettős dollárjellel kapcsolhatjuk be és ki. A dollárjelek között azonban ugyanazok a szabályok érvényesek, mint amelyeket eddig elmondtunk a matematikai módról.

Néhány példa a kiemelt matematikai módra: Ha azt gépeljük be, hogy `$\pi=3.1416\dots$`, akkor ez jelenik meg: $\pi = 3.1416\ldots$.
Ha viszont ezt gépeljük be `$$\pi=3.1416\dots$$`, akkor ezt így szedi a TEX:

$$\pi = 3.1416\ldots.$$

Láthatjuk, hogy sem új sor kezdésére, sem középre helyezésre, sem függőleges helyek kiadására nem kellett parancsot adnunk, ezt a TEX automatikusan megtette.

Az egyik leggyakoribb szépséghiba a központozási jelek elhelyezésével kapcsolatos. Ha szöveg közben használunk matematikai kifejezéseket, a matematikai módot sűrűn ki-be kell kapcsolgatni. Nézzük például a következő mondatot:

Tegyük fel, hogy f kétszer differenciálható az $[a,b]$ intervallumon, $f(a) < 0$, $f(b) > 0$, továbbá $g(x) \geq \delta > 0$ és $0 \leq h(x) \leq M$ bármely $x \in [a,b]$ esetén. Ehhez a mondathoz a következőket kell begépelni:

```
Tegyük fel, hogy $f$ kétszer differenciálható
az $[a,b]$ intervallumon, $f(a)<0$, $f(b)Ú0$, továbbá
$g(x)\geq \delta Ú0$ és $0\leq h(x)\leq M$ bármely
$x\in [a,b]$ esetén.
```

Bár az $f(a) < 0$, $f(b) > 0$ matematikai kefejezések között csak egyetlen vessző áll, mégis vissza kell kapcsolnunk szövegmódba. Ennek az az oka, hogy a vesszőket és más központozási jeleket matematikai módban a TEX máshogy helyezi el. Például, ha azt írjuk, hogy 'x,y,' akkor a ',' és az 'y' között kevesebb hely marad, mint matematikai módban az $f(x,y)$ kifejezésnél.

Általános szabályként megfogalmazhatjuk, hogy a szöveg központozási jeleit mindig a matematikai módon kívül kell elhelyezni. Egyetlen kivétel az, ha kiemelt matematikai kifejezést a mondat utolsó elemeként használunk. Például:

```
...ezzel beláttuk, hogy
$$\gamma =128.$$
Lássuk a következő állítást, amely ...
```

...ezzel beláttuk, hogy

$$\gamma = 128.$$

Lássuk a következő állítást, amely ...

Ahhoz, hogy ezt a helyes központozást kapjuk, a mondatot befejező pontot a $$ jelenken „belülre" kell tennünk, ugyanis ha azt írjuk, hogy

```
... ezzel beláttuk,
hogy $$\gamma =128 $$. Lássuk a következő állítást,
amely ...
```

akkor ezt kapjuk:

...ezzel beláttuk, hogy

$$\gamma = 128$$

. Lássuk a következő állítást, amely ...

Természetesen ha sokszor kell valamit ki-be kapcsolgatni, előbb-utóbb elfelejtjük kinyitni vagy becsukni. Tekintve, hogy ez — különösen egy hosszabb szöveg esetén — szinte bizonyosan előfordul, a TEX, amennyire lehet, felkészül rá. Ha olyan parancsot talál matematikai módban, amely csak szövegmódban fordul elő, hibajelzést ad, és a parancs elé beír egy $

jelet. Ez bizonyosan nem ront a helyzeten, de lehet, hogy a matematikai módot már sokkal előbb kellett volna kikapcsolni. Természetesen ennek a fordítottja is előfordulhat. A TEX ekkor is egy $ jelet tesz a szövegbe, de most azért, hogy bekapcsolja a matematikai módot.

Matematikai szövegben — mivel semmi értelme — nem lehet üres sor. Ezt a TEX új bekezdésre való parancsként fogja fel, amely matematikai kifejezéseknél nem alkalmazható.

5.1. Indexek, kitevők

Matematikai szövegben gyakran előfordul, hogy betűket, számokat vagy egész kifejezéseket egy betű kitevőjébe vagy indexébe kell tenni. Az index létrehozására a _ jel szolgál:

```
$x_2$      x_2
$a+b_c$    a+b_c
```

A TEX az indexet mindig azután a karakter után teszi, amely a _ jel előtt áll[1]. Indexbe pedig csak az a karakter kerül, amelyik a _ jel után következik. Ha azt akarjuk, hogy az indexben több karakter legyen, környezet kijelölő szimbólumokat kell használnunk:

```
$x_{2a}$                x_{2a}
$a+b_{f(x,y,z)}$        a+b_{f(x,y,z)}
```

de ugyanakkor

```
$x_2a$                  x_2a
$a+b_f(x,y,z)$          a+b_f(x,y,z) .
```

Kitevők létrehozásához a ^ jelet használjuk:

```
$x^2$      x^2
$a+b^c$    a+b^c .
```

A TEX a kitevőt a ^ jel előtt álló karakterhez illeszti. A környezet kijelölő jeleket most is használhatjuk, ha a kitevő több betűből, jelből áll:

[1] Feltéve, hogy az nem egy másik index ill. kitevő. Lásd később.

```
$x^{2a}$          x^{2a}
$a+b^{f(x,y,z)}$  a+b^{f(x,y,z)}
$a+b^f(x,y,z)$    a+b^f(x,y,z).
```

Abban az esetben, ha egy kifejezéshez akarunk indexet vagy kitevőt tenni, a kifejezést kell környezetnek kijelölnünk. Ekkor nem a kifejezés utolsó karakteréhez, hanem a kifejezés egészéhez igazítja az idexet ill. kitevőt. A különbség sokszor szemmel észre sem vehető, de nagy, bonyolult kifejezéseknél fontos lehet. Másként szedi például a TEX, ha azt adjuk meg, hogy `$f(x)_+$` amelyből $f(x)_+$-t készít; mintha azt adtuk volna meg, hogy `${f(x)}_+$`, amelyből $f(x)_+$ lesz. Hasonlóan:

```
${(x+1)}^{2n}$  (x+1)^{2n}
$(x+1)^{2n}$    (x+1)^{2n}.
```

Az indexes, kitevős kifejezéseket egymásba is ágyazhatjuk tetszőleges mélységben:

```
$ Q_{i_k}$      Q_{i_k}
$2^{2^n}$       2^{2^n}
$a^{b^{(c+1)}}$ a^{b^{(c+1)}}.
```

Ha indexet és kitevőt együtt alkalmazunk ugyanarra a betűre (jelre), akkor mindegy, hogy milyen sorrendben állnak:

```
$a_0^n$  a_0^n
$a^n_0$  a^n_0.
```

A TEX a következőképpen értelmezi az ilyen „vegyes" kifejezéseket:

- Ha talál egy _ vagy ^ jelet, először meghatározza, mi kerüljön az indexbe, ill. a kitevőbe. Ha a jel után nem áll kapcsos zárójel, a kitevőbe a következő — nem szóköz vagy sorvége — karaktert teszi. Ha az index ill. kitevő jel után kapcsos zárójel áll, akkor a környezetben levő egész kifejezés az indexbe ill. kitevőbe kerül.
- Ezután meghatározza, hogy mi legyen az idex ill. kitevő alapja: a _ ill. ^ jel előtt álló karaktereket vizsgálja visszafelé egészen addig, míg egy olyan karaktert talál, amely nem tartozik sem az indexben lévő, sem a kitevőben lévő kifejezéshez.

  Kövessük végig a fenti algoritmus a következő példákon:

`$a_0^n$`	a_0^n
`$q_{ij}^*$`	q_{ij}^*
`$a_{ij}^{2^n}$`	$a_{ij}^{2^n}$
`$X_{a_1^+}^{b_k^-}$`	$X_{a_1^+}^{b_k^-}$
`$2^{a_n^{c_i^*}}$`	$2^{a_n^{c_i^*}}$.

Ha a fenti értelmezést a TEX nem tudja végrehajtani, hibaüzenetet kapunk. Például, ha létre akarjuk hozni a $2^n 3^k$ kifejezést, akkor ezt gépeljük be: `$2^n3^k$`. Ha véletlenül lehagyjuk a 3-at, akkor a `$2^n^k$`-ra a TEX hibát jelez. Két külön kitevőt kellene ugyanis a 2-re alkalmaznia. Hasonlóan hibához vezet az `$a_b_c$` kifejezés is.

Lehetnek természetesen olyan hibák is, amelyeket a TEX nem vesz észre. Egy hibásan megadott kifejezésnél nem az lesz a kitevő vagy az index alapja, amit szeretnénk. Ha azt akarjuk írni a_i^{2n}, és ehhez az `$a^2n_i$` karaktereket gépeljük be, akkor hibajelzést nem kapunk, de a megjelenő formula ez lesz: $a^2 n_i$. Mivel a 2n-et nem zártuk kapcsos zárójelek közé, az a kitevője csak a 2-es szám lett és a TEX úgy értelmezte, hogy az i alapbetűje az n és nem a 2. Helyesen így lett volna: `$a^{2n}_i$`.

A _ és ^ jelek helyett használhatunk parancsszavakat is. Az indexre a `\sb` , a kitevőkre a `\Sp` parancsszót.[2] Hatásuk teljesen megegyezik a fentebb ismertetett jelekével, de ne felejtsük el, hogy — mivel ezek parancsszavak — utánuk egy szóközt kell hagyni. Például: `$x\sp 2$` kiszedve x^2. Láttuk már a példák során, hogy index és kitevő nemcsak betű vagy szám, hanem tetszőleges (esetleg paranccsal előállított) jel is lehet. Ez érvényes az indexek és kitevők alapjára is, például:

`$\beta^2_i$`	β_i^2
`$\Re_\infty ^{\partial x\partial y}$`	$\Re_\infty^{\partial x \partial y}$
`$f_+^\prime +f_-^{\prime \prime}$`	$f_+' + f_-''$
`${}_2C_6^+$`	${}_2C_6^+$.

Az utolsó példában az üres környezetet használtuk fel arra, hogy indexet, az alapbetű elé tegyünk. Bizonyos kifejezéseknél, kémiai, biológiai

[2] 'subscript' (=aláírás), 'superscript'(=föléírás) angol szavakból

képleteknél szükség lehet arra, hogy a kitevő is az alapkarakter előtt álljon. Ezt hasonló módon a {} üres környezettel érhetjük el.

```
${}^{\prime}f(x)$                ${}'f(x)$
${}_{y_i^{*\prime }}\Gamma^2$    ${}_{y_i^{*\prime}}\Gamma^2$.
```

5.2. Emeletes kifejezések

Ebben a szakaszban foglalkozunk olyan összetett kifejezések szedésével, mint például a törtek és a binomiális együtthatók.

A törteket kétféleképpen szokás szedni. Az egyik esetben a tört a szöveg közé kerül, ekkor a számlálót / jel választja el a nevezőtől: (n+1)/2, a/b.

A másik esetben a törtet kiemelik a szövegből, így:

$$\frac{a+b+c}{x+y+z}.$$

Az előbbi sokkal egyszerűbb; TEX parancs nélkül is azonnal tudunk egyszerű törteket készíteni:

```
$a/b$                          $a/b$
$(a+b+c)^2/(x^2_0-y^2_0)$      $(a+b+c)^2/(x^2_0-y^2_0)$.
```

Bonyolultabb törtkifejezések azonban követhetetlenné válnak, ha ilyen módon írjuk őket. A vízszintes-törtvonalas kifejezéseket az \over [3] parancsszóval készíthetünk. Az előbbi példához ezt gépeltük be:

```
$$a+b+c\over x+y+z$$
```

Az \over parancs kiadása előtt álló összes karakter a tört számlálójába, az utána álló összes jel a tört nevezőjébe kerül. Itt az „összes" alatt csak egy adott környezeten belüli összes karaktert értjük. Ha a tört mellett más is szerepel a kifejezésben, a törtet kapcsos zárójelekkel kell elkülönítenünk:

```
$(1+{1\over x})^n$
```

$$(1+\frac{1}{x})^n.$$

[3] over=fölött angol szó

Itt a kapcsos zárójelekkel kijelölt környezeten belül csak az 1 áll az \over előtt, és csak az x van a parancsszó után. Ha a kapcsos zárójeleket lehagyjuk, ezt kapjuk:

$$\frac{(1+1}{x)^n}.$$

További példák törtekre:

`${a^2-b^2\over a-b}=a+b$`	$\frac{a^2-b^2}{a-b}=a+b$
`$\Gamma_{a\over 2}^n$`	$\Gamma_{\frac{a}{2}}^n$
`$a+b^{c^2\over x-1}-1$`	$a+b^{\frac{c^2}{x-1}}-1$
`$1+{1\over {1+1\over x}}$`	$1+\frac{1}{\frac{1+1}{x}}$.

Egyszerűbb törteket is tartalmazó kifejezéseknél általában célszerűbb mindkét törtjelölést alkalmazni. Az utolsó példát írhattuk volna így: $1+\frac{1}{1+1/x}$. Jobban olvasható, ha azt írjuk:

$$\frac{a/b}{2}.$$

mintha azt írnánk

$$\frac{\frac{a}{b}}{2}.$$

Bizonyos kifejezésekhez

$$\begin{matrix} a \\ x+y \end{matrix}$$

alakú formulára van szükség. Erre az \atop parancsszó szolgál. Használata megegyezik az \over parancs használatával, az eredményként előálló formula is ugyanaz, kivéve, hogy hiányzik a törtvonal.

`$\beta \atop \gamma$`	$\begin{matrix}\beta\\ \gamma\end{matrix}$
`$a_0 \atop c^*_1+b^*_1$`	$\begin{matrix}a_0\\ c_1^*+b_1^*\end{matrix}$.

Ilyen kifejezés a binomiális együttható is, azzal a különbséggel, hogy a kifejezést zárójelek közé teszi:

$$\binom{n}{k}$$

A binomiális együtthatókat előállíthatnánk az \atop paranccsal is, de a TEX-ben a \choose parancs kifejezetten ezt a célt szolgálja:

```
$n\choose k$
```
$\binom{n}{k}$

```
$n+k\choose k-1$
```
$\binom{n+k}{k-1}$.

Matematikai kifejezéseket (törteket, kitevőket, indexeket, binomiális együtthatókat stb.) tetszőleges mélységben — vegyesen is — egymásba ágyazhatunk, de mindig gondosan ügyelnünk kell a környezet kijelölésére. Hibához vezet, ha például a \choose és \over parancsokat együtt használjuk: $n\choose k \over 2$, de a megfelelő környezetkijelöléssel helyes kifejezéshez jutunk:

```
${n\choose k}\over 2$
```
$$\frac{\binom{n}{k}}{2}$$

```
$n\choose{k\over 2}$
```
$\binom{n}{\frac{k}{2}}$.

Az \atop parancsra épül a \brack és a \brace parancs. Ezek abban különböznek a \choose -tól, hogy nem gömbölyű, hanem szögletes ill. kapcsos zárójelek közé teszik az \atop paranccsal felépített kifejezést.

```
$m\brack n$
```
$\left[\begin{matrix} m \\ n \end{matrix}\right]$

```
$m+n+1\brace \alpha +\beta +\gamma $
```
$\left\{\begin{matrix} m+n+1 \\ \alpha+\beta+\gamma \end{matrix}\right\}$.

5.3. Szumma és integráljelek

Az összegzést jelölő görög $\sum$ (nagy szigma) gyakran szerepel matematikai kifejezésekben. Előállíthatjuk, ha $\Sigma $ karakterek begépelésével is, de ez a szigma betű túlságosan kicsi lesz, és az összegzés határait csak aprólékos munkával tudjuk alá, ill. fölé írni. Ha például arra van szükségünk, hogy

$$\sum_{i=1}^{n} a_i \,,$$

akkor ehhez a következőt gépeljük be:

`$$\sum_{i=1}^n a_i$$`.

A $\sum$ jel alá a _ karakter után álló karakter (környezet esetén karakterek) kerülnek, fölé a ^ után állók, és az utána következők állnak majd a $\sum$ jel „hasában". Könnyíti a `\sum` parancs használatának megjegyzését, hogy az összegzés határai formailag mint indexet és kitevőt kell megadnunk. Valójában ez nem index és kitevő, hiszen `$$\Sigma_{i=1}^n a_i$$` begépelésével mást kapunk:

$$\Sigma_{i=1}^n a_i \,.$$

Bizonyos estekben ezt is lehet használni, de a szumma jel tulságosan kicsi ahhoz, hogy nagyobb formulákat képes legyen összefogni.

Az alsó és felső korlátokat teszőleges sorrendben megadhatjuk, tehát: `$$\sum_{i=1}^n a_i$$` és a `$$\sum ^n_{i=1}a_i$$` ugyanazt eredményezi. A `\sum` parancsszót és az ahhoz tartozó alsó és felső korlátokat ugyanúgy értelmezi a TEX mint a betűk, kifejezések esetén az indexeket és a kitevőket, ezért hasonló hibákat, ill. hibajelzéseket kapunk ha valamelyik betűt, környezet kijelölő zárójelet stb. lehagyjuk.

Eddig a szumma jelet csak kiemelt matematikai kifejezéseknél használtuk. Szövegközben az összegzési határokat nem alá, hanem mellé teszi a TEX, hogy a kifejezés jobban elférjen a sorban. Például: `$\sum_{i=1}^n a_i$` eredménye képpen $\sum_{i=1}^n a_i$-t kapunk.

Előfordulnak olyan összegzések is, ahol az összegzési feltételeket a $\sum$ jel alatt több sorban lehet csak megjeleníteni.

$$\sum_{\substack{0<i<n \\ 0<\delta<m}} c(i,j).$$

Ekkor a $0 < i < n$ és $0 < \delta < m$ kifejezésekből az `\atop` parancs segítségével egy „emeletes" kifejezést készítünk, és ez áll a $\sum$ jel alatt: `$$\sum_{0<i<n \atop 0<j<m} c(i,j)$$`.

Integráljeleket a következő parancsokkal állíthatunk elő:

`\int` $\int$ `\oint` $\oint$.

Például: `$\int_a^b$`. Az integrálás alsó, ill. felső határát most is a _, ill. a ^ jelekkel adhatjuk meg. Ha kiemelt kifejezésben használjuk: `$$\int_0^{+\infty}$$`

$$\int_0^{+\infty}$$

A TEX még a kiemelt kifejezéseknél is az integrálási határokat az integráljel mellé és nem alá írja. Ha a képlet úgy kívánja, külön paranccsal elérhetjük azt is, hogy ezek az integráljel alá ill. fölé kerüljenek:

$$\int_0^{+\infty} .$$

A TEX-ben két külön parancsszó használható arra, hogy a $\sum$, ill. a $\int$, jeleknél a TEX konvencióit fölülbíráljuk, és az összegzési, ill. integrálási határokat a jelek mellé, ill. fölé, alá tegyük. Ezek a parancsok a `\limits` és a `\nolimits`. A `\limits` parancs hatására az összegzési vagy integrálási határok a szumma, ill. integráljel alá, fölé kerülnek, függetlenül attól, hogy szövegközbeni vagy kiemelt kifejezésről van szó.

```
$\sum\limits_{j=k+1}^{+\infty}$
```
$\sum\limits_{j=k+1}^{+\infty}$

```
$\int\limits_{-\pi/2}^{+\pi/2}$
```
$\int\limits_{-\pi/2}^{+\pi/2}$.

Kiemelt kifejezés esetén: `$$\int\limits_{-\pi/2}^{+\pi/2}$$`

$$\int\limits_{-\pi/2}^{+\pi/2}$$

A `\nolimits` parancs hatása a `\limit` parancséval ellentétes: a TEX mindig a megfelelő jel mellé teszi a korlátokat, például: `$$\sum \nolimits _ {i=1}^ {+\infty} {1\over x^2}$$`

$$\sum\nolimits_{i=1}^{+\infty} \frac{1}{x^2} .$$

Az integrált tartalmazó kifejezések utolsó eleme általában azokat a változókat tartalmazza, amelyek szerint integrálunk. Ha azt írjuk: `$$\int_0^1 f(x,y)dxdy$$`

$$\int_0^1 f(x,y)dxdy ,$$

nem alkalmazkodunk ahhoz a konvencióhoz, miszerint ezt az utolsó *dxdy* tagot egy kicsit elkülönítve, távolabb kell írni az integrált függvénytől. Ezt a TEX sajnos nem képes automatikusan megtenni, hanem egy külön \, parancsjellel kell utasítanunk. A formula helyesen: `$$\int_0^1f(x,y)\, dx\, dy$$`

$$\int_0^1 f(x,y)\, dx\, dy\,.$$

A \, parancsjelet természetesen más kifejezésekben is használjuk, ahol egy kis helyet kell hagyni a betűk, szimbólumok között, de *csak matematikai módon belül.* A \, jelhez hasonlóan a betűk, jelek közötti helyhagyást lehet szabályozni a \! paranccsal. Ennek hatására azonban nem távolabb kerülnek a betűk, jelek, hanem közelebb: a parancs egy kismértékű „negatív" helyet eredményez. Erre van szükségünk például akkor, amikor többszörös integrálokat akarunk előállítani. Az integráljelek közelebbhúzás nélkül így néznek ki:

$$\int\int\int f(x,y,z)dx\,dy\,dz$$

Sokkal jobb, ha az integráljelek közelebb kerülnek egymáshoz:

$$\iiint f(x,y,z)dx\,dy\,dz$$

Mivel a \! csak nagyon kicsit húzza közelebb a jeleket betűket, az előbbi példában háromszor is alkalmaztuk két integráljel között:

```
$$ \int \!\!\!\int \!\!\!\int f(x,y,z)dx\,dy\,dz$$
```

Két másik utasítás szolgál még a szóközök nagyságának vezérlésére, a \; és a \>. Ezek ugyanúgy mint a \, parancsjel, bizonyos mértékű szóközt eredményeznek. Ha több, faktoriálisból álló kifejezést írunk, jobban olvasható lesz, ha közöttük egy kis helyet hagyunk. Például igy néz ki egy kifejezés, ha a faktoriálisok között nem hagyunk „extra" helyet:

$$11!22!33!44!\,,$$

ha a \, parancsjelet alkalmazzuk:

$$11!\,22!\,33!\,44!\,,$$

ha a \; parancsjelet alkalmazzuk:

$$11!\,22!\,33!\,44!\,,$$

és ha a \> parancsjelet alkalmazzuk:

$$11!\,22!\,33!\,44!\,.$$

Ebben az esetben nincs értelme a \! parancsjelet használni, hiszen akkor ezt kapnánk:11!22!33!44!. Mind a négy parancsjel csak matematikai módban használható. Ennek oka az, hogy a ilyen szóközhagyó parancsokra elsősorban a matematikai módban van szükség, hiszen ekkor a TEX minden szóközt lenyel.

Van azonban két olyan parancsszó, amely nem csak matematikai módban, hanem szövegmódban is használható: a \quad és a \qquad. Ezek sokkal nagyobb szóközt helyeznek el a betűk között, mint az előző parancsok bármelyike. A \quad esetében ez: ilyen nagy, de a \qquad esetében ez: még nagyobb. A egyes szóköznagyságok a \quad-hoz viszonyítva:

\,	a \quad nagyságának 1/6-oda
\>	a \quad nagyságának 2/9-ede
\;	a \quad nagyságának 5/18-ada
\!	a \quad nagyságának −1/6-oda
\qquad	a \quad nagyságának 2-szerese.

Ha a szumma vagy az integráljel nagyobb kifejezés részeként szerepel, kapcsos zárójelek közé kell tennünk, mert egyébként minden, ami a \sum, ill. a \int parancs után áll, azok „hasába" kerül:

```
$1+{\sum_{i=0}^{10} i^n}+n^2$
```

$$1+\sum_{i=0}^{10} i^n+n^2\,.$$

5.4. Változó nagyságú matematikai jelek

Szövegek aláhúzására szövegmódban az \underbar parancsot használtuk. Matematikai módban az \underline[4]: parancs segítségével tehetjük ezt

[4] underline(=aláhúzás) angol szóból

meg. Alkalmazásának szabályai megegyeznek azzal, amit az \underbarról mondtunk; csak a parancsután következő betűt (vagy a környezetettel kijelölt elemeket) húzza alá:

```
$\underline xy$                    $\underline xy$
$\underline {xy}$                  $\underline{xy}$
$\underline x+\underline y
\geq\underline{x+y}$               $\underline{x}+\underline{y}\geq\underline{x+y}$ .
```

Föléhúzást az \overline[5] paranccsal készíthetünk:

```
$\overline {A\land B}=
\overline A\lor \overline B$       $\overline{A\land B}=\overline{A}\lor\overline{B}$ .
```

Vektorok esetén inkább egy balra vagy jobbra mutató nyilat húznak a betűk, jelek fölé. Ezeket az \overleftarrow : 'balra mutató nyíl föléhúzása'; ill. a \overrightarrow : 'jobbra mutató nyíl föléhúzása' parancsokkal készíthetjük el.

```
$\overrightarrow{x+y}
=-\overleftarrow {x+y}$            $\overrightarrow{x+y}=-\overleftarrow{x+y}$ .
```

A TEX, mint ahogy az előző példából is láttuk, mindig a kifejezésnek megfelelő hosszúságú vonalat húz alá, ill. fölé. A törteknél a törtvonal hosszát is automatikusan állapította meg. Más jeleknél, mint például a négyzetgyökjelnél a jel nagyságát az alatta lévő kifejezés méretéből számítja ki.

Négyzetgyökjelet az \sqrt paranccsal állíthatunk elő:

```
$\sqrt 2$                          $\sqrt2$
$\sqrt {a^2+n^2}$                  $\sqrt{a^2+n^2}$
$\sqrt {x\over a+b}+1$             $\sqrt{\frac{x}{a+b}}+1$
$\sqrt {a^0+a^1+a^2+\ldots +a^n}$  $\sqrt{a^0+a^1+a^2+\ldots+a^n}$ .
```

A TEX képes tetszőlegesen nagy négyzetgyökjelet építeni. Egy bizonyos méret fölött ugyanis a '–' és a '|' elemek összeállításával nyújtja kellő nagyságúra a gyökjelet:

[5] overline= föléhúzás angol szóból

```
$$\sqrt 1+{1+\sqrt {1+\sqrt
{1+\sqrt {1+\sqrt {1+\sqrt {1+x}}}}}}$$
```

$$\sqrt{1}+1+\sqrt{1+\sqrt{1+\sqrt{1+\sqrt{1+\sqrt{1+x}}}}}$$

Ha nem négyzetgyök, hanem köb- ill. magasabb rendű gyökjeles kifejezést készítünk, a `\root...\of` parancsszavakat használjuk:[6]

`$\root 3\of 2$`	$\sqrt[3]{2}$
`$\root n\of {x^n+y^n}$`	$\sqrt[n]{x^n+y^n}$
`$\root {n+1}\of {x^2+{a+b\over y^2}}$`	$\sqrt[n+1]{x^2+\frac{a+b}{y^2}}$.

A leggyakrabban szereplő változó nagyságú jelek a zárójelek. A különböző zárójeleknek mindig „körül kell ölelnie" a benne lévő kifejezést, ezért fontos, hogy a zárójeleket minden méretben ki lehessen szedni. A TEX képes tetszőlegesen nagy zárójeleket szedni, ezt ugyanazzal a módszerrel éri el, mint amit a négyzetgyökjelnél már elmondtunk.

$$\left(\left(\left(

(	(	\lbrack	[	\lbrace	{
)	)	\rbrack	]	\rbrace	}
\lfloor	⌊	\lceil	⌈	\langle	⟨
\rfloor	⌋	\rceil	⌉	\rangle	⟩
\vert	|	\Vert	‖	\backslash	\ .

$\uparrow $	↑
$\Uparrow $	⇑
$\downarrow $	↓
$\Downarrow $	⇓
$\updownarrow $	↕
$\Updownarrow $	⇕
$/$	/
$\backslash $	\

A bemutatott zárójeleken kívül még 16 más zárójelet lehet a TEX-hel szedni. Az összes zárójelfajtát gyűjtöttük egybe a fenti táblázatban.

A zárójelekhez hasonlóan „hosszabbíthatók" a függőleges nyilak és az elhatároló jelek (pl. / per jel vagy a \)

A különböző nagyságú zárójeleket és függőleges nyilakat segédparancsokkal érhetjük el. Az első nagyobb méretet a \bigl és a \bigr parancsok alkalmazásával kapjuk úgy, hogy a megfelelő zárójel karaktert vagy parancsot közvetlenül a \bigl, ill. \bigr után írjuk. A \bigl a baloldalon álló nyitó zárójelekhez, a \bigr pedig a jobb oldalon álló záró zárójelekhez tartozik[7]. Azért van a nyitó és a záró zárójeleknek külön parancsa, mert a TEX — a matematikai koonvenciónak megfelelően — attól függően, hogy nyitó vagy záró zárójelet szed, jobbra ill. balra eltolja egy kicsit.

```
$\bigl(x-a(x)\bigr)
\bigl[y-a(y)\bigr]$          (x − a(x))[y − a(y)]
$\bigl||x|+|y|\bigr|<\bigl
\lfloor x+y\bigr\rfloor $    ||x| + |y|| < ⌊x + y⌋ .
```

[7] bigl= a „big left" nagy bal; a bigr= a „big right" nagy jobb angol szerkezetekből származik

A következő nagyságot a \Bigl és a \Bigr parancsokkal, az azt követő méretűt a \biggl és \biggr , majd a legnagyobb méretűt a \Biggl és \Biggr parancsok használatával kapjuk. Álljanak itt az összehasonlíthatóság kedvéért egymás után a különböző méretek: Eredeti méret:

$$()[]\{\}\lfloor\rfloor\lceil\rceil\langle\rangle/\backslash\|\uparrow\Uparrow\downarrow\Downarrow\updownarrow\Updownarrow,$$

amit a \big ... alkalmazásával kapunk:

$$\big(\big)\big[\big]\big\{\big\}\big\lfloor\big\rfloor\big\lceil\big\rceil\big\langle\big\rangle\big/\big\backslash\big|\big\|\big\uparrow\big\Uparrow\big\downarrow\big\Downarrow\big\updownarrow\big\Updownarrow.$$

A \Big... alkalmazásával az eredeti méret kétszeresét, az előző — \big... — méretek másfélszeresét kapjuk:

$$\Big(\Big)\Big[\Big]\Big\{\Big\}\Big\lfloor\Big\rfloor\Big\lceil\Big\rceil\Big\langle\Big\rangle\Big/\Big\backslash\Big|\Big\|\Big\uparrow\Big\Uparrow\Big\downarrow\Big\Downarrow\Big\updownarrow\Big\Updownarrow,$$

a \bigg... parancsokkal a \big... méretekhez képest kétszer akkora jeleket írhatunk:

$$\bigg(\bigg)\bigg[\bigg]\bigg\{\bigg\}\bigg\lfloor\bigg\rfloor\bigg\lceil\bigg\rceil\bigg\langle\bigg\rangle\bigg/\bigg\backslash\bigg|\bigg\|\bigg\uparrow\bigg\Uparrow\bigg\downarrow\bigg\Downarrow\bigg\updownarrow\bigg\Updownarrow,$$

végül a legnagyobb méretű jelek, amelyeket \Bigg ... paranccsal kaphatunk 2,5-szer akkorák, mint a \big... jelek:

$$\Bigg(\Bigg)\Bigg[\Bigg]\Bigg\{\Bigg\}\Bigg\lfloor\Bigg\rfloor\Bigg\lceil\Bigg\rceil\Bigg\langle\Bigg\rangle\Bigg/\Bigg\backslash.$$

Ezek a méretek az esetek túlnyomó többségében elegendőek. Ennél nagyobb jeleket csak bonyolultan készíthetünk, erre a könyvben nem térünk ki.

A fenti parancsok elsősorban párba állított zárójelekre alkalmazhatók, de az sem okoz hibát, ha egy ilyen parancs megfelelő pár nélkül áll:

```
$${a+1\over b}\biggl/{c+1\over d}$$
```

$$\frac{a+1}{b}\Big/\frac{c+1}{d}$$

A középen álló 'per' jelet szebben helyezi el a TEX, ha az iménti parancsoknak megfelelő, de nem a bal ill. a jobb, hanem a középen álló jelek

nagyobbítására szánt \bigm \Bigm \biggm \Biggm parancsokat használjuk.[8] Ekkor ugyanis a középső jel előtt szimmetrikusan két kis helyet hagy a TEX: **$${a+1\over b}\biggm/{c+1\over d}$$**

$$\frac{a+1}{b} \Big/ \frac{c+1}{d}$$

Még két példa a fenti parancsok használatára:

$\biggl(\sum_{i=1}^k x_i\biggr)^n$

$$\biggl(\sum_{i=1}^{k} x_i\biggr)^n$$

$a\Biggl[\biggl(x+{b\over 2a}\biggr)^2
\biggl({c\over a}-{b^2\over 4a}\biggr)\Biggr]$

$$a\Biggl[\biggl(x+\frac{b}{2a}\biggr)^2+\biggl(\frac{c}{a}-\frac{b^2}{4a}\biggr)\Biggr]$$

Mivel a zárójelek nagyságát mindig úgy kell megválasztanunk, hogy illeszkedjen a közöttük levő kifejezés méretéhez, pontosan kellene tudnunk, hogy egy tört, gyökjeles kifejezés, integrál vagy szumma jel mekkora lesz a szedett szövegben. Ennek megbecslése és a megfelelő méretű zárójel kiválasztása még nagy gyakorlat esetén is sok munkát igényel. Szerencsére a legtöbb esetben nem kell ezzel törődnünk, mert van két olyan parancs, amelyikkel megmondhatjuk a TEX-nek, hogy automatikusan, a szedett kifejezés nagyságához igazítsa a zárójelek, nyilak nagyságát. Ez a két parancs — ugyanúgy mint például a \big esetében — tulajdonképpen sok parancs. A \left...és \right... parancsszó után közvetlenül most is tetszőleges zárójel, függőleges nyíl állhat. Például:

$\left(\sqrt{{a-1\over b-1}+1}
\right)$

$$\left(\sqrt{\frac{a-1}{b-1}+1}\right)$$

$\left({n\choose k}\over{k\choose i}\right)$

$$\left(\frac{\binom{n}{k}}{\binom{k}{i}}\right).$$

[8] a bigm= a „big medium" nagy középső szavakból származik

Az utóbbi példa rávilágít a `\left` ... ill. `\right` ... parancsok egy másik fontos tulajdonságára: a törteknél elmondottak értelmében a kifejezést a TEX-nek így kellett volna kiszednie:

$$\frac{\left(\binom{n}{k}\right.}{\left.\binom{k}{i}\right)}.$$

A TEX azért szedte jól a képletet, mert ezek a zárójelek amellett, hogy a szedett szövegben megjelennek, még környezetkijelölő jelként is működnek. Ezért például a `\left(` parancs egyenértékű a megfelelő méretű gömbölyű zárójel és egy kezdő kapcsos zárójel bevitelével:

```
$\left(1+{1\over x}\right)^n$
```

$$\left(1+\frac{1}{x}\right)^n$$

```
$x^*+\left(\sum_{j=1}^m c_j^k\right)+1$
```

$$x^*+\left(\sum_{j=1}^{m} c_j^k\right)+1\,.$$

A TEX tudja, hogy a `\left` parancsszónál egy környezet kezdődik, és a `\right` parancsszónál végződik. Fontos, hogy ezek a parancsok — mint a környezetkijelölő jelek mindig — párban álljanak. Hibához vezet, ha egy kifejezésben egy `\left`, ill. `\right` parancsszóhoz nem tartozik pár. Néha előfordulhat, hogy bizonyos jeleket nem párban akarunk használni. Ilyen például a következő két kifejezés:

$$\left.\frac{dx^2}{dx}\right|_{x=1}=2$$

$$\frac{a+b}{c}\Big/\frac{x+y}{z}\,.$$

Ahhoz, hogy ezekben a formulákban is párosítani lehessen a `\left`, ill. `\right` parancsszavakat, létezik két olyan parancsszó, amelyből nem jelenik meg a szedett szövegben semmi, de a megfelelő környezetet „kinyitja", ill. „becsukja": `\left.` ill. `\right.`. Az előbbi két pédát így gépeltük be:

```
$$\left. {dx^2\over dx}\right|_{x=1}=2 $$
$${a+b\over c}\left/{x+y\over z}\right..$$
```

5.5. Matematikai szövegek szedési stílusai

A bemutatott matematikai kifejezéseknél láttuk, hogy a TEX automatikusan választja ki a betűméreteket. Ha például indexbe vagy kitevőbe kerül egy szám, betű vagy jel, akkor azt kisebbel szedi, a nagy képletekhez tartozó zárójeleket a megfelelő nagyságban szedi stb. Sokkal kevésbé észrevehető, hogy a különböző helyeken más és más elhelyezési konvenció, ún. stílus szerint dolgozik. Matematikai módon belül a következő négy stílus lehetséges:

- kiemelt kifejezések stílusa — szövegből kiemelt képleteket, formulákat szed ilyen stílusban
- szövegközi stílus — szövegben lévő kifejezések stílusa
- index stílus — indexeket, kitevőket (szumma ill. integráljelnél a korlátok megadásánál) szedi e szerint a stílus szerint
- index-index stílus — indexben vagy kitevőben lévő kifejezések indexeit, kitevőit szedi ebben a stílusban

A négy stílushoz négy féle betű, ill. jelméret tartozik:

kiemelt kifejezések stílusa $\displaystyle\sum_{i=1}^{n} \beta_a b^c$

szövegközi stílus $\sum_{i=1}^{n} \beta_a b^c$

index stílus $\sum_{i=1}^{n} \beta_a b^c$

index-index stílus $\sum_{i=1}^{n} \beta_a b^c$.

A TEX automatikusan választja ki a stílust és a betűnagyságot, de parancsszavakkal ezt felülbírálhatjuk. A betűk, jelek méretének és elhelyezésének konvencióit a következő parancsokkal válthatjuk át:

`\displaystyle`	„kiemelt" stílusra vált át
`\textstyle`	szövegközi stílusra vált át
`\scriptstyle`	index stílusra vált át
`\scriptscriptstyle`	index-index stílusra vált át.

Néhány esetben szükséges, hogy felülbíráljuk a TEX által kiválasztott stílust. Ha egy emeletes törtet szedünk TEX-hel, a betűk egyre kisebbek

lesznek, hiszen először normál méretű betűket használ, majd index méretűt, majd index-index méretűt. Az eddigi ismereteink alapján elkészíthetjük:

```
$$x_0+{1\over x_1+{1\over x_2+{1\over x_3+{1\over x_4}}}}$$
```

amely így jelenik meg:

$$x_0+\cfrac{1}{x_1+\cfrac{1}{x_2+\cfrac{1}{x_3+\cfrac{1}{x_4}}}}$$

Sokkal olvashatóbb, ha a törtben betűk mérete változatlan marad:

$$x_0+\cfrac{1}{x_1+\cfrac{1}{x_2+\cfrac{1}{x_3+\cfrac{1}{x_4}}}}$$

Ehhez következőket gépeltük be:

```
$$x_0+{1\over\displaystyle x_1+
{\strut 1\over\displaystyle x_2+
{\strut 1\over\displaystyle x_3 +
{\strut 1\over\displaystyle x_4}}}}$$
```

A `\displaystyle` parancson kívül szükségünk volt a `\strut` parancsszóra is. Ennek hatására a TEX nem húzza közelebb a törtvonalakat — a kifejezés sorait — egymáshoz, vagyis „kimerevednek" a sorközök.

5.6. Mátrixok szedése

Mátrixok szedésére a `\matrix` parancs szolgál. A mátrix elemeit a parancsszó után kapcsos zárójelek közé kell tenni, és ezen belül ki kell jelölnünk, hogy mi legyen a mátrix egy eleme ill. egy sora. Az összetartozó elemeket az `&` jellel adhatjuk meg, a sorokat pedig a `\cr` parancsszóval különítjük el. Például a következő egyszerű mátrixhoz

$$\begin{pmatrix} a & 0 \\ 0 & b \end{pmatrix}$$

ezt kell beírnunk:

```
$$\left(\matrix {a & 0 \cr 0 & b \cr} \right)$$
```

A \matrix parancs nem állítja elő a mátrixot közrefogó zárójeleket, csak sorokba, oszlopokba rendezi az elemeket. A { után álló betűket, jeleket, kifejezéseket a mátrix első sorának első elemébe teszi, egészen addig, míg egy &-jelre akad. Ettől kezdve minden elemet a mátrix első sorának második elemébe tesz egészen addig míg egy újabb & jelet nem talál stb. Ha karakterek feldolgozása közben \cr parancsra akad, új sort nyit, és elkezdi feltölteni elemekkel.

Az & karaktert bármely más karakterrel egybeírhatjuk, ill. tetszőlegesen sok szóközt hagyhatunk a karakterek között. Ezek egyáltalán nem befolyásolják a mátrix formáját. A \cr parancsot azonban nem írhatjuk egybe a következő sor elemeivel, mert akkor a TEX nem ismeri föl.

A TEX egyes elemeket, amelyek bonyolult függvénykifejezések is lehetnek, az oszlopoknak megfelelően középre helyezi. Az így kiszedett táblázatot (a következő fejezetben látni fogjuk, hogy ez tulajdonképpen az), tetszőleges zárójelek közé zárhatjuk. Itt is hasznos, ha a TEX-re hagyjuk a zárójek nagyságának kiválasztását, és a \left... és a \right... parancsokat használjuk: Például az

$$A = \begin{bmatrix} x-\lambda & x_{1,2} & x_{1,3}^2 \\ x_{2,1} & x-\lambda & x_{2,3}^2 \\ x_{3,1}^2 & x_{3,2} & x-\lambda \end{bmatrix}$$

mátrix kiszedéséhez ezt gépeltük be:

```
$$A=\left[\matrix{
x-\lambda & x_{1,2} & x_{1,3}^2 \cr
x_{2,1} & x-\lambda & x_{2,3}^2 \cr
x_{3,1}^2 & x_{3,2} & x-\lambda \cr
}\right]$$
```

Nagyméretű mátrixok esetén nem lehet minden elemet megadni, ekkor pontokkal szokták jelezni a hiányzó elemeket:

$$M = \begin{vmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{vmatrix}$$

A megfelelő pontozást az \ldots, \vdots, \ddots parancsokkal hozhatjuk létre. Mindegyik paranccsal három—három pontot (függőlegesen, vizszintesen, ill. átlós irányban) helyezhetünk el. Az előbbi mátrix tehát:

```
$$M=\left| \matrix{
a_{11}&a_{12}&\ldots&a_{1n}\cr
a_{21}&a_{22}&\ldots&a_{2n}\cr
\vdots&\vdots&\ddots&\vdots\cr
a_{m1}&a_{m2}&\ldots&a_{mn}\cr
}\right|$$
```

Leggyakrabban gömbölyű zárójelekbe zárt mátrixok szerepelnek matematikai szövegekben. Ezek szedésének megkönnyítését szolgálja a \pmatrix parancs, amely egyenértékű a \left(\matrix{ .....}\right) parancsok kiadásával. A szögletes zárójelben megadott mátrixot így írhatjuk át a \pmatrix segítségével:

```
$$A=\pmatrix{
x-\lambda & x_{1,2} & x_{1,3}^2 \cr
x_{2,1} & x-\lambda & x_{2,3}^2 \cr
x_{3,1}^2 & x_{3,2} & x-\lambda \cr
}$$
```

$$A = \begin{pmatrix} x-\lambda & x_{1,2} & x_{1,3}^2 \\ x_{2,1} & x-\lambda & x_{2,3}^2 \\ x_{3,1}^2 & x_{3,2} & x-\lambda \end{pmatrix}$$

5.7. Több kifejezésből álló elrendezések

Függvények megadásánál, egyenletrendszerek, feltételrendszerek szedésekor nem különálló kifejezéseket, hanem azok együttesét kell arányosan elhelyezni.

Erre van szükség akkor, ha például a függvény értékét esetek szétválasztásával adjuk meg:

$$|x| = \begin{cases} x, & \text{ha } x \geq 0; \\ 0, & \text{ha } x = 0; \\ -x & \text{máskülönben.} \end{cases}$$

Ehhez a \cases parancsot használjuk. Az ilyen szerkezetek hasonlítanak a mátrixokhoz, az egyes elemek, sorok, megadásában, de három ponton eltérnek tőle:

- csak egy zárójelet, elhatároló jelet tartalmaznak,
- az egyes elemek balra vannak sorkizárva,
- a szerkezeten belül az & jelig matematikai módban van a TEX, majd az & jel után szövegmódba vált át automatikusan. Ezért ha matematikai szimbólumokat, képleteket akarunk itt elhelyezni, akkor be kell kapcsolnunk a matematikai módot

Az előbbi példát így állítottuk elő:

```
$$|x|=\cases{x, &ha $x\ge0$;\cr
0 &ha $x=0$;\cr
-x & máskülönben.\cr}$$
```

A példában a nagy kapcsos zárójelet a TEX a \cases parancs hatására szedte ki, a \cases után álló kapcsos zárójel csak környezetkijelölő jel, amelyik nem jelenik meg. A kapcsos zárójelek között megadott szöveget a TEX először sorokra bontja a \cr parancsok alapján. A sorokon belül az első elemeket balra kizárva egymás alá helyezi, majd az & jel után álló elemeket ismet balra kizárva teszi egymás alá.

$$A_{megold} = \begin{cases} finom & \text{ha } X\text{-nek tetszik, } Y\text{-nak tetszik, } < 1988 \\ kellemes & \text{ha } X\text{-nek tetszik, } Y\text{-nak tetszik} \\ nem - az - igazi & \text{ha } Y\text{-nak nem tetszik} \\ rossz & \text{ha } \geq 2000 \end{cases}$$

```
$$A_{megold}=
```

```
\cases{finom& ha $X$-nek tetszik, $Y$-nak tetszik, $<1988$\cr
kellemes& ha $X$-nek tetszik, $Y$-nak tetszik \cr
nem-az-igazi& ha $Y$-nak nem tetszik \cr
rossz& ha $\geq 2000$ \cr }$$
```

Mátrix-szerű elrendezést kivánnak az egyenletrendszerek is. Ezek több sorból álló, egyetlen kiemelt kifejezést alkotnak úgy, hogy az egyenlőségjelek egymás alá kerülnek:

$$\begin{aligned} 2x + y + z + 13 &= 4n + k + 200 \\ 2x + 3y + z &= k + 99 \\ z + 1 &= n - k \\ x - y - z + 20003 &= 1 \end{aligned}$$

Ilyen többsoros kifejezéseket a `\eqalign` paranccsal szedhetünk.[9] A parancs után kapcsos zárójelben adjuk meg az egyes egyenleteket, a középre helyezendő karakter elé tesszük a `&` jelet, és az egyenlet végét a `\cr` paranccsal jelezzük a TEX-nek. A parancs nagyon hasonló a `\matrix`, parancshoz, de lényeges különbség, hogy — mint a `\cases` parancsnál is — az `\eqalign` parancsnál minden egyes sorban (nem a forrásszöveg, hanem a kiszedett szöveg egy sorában) csak egyszer használhatjuk a `&` jelet, különben — a meglehetősen gyakori —

```
!Extra alignment tab has been changed to \cr.
```

hibaüzenetet kapjuk. Ennek értelme az, hogy a második `&` jelet `\cr` parancsként értelmezte a TEX. Az előbbi egyenletrendszert a következőképpen készítettük:

```
$$\eqalign{
2x+y+z+13 &=4n+k+200\cr2x+3y+z &=k+99\cr
z+1 &=n-k\cr x-y-z+20003 &=1\cr}$$
```

Az egyenlőségjelek helyett természetesen más jelet, betűt is megadhatunk, és akkor azok kerülnek középre, egymás alá.

---

[9] a parancs az angol 'equation'(= egyenlőség) és 'align'(=elrendez) szavakból származik

Az \eqalign használható arra is, hogy több egymás mellett elhelyezett egyenletrendszert szedjünk vele. Például:

$$\begin{aligned} f'(x) &= \alpha(x) + \beta(x) + c_1 \\ g'(x) &= \alpha^2(x^2) + c_2 \\ h'(x) &= \beta^2(2x) + c_3 \end{aligned} \qquad \begin{aligned} \alpha^3(x) - \beta^3(x) &= x + y \\ \alpha(x) + \beta(x) &= x^2 - y^2 \end{aligned}$$

Ehhez az kell, hogy egyetlen kiemelt kifejezést (**$$...$$**) használjunk, és ebbe rakjuk bele mindkét \eqalign parancsot:

```
$$\eqalign{f^\prime (x) &= \alpha(x)+ \beta(x)+c_1\cr
g^\prime(x) &= \alpha^2(x^2)+c_2\cr
h^\prime(x) &= \beta^2(2x)+c_3\cr}
\eqalign {\alpha^3(x)-\beta^3(x) &=x+y \cr
\alpha(x)+\beta(x) &= x^2-y^2\cr}$$
```

Ez a parancssorozat valóban azt eredményezi, hogy a két egyenletrendszer egymás mellett áll, de nem azt kapjuk amit bemutattunk, hanem:

$$\begin{aligned} f'(x) &= \alpha(x) + \beta(x) + c_1 \\ g'(x) &= \alpha^2(x^2) + c_2 \\ h'(x) &= \beta^2(2x) + c_3 \end{aligned} \qquad \begin{aligned} \alpha^3(x) - \beta^3(x) &= x + y \\ \alpha(x) + \beta(x) &= x^2 - y^2 \end{aligned}$$

A kettő között az a különbség, hogy az eredeti példánkban a két egyenletrendszer között megfelelő hely van ahhoz, hogy ne „olvadjanak egybe". Ezt a helyet a már bemutatott \qquad paranccsal értük el úgy, hogy a két \eqalign közé tettük:

```
$$\eqalign{f^\prime (x) &= \alpha(x)+ \beta(x)+c_1\cr
g^\prime(x) &= \alpha^2(x^2)+c_2\cr
h^\prime(x) &= \beta^2(2x)+c_3\cr}
\qquad \eqalign {\alpha^3(x)-\beta^3(x) &=x+y \cr
\alpha(x)+\beta(x) &= x^2-y^2\cr}$$
```

Fontos észrevennünk, hogy most a \qquad paranccsal nem egyetlen soron belüli szóközt nyújtottunk meg, hanem a kifejezés *minden sorában* távolabb tettük az egyenleteket. Ennek oka az, hogy a TEX az \eqalign

„hasában" levő egyenleteket egyetlen dobozként kezeli, így ha parancsot adunk arra, hogy távolabb tegye őket, az egész egynletrendszert, mint egy dobozt mozgatja el.

Ez az elv érvényesül akkor is, ha az egynletrendszert — vagy más hasonló elrendezést — jelekkel össze akarunk fogni:

$$\left\{\begin{aligned} f'(x) &= \alpha(x)+\beta(x)+c_1 \\ g'(x) &= \alpha^2(x^2)+c_2 \\ h'(x) &= \beta^2(2x)+c_3 \end{aligned}\right\} \qquad \left[\begin{aligned} \alpha^3(x)-\beta^3(x) &= x+y \\ \alpha(x)+\beta(x) &= x^2-y^2 \end{aligned}\right]$$

Ehhez ugyanis elég, ha a kifejezéseket a `\left` ill. `\right` parancsok segítségével a megfelelő típusú zárójelek közé tesszük. A TEX automatikusan fogja kiválasztani a megfelelő méretű zárójeket, amely az egyenletrendszert — mint egyetlen dobozt — átfogja:

```
$$\left\{\eqalign{f^\prime(x)&=\alpha(x)+\beta(x)+c_1\cr
g^\prime(x) &= \alpha^2(x^2)+c_2\cr
h^\prime(x) &= \beta^2(2x)+c_3\cr}\right\}
\qquad \left[\eqalign {\alpha^3(x)-\beta^3(x) &=x+y \cr
\alpha(x)+\beta(x) &= x^2-y^2\cr}\right]$$
```

Az `\eqalign` „továbbfejlesztett" változata az `\eqalignno` amely az egyenletek mellé odateszi a hivatkozási számot, a címkét.

$$\left(\int_{-\infty}^{\infty} e^{-x^2}\,dx\right) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} e^{-(x^2+y^2)}\,dx\,dy \qquad (13)$$
$$= \int_0^{2\pi}\int_0^{\infty} e^{-r^2} r\,dr\,d\theta$$

Használata mindenben megegyezik az `\eqalign` használatával kivéve azt, hogy az `\eqalignno` esetében két `&` jelet is megadhatunk egy sorban. A második `&` jel után kell állnia a címkének abban a formában ahogy a lap szélén majd látni akarjuk (pl. gömbölyű zárójelek között), és ez után már csak a `\cr`, a sor végét jelölő parancs állhat. A címkét ahhoz a sorhoz teszi a TEX, amelyben megadtuk. Az előbbi kifejezés forrásszövege:

```
$$\eqalignno{\left(\int_{-\infty}^\infty e^{-x^2}\,dx\right)
&=\int_{-\infty}^\infty \int_{-\infty}^\infty
e^{-(x^2+y^2)}\,dx\,dy & (13) \cr
&=\int_0^{2\pi}\int_0^\infty e^{-r^2} r \,dr\,d\theta \cr}$$
```

Az \eqalign parancstól csak kissé különbözik a \leqalignno parancs, amely a címkét nem a jobb, hanem a bal margóra teszi. Ha \leqalignno parancsban címkét adunk meg, ennek is közvetlenül a \cr előtt kell állnia:

```
$$\leqalignno{\left(\int_{-\infty}^\infty e^{-x^2}\,dx\right)
&=\int_{-\infty}^\infty \int_{-\infty}^\infty
e^{-(x^2+y^2)}\,dx\,dy & (13) \cr
&=\int_0^{2\pi}\int_0^\infty e^{-r^2}r\,dr\,d\theta\cr}$$
```

$$\left(\int_{-\infty}^{\infty} e^{-x^2}\,dx\right) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} e^{-(x^2+y^2)}\,dx\,dy = \int_0^{2\pi}\int_0^{\infty} e^{-r^2} r\,dr\,d\theta \tag{13}$$

5.8. Matematikai függvények

Matematikai módban minden betű a matematikai kurzív fontból jön. Vannak azonban olyan matematikai függvények amelyeket képletekben, kifejezésekben is antikva betűvel szednek. Ha azt akarjuk kiszedni, hogy

$$\sin 2\alpha = 2\sin\alpha\cos\alpha$$

akkor nem írhatjuk ezt:

```
$$sin 2 \alpha =2 sin \alpha cos \alpha$$
```

mert ez így jelenik meg:

$$sin2\alpha = 2sin\alpha cos\alpha.$$

```
\arccos  \arcsin  \arctan  \arg
\cos     \cosh    \cot     \coth
\csc     \deg     \det     \dim
\exp     \gcd     \hom     \inf
\ker     \lg      \lim     \liminf
\limsup  \ln      \log     \max
\min     \Pr      \sec     \sin
\sinh    \sup     \tan     \tanh
```

Bár megtehetnénk, hogy matematikai módban is bekapcsoljuk az antikva betűtípust, ez a kapcsolgatás sok munkával és sok hibalehetőséggel járna. Ehelyett használjuk a függvényeket előállító parancsszavakat, amelyek a legfontosabb 32 függvény esetében érhetők el:

A parancsok begépelése csak annyi többletmunkát okoz, hogy az egyes függvények neve elé a \ vezérlőkaraktert kell tennünk.

Azoknál a függvényeknél, amelyek indexszel állhatnak, ezt a szokásos módon, a _ jellel adhatjuk meg. Ezek sokszor nem igazi indexek: például a lim függvény esetében az index nem az m-betű után, hanem ahogy a matematikai képletekben szokás, alatta áll:

```
$\lim_{n \to \infty} a^n_n=0$
```

$$\lim_{n\to\infty} a_n^n = 0$$

```
$\lim_{x \to 0}{\sin x \over x}=1$
```

$$\lim_{x\to 0} \frac{\sin x}{x} = 1$$

```
$\lim_{n \to \infty} x_n \exists
\Longleftrightarrow\limsup_{n \to \infty}
x_n=\liminf_{n \to \infty} x_n$
```

$$\lim_{n\to\infty} x_n \exists \Longleftrightarrow \limsup_{n\to\infty} x_n = \liminf_{n\to\infty} x_n$$

```
$H(X)=\sum_{i=1}^n
```

```
\Pr(X=x_i)\log_2\Pr(X=x_i)$
```

$$H(X) = \sum_{i=1}^{n} \Pr(X = x_i) \log_2 \Pr(X = x_i)$$

```
$\max_{1 \leq n \leq m}
\sinh^2(nx+\pi)/f^n(x)$
```

$$\max_{1 \leq n \leq m} \sinh^2(nx + \pi)/f^n(x) .$$

6. fejezet

Táblázatok, ábrák

A fejezetben azt írjuk le, hogyan lehet a TEX programot táblázatok, ábrák, illetve egyéb, a kinyomtatandó dokumentumhoz tartozó, de nem szövegszerű részletek készítésére felhasználni. A szövegszedő program ábrákat nem tud rajzolni, de azok helyét kihagyathatjuk, feliratát elkészíthetjük vele. Előbb azonban tekintsük át még egyszer, részletesebben a dobozokról tanultakat, hiszen, mint majd kiderül, ebben a fejezetben nagy szükségünk lesz ezekre az ismeretekre.

6.1. Dobozok

Mint azt a korábbi fejezetekben már láttuk, a \hbox és \vbox parancsokkal hozhatjuk létre a TEX ún. dobozait, amelyekbe a program az első esetben vízszintesen, a második esetben függőlegesen helyezi el egymás után a változóban megadott szövegrészleteket, az alábbi példa szerint:

```
Itt van                    \hbox{két sor\vbox{\hbox{Itt
két sor egy dobozban       van}\hbox{egy dobozban}}}
```

Felmerülhetnek további igények is a dobozok készítésekor. Először is szükség lehet arra, hogy a dobozok méreteit mi határozhassuk meg. A \hbox magassága adott, a szélességét viszont a `hbox to 20pt` illetve \hbox spread 20pt típusú parancsokkal befolyásolhatjuk. Mindkét esetben a parancs után ugyanúgy kell megadni a doboz tartalmát, mintha egy közön-

séges dobozról lenne szó. Az első esetben a doboz szélessége 20 pont lesz, már amennyiben a program képes a benne szereplő rugalmas mértékek felhasználásával ilyen szélesre szedni a tartalmát. Ha nem, akkor az oldaltördelésből már megismert `Overfull \hbox` vagy `Underfull \hbox` típusú figyelmeztető üzenetet kapjuk, és természetesen ennek megfelelően egy olyan dobozt, amelynek vagy határain túlra is kiterjed a tartalma, vagy nincs eléggé kitöltve. A `\hbox spread ...` parancs esetén nem azt határozzuk meg a parancsban adott mértékkel, hogy milyen széles legyen a doboz, hanem azt, hogy *mennyivel legyen szélesebb,* mint a természetes szélessége. Így a következő dobozokat a `\hbox spread 0pt`, `\hbox spread 10pt`, `\hbox spread 20pt` és `\hbox spread 35pt` parancsokkal szedtük ki:

Figyeljük meg a sor hosszának változását.
Figyeljük meg a sor hosszának változását.
Figyeljük meg a sor hosszának változását.
Figyeljük meg a sor hosszának változását.

Mivel az első változatban a `0pt` jelentése éppen az, hogy a program a természetes szélességére szedje a sort, amit az alapértelmezés szerint is megtesz, a `spread 0pt` részt nem szükséges kitenni.

A `to` és `spread` parancsokat a `\vbox` paranccsal együtt is használhatjuk, amikor is természetesen azok nem a doboznak a szélességére, hanem a magasságára vonatkoznak. Ilyenkor kicsit óvatosnak kell lennünk, mivel ezeknek a dobozoknak nemcsak magasságuk, hanem mélységük is van. Anélkül, hogy kitérnénk azokra a bonyolult szabályokra, ahogy a program a `\vbox` dobozok magasságát és mélységét számolja, maradjunk csak annyiban, hogy ezek függőleges kiterjedése nagyobb részt a magassághoz és csak nagyon kis részt a mélységhez tartozik.

Nagyon fontos annak meghatározása, hogy az általunk létrehozott dobozok milyen módon kerülnek az adott oldalra. Ezt a legegyszerűbben úgy dönthetjük el, hogy az adott doboz helyére egy betűt képzelünk. A program oda helyezi a dobozt, ahova a betű kerülne. Persze egy doboz mérete sokkal nagyobb lehet, mint egy betűé, ebből adódhatnak az eltérések. Így még azt is tudnunk kell, hogy a doboz melyik pontja lesz az, amelyik az éppen érvényes alapvonalra kerül. Amennyiben `\hbox`-ról van szó, úgy annak alapvonala a bele kerülő egységek közös alapvonala lesz, míg egy `\vbox` esetén annak legalsó sorának az alapvonala. Egy-egy doboz referenciapontja

mindig az alapvonal kezdőpontjára esik, ami majdnem mindig annak bal végpontja. Az egyetlen kivétel, ha egy \hbox-ban a to vagy spread utasításokban negatív mértéket határozunk meg, ami által a doboz kiterjedése negatív lesz, azaz a megszokott helyzettől eltérően nem balról jobbra, hanem jobbról balra terjed. Ebben az esetben az alapvonal kezdőpontja annak jobb végpontja.

A dobozok fenti automatikus elhelyezését azonban befolyásolhatjuk. Amennyiben az adott doboz vízszintesen kerülne az adott (azaz az előtte levő legutolsó jel vagy másik doboz *mellé, és nem alá* illesztené a program), akkor a vízszintes elhelyezkedését az előtte kiadott szokásos \vskip parancs írhatja felül, míg a függőleges elhelyezést a korábbról már ismert lower és raise parancsok helyesbíthetik. A \lower-ben kiadott mértékkel lejjebb kerül egy doboz, míg a \raise-zel kiadott mértékkel feljebb. Így aztán a \lower3pt\hbox{...} utasítás teljesen azonos hatású a \raise3pt\hbox{...} utasítással. A már korábban használt példánkat a következőképpen módosíthatjuk ennek szemléltetésére:

két sor Itt van egy dobozban

```
\raise5pt\hbox{két sor}
\vbox{\hbox{Itt van}
\hbox{egy dobozban}}
```

Ha egy doboz az oldalon a legutóbb oda került dologhoz képest függőlegesen, az alá kerülne, akkor függőleges elhelyezését a már régen ismert \vskip paranccsal befolyásolhatjuk, míg a vízszintes elhelyezését a \lower és \raise-hez hasonló működésű \moveright és \moveleft parancsok írhatják felül. A program az ezek után kiadott mértékkel jobbra illetve balra mozdítja el az utánuk következő dobozokat. Ebben az esetben is fennáll, hogy a negatív mértékkel kiadott \moveright utasítás ugyanazt a hatást váltja ki, mintha annak ellentettjével adtunk volna ki egy \moveleft parancsot. Megint a már ismert példával szemléltetjük ezek hatását.

két sor Itt van egy dobozban

```
\raise5pt\hbox{két sor}
\vbox{\hbox{Itt van}
\moveright15pt\hbox{egy ...}}
```

Van még egy utasítás, ami a \vbox parancs esetén sokat segíthet a doboz függőleges elhelyezésében. Ha a \vbox helyett a \vtop utasítással hozunk létre egy dobozt (ahol a doboz meghatározása a parancsszón kívül minden

egyébben megegyezik azzal, amit a \vbox-nál tanultunk), akkor annak alapvonala nem a benne szereplő legalsó sor alapvonalával egyezik meg, hanem a legfelső soréval. A \vbox utasítással létrehozott dobozokkal ellentétben tehát az ilyen dobozoknál a doboz függőleges kiterjedése legnagyobbrészt annak mélységéhez járul hozzá, és mindössze a legfelső elemének (sorának, \hbox-ának, stb.) magassága eredményezi a \vtop doboz magasságát. Használatával az eddigi példánk a következőképp módosul.

Itt van két sor
egy dobozban

```
\hbox{Itt van
\vtop{\hbox{két sor}
\hbox{egy dobozban}}}
```

Ugyan első látásra nem derül ki, hogy a következő két utasításnak mi a köze a dobozokhoz (mi pedig nem magyarázzuk most el — a könyv felépítése szempontjából ez nem fontos), mégis itt ismertetjük őket. Néha előfordulhat, hogy valamit ki szeretnénk írni a szövegben, de nem akarjuk, hogy annak jelenlétét a program figyelembe vegye. Erre a legjobb példa talán az az eset, amikor két jelet szeretnénk egymásra írni, hogy mondjuk a $\neq$ jelet megkaphassuk[1]. Ilyen helyzetekben használhatjuk fel a \rlap és \llap utasításokat, amelyek közül az első hatására az oldalon a képzeletbeli kurzor pozíciója (tehát az a hely, ahova a program majd a következő jelet kiteszi) nem változik, noha attól jobbra megjelenik a parancs változójában megadott szövegrészlet. Ennek segítségével a fenti jel a \rlap/= paranccsal hozható létre. A másik, \llap utasítás igen hasonló hatású, a képzeletbeli kurzort ez sem mozdítja el, mindössze a változójában megadott dolgokat attól balra (az esetek többségében a már meglévő szövegrészletben) helyezi el. A /\llap= parancs ugyanazt a $\neq$ jelet hozza létre, mint a fenti másik parancs.

A dobozok speciális fajtája (talán inkább ezeket gondolnánk „dobozoknak") a teljesen kitöltött (azaz fekete) téglalap. Ezeket a speciális \hrule illetve \vrule parancsokkal lehet létrehozni. Ne felejtsük el, hogy a hosszú egyenes vonal is egy ilyen (kicsit vékonyra sikerült) téglalap, tehát ezeket is a fenti parancsok állítják elő. Amennyiben nem adjuk meg a téglalapok méreteit, a \hrule parancs egy olyan vízszintes vonalat hoz létre, amely-

[1] Ezt ugyan megteszi nekünk a \ne utaítás is, de nem akarunk bonyolultabb példát hozni.

nek vastagsága előre adott (4 pont), hossza viszont az adott környezettől függ. Hasonlóan, a \vrule egy éppen ilyen jellemzőkkel rendelkező függőleges vonalat hoz létre. Ha meg kívánjuk adni a téglalap méreteit, már teljesen mindegy, hogy melyik parancsot használjuk. Például a `\vrule height6pt width5pt depth2pt` parancs egy olyan téglalapot hoz létre, amelynek a hossza (ami most mindenképpen a vízszintes kiterjedést jelenti) 5 pont (ezt a dimenziót adja meg a `width` paraméter), az alapvonal fölé nyúló magassága (ezt adja meg a `height` paraméter) 6 pont és az az alá nyúló mélysége (amit a \depth paraméter ad meg) 2 pont, azaz ▮. Mint említettük, ilyen esetekben már mindegy, hogy melyik parancsot használjuk a kettő közül, pontosan a ▮ téglalapot hozta volna létre a `\hrule height6pt width5pt depth2pt` parancs is. Az egyetlen különbség köztük, hogy a \vrule vízszintesen adja hozzá az oldalon szereplő szövegrészlethez a dobozt, míg a \hrule függőlegesen. Ha nem téglalapokat, hanem egyenes vonalakat rajzoltatunk a parancsokkal, akkor ezt is várjuk tőlük. Például az oldal teljes szélességében húzodó vízszintes vonalat nyilván a már meglévő anyag alá szretnénk húzni.

A \hrule parancs egy változata a \hrulefill. Ezt olyan esetekben használjuk, amikor nem tudjuk, pontosan milyen hosszú vízszintes vonalat akarunk húzni, mindössze annyit tudunk, hogy a kihagyott üres helyet azzal szeretnénk kitölteni. Azaz ez a parancs nem más (ahogy neve is mutatja), mint a `hfill` és \hrule parancsok ötvözete. Használatát jól szemlélteti az alábbi példa:

Egy_____kettő `Egy \hrulefill kettő`

Talán még többször előfordul, hogy vízszintes vonal helyett kipontozással szeretnénk a kihagyott részeket kitölteni. A \dots parancs erre nem alkalmas, hiszen ez csak három pontot tesz ki. Helyette használhatjuk a `hrulefill` mintájára működő \dotfill parancsot:

Egy.....kettő `Egy \dotfill kettő`

6.2. Táblázatok készítése

6.2.1. Állandó szélességű oszlopk

Nagyon sokszor előfordul, hogy könyvekben vagy egyéb szövegekben különálló oszlopokba kell szedni egyes szövegrészleteket. Az írógépen a gépíróknak az oszlopok közötti ugrásra a (tab) billentyű áll a rendelekzésére. A TEX-ben ezt a következő mechanizmus helyettesíti. A `\settabs` *n* `\columns` parancssorozat a rendelkezésre álló vízszintes szélességet (általában az oldal szélességét, de például egy doboz belsejében természetesen annak a szélességét) *n* egyenlő szélességű oszlopra osztja. Ha ezek után egy sort a `\+` paranccsal kezdünk, akkor abban az `&` jelekkel elválasztott szövegrészletek mindig a következő új oszlopban kezdődnek, mindaddig, amíg egy `\cr` paranccsal le nem zárjuk a sort. Nézzünk egy példát:

```
\settabs 3 \columns
\+ & Ez a szövegrészlet a második oszlopban kezdődik.\cr
\+ Ez az első oszlop. &&Ez a harmadik. \cr
\+\sl Ez megint az első oszlop, csak hosszu.&&
Ez megint a harmadik.\cr
```

Ezen parancsok hatására a következő részlet jelenik meg a szövegben:

	Ez a szövegrészlet a második oszlopban kezdődik.	
Ez az első oszlop		Ez a harmadik.
Ez megint az első oszlop, csak hosszu.		Ez megint a harmadik.

Megfigyelhetjük, hogy a fenti utasítások nagyon hasonlítanak az írógép tabulátor billentyűjéhez. Megvan viszont az a lényeges eltérés, hogy a `&` jel hatására a program nem a fizikailag következő (azaz a befejezett szövegrészlettől jobbra levő) pozícióba ugrik, hanem az elméletileg következőre. A fenti példa harmadik sorában az első oszlopba került szöveg bőven belenyúlik a másodikba, mégis két `&` jelet kellet felhasználnunk, hogy utána a harmadik oszlop elejére kerüljünk. A `\cr` parancs, ami az angol carrige return (kocsi vissza) szavak rövidítéséből ered, mindig új sort kezd, akkor is, ha nincsen

kitéve az összes oszlophoz az azokat kezdő & jel. Figyeljük meg, hogy a \+ parancsot és a & jelet követő szóközök nem számítanak, a program nem veszi őket figyelembe, valamint hogy az egyes oszlopokon belül kiadott parancsok (például a fenti \sl parancs) csak az oszlopon belül érvényesek.

Ha egyszer kiadtunk egy \settabs ... \columns parancsot, akkor az annak megfelelő beosztás mindaddig érvényben marad az adott környezeten belül, amíg azt egy másikkal felül nem írjuk. Természetesen hatását csak azokban az általunk meghatározot sorokban fejti ki, amelyeket \+ paranccsal kezdünk. Az elején, az alapértelmezés szerint mindössze egyetlen oszlopra van osztva az elméleti oldal, aminek természetesen sok gyakorlati haszna nincsen. Ha egy környezeten belül állítjuk be az oszlopok számát, akkor az a parancs csak az adott környezeten belül érvényes, azon kívül a régi oszlopbeállítás marad érvényben. Egyetlen dologra hívjuk még fel a figyelmet: az & jel ebben az esetben nem parancsjel (nem előzi meg a \ jel), mindössze egy speciális karakter. Mivel ilyen kitüntetett szerepe van, mint azt már említettük, írásjelként nem használható (a & jel előállítására a \& parancs szolgál).

Sokszor előfordul, hogy a táblázatok készítésekor nem ugyanolyan szélességű oszlopokra van szükségünk, hiszen az egyik oszlopba kerülő anyag jellegénél fogva sokkal keskenyebb lehet, mint a másikba kerülő. Ilyenkor az oszlopok számát megadó sorban a \settabs parancs után nem az eddig megadott részletet szerepeltetjük, hanem egy mintasort, mint a következő példában: [2]

```
\settabs\+\indent&Az oldal alakja\qquad&\cr
\+&Az oldal alakja&Szedéstükör mérete\cr
\+ &A4 &172mm$\times$243mm \cr
\+ &A5 &108mm$\times$162mm \cr
\+ &B5 &126mm$\times$189mm \cr
```

Az oldal alakja	Szedéstükör mérete
A4	172mm×243mm
A5	108mm×162mm
B5	126mm×189mm

[2] Ez a példa, amely azt mutatja meg, hogy néhány nyomdai szabványoldalnak mekkora a szedéstükörje, végigvonul majd az egész fejezeten.

Mint láthatjuk, az utolsó oszlopot nem kell szerepeltetnünk a mintasorban. Természetesen célszerű az oszlopból a legszélesebb szövegrészletet tenni a mintasorba, hacsak nem akarjuk az oszlopok egymásra csúszását egyes sorokban. Figyeljük meg azt is, hogy amennyiben a sorok között távolságot szeretnénk tartani, azt szerepeltetnünk kell a mintasorban. Az utolsó oszlop szélességét nem szükséges kijelölnünk, de annak létezését jelölni kell. Ezért szerepel a mintasorban az utolsó & jel. A mintasor és a felhasználás első oszlopai csak arra szolgálnak, hogy a táblázat ne kezdődjön mindjárt az oldal bal szélén, hanem egy bekezdéssel beljebb.

Itt, és a továbbiakban segítségünkre lehetnek a részben már ismert \enskip, \quad és \qquad parancsok, amelyek a megadott sorrendben egy fél „em"-nyi, egy egész „em"-nyi és két „em"-nyi vízszintes helyet (beosztást) hagynak ki, ahogy a következő példa ezt szemlélteti.

```
a b c    d                    a\enskip b\quad c\qquad d
```

A fenti táblázat bizony nem nagyon szép. Sokat javítana a külalakján, ha a táblázat elemeit az oszlopaikban középre zárnánk. Mint megfigyelhettük, ha ezt a programra bízzuk, az magától balra zárja minden egyes oszlopban a szöveget, mint ahogy azt az egyéb szövegrészletekkel is teszi. Ennek megfelelően a középre vagy balra zárást a szokásos \hfill parancsokkal érhetjük el. *Felhívjuk a figyelmet, hogy az oszlopkezelő parancsok esetén csak a* \hfill *parancsot használjuk, a* \hfil*-t ne, mivel az váratlan hatásokat eredményezhet.* Ha tehát a fentebbi példában a második sortól kezdődően az egyes oszlopokban szereplő anyagot középre kívánjuk zárni, a következőképpen járjunk el:

```
                                \settabs\+Az oldal\quad&\cr
                                \+Az oldal&Szedéstükör mérete\cr
Az oldal  Szedéstükör mérete    \+\hfill A4\hfill\quad&\hfill
  A4      172mm×243mm           172mm$\times$243mm\hfill\cr
  A5      108mm×162mm           \+\hfill A5\hfill\quad&\hfill
  B5      126mm×189mm           108mm$\times$162mm\hfill\cr
                                \+\hfill B5\hfill\quad&\hfill
                                126mm$\times$189mm\hfill\cr
```

6.2.2. Bonyolultabb táblázatok

Az előző módszer sok esetben megoldja a táblázatkészítéskor felmerülő problémákat, de van, amikor nem elegendő. Sok más hibája mellett az egyik legalapvetőbb, hogy amennyiben nem egyenlő szélességű oszlopokat kívánunk beállítani, akkor magunknak kell megsejteni az oszlopba kerülő legszélesebb szöveget. A most következő, a TEX által a táblázatkészítéshez nyújtott másik módszer kicsit bonyolultabb lesz, de feltétlenül sokkal hatékonyabb. Ebben az esetben a teljes táblázat a `\halign` parancs változójában szerepel, és így természetesen a rá adott meghatározások is csak itt érvényesek. Ez adja a módszer talán egyetlen hátrányát az előzővel szemben, tudniillik, hogy két egymás után következő táblázatnak így nincsen kapcsolat, nem lehet az oszlopaikat egymáshoz igazítani.

A `\halign` parancs változójában is `\cr` parancsok választják el az egyes sorokat, azon belül `&` jelek az egyes oszlopokat, itt is egy mintasor szerepel először, de nem kell az egyes sorokat `\+` paranccsal kezdeni. Az eddigiekkel ellentétben itt a mintasor nem határozza meg az oszlopok szélességét, csak az oszlopokba kerülő szövegek azon közös részleteit helyezzük ide, amelyek minden sorban előfordulnak majd. Ezek után a program a változó befejezésekor maga számítja ki, milyen széles legyen egy-egy oszlop, hogy minden sorban elegendő hely legyen az ott szereplő anyag kiírására. A mintasorban minden oszlopban egy # jelzi, hogy az oszlop egyedi tartalma, amelyet a következő sortól kezdve sorolunk fel, hova kerül. A már jól ismert példánknál maradva, de egy kicsit tovább szépítve a táblázatot:

```
\halign{\indent\hfil#\hfil&\qquad\hfil#\hfil\cr
Az oldal alakja&Szedéstükör mérete\cr
A4 &172mm$\times$243mm \cr
A5 &108mm$\times$162mm \cr
B5 &126mm$\times$189mm \cr}
```

Az oldal alakja	Szedéstükör mérete
A4	172mm×243mm
A5	108mm×162mm
B5	126mm×189mm

A \halign parancs, az előző pontban szerepelt módszerrel szemben, maga számolja ki az oszlopok szélességét. Ehhez szüksége van a teljes változójára, csak annak beolvasása képes a leghosszabbat kiválasztani. Ezért túlságosan hosszú, több oldalon át tartó táblázatok szedésére nem alkalmas, hiszen azokat a számítógép memóriája nem képes egyszerre tárolni. Ezért a megkötésért cserébe nagyon sok hasznos tulajdonsága van. Ilyen többek között az, hogy ha egy táblázat sorai sokban hasonlítanak egymáshoz, akkor az azonos részeket elég egyszer kiszedni vagy meghatározni a mintasorban. Az alábbi táblázat, amely a fejrész[3] nélkül tartalmazza az eddigi példát, a következő igen rövid részlettel is kiszedhető:

```
\halign{\indent\hfil#\hfil&\qquad#mm$\times$&#mm\cr
A4&172&243\cr A5&108&162\cr B5&126&189\cr}
```

A4	172mm×243mm
A5	108mm×162mm
B5	126mm×189mm

Ha tovább akarunk javítani a táblázat külalakján, akkor keményebben kell dolgoznunk, bonyolultabb lesz az input. Nem tartozik szorosan a külalak javításához, de jó lenne elkerülni, hogy az oszlopok között nekünk kelljen definiálni a távolságot minden egyes esetben. A programnak van egy változója erre a célra, a \tabskip, amely a többi \skip típusú változóhoz hasonlóan működik. Ha megadjuk a \tabskip=2em plus 2em értéket, akkor annak megfelelően fogja a TEX az oszlopok közötti távolságot beállítani. A \tabskip által meghatározott mértéket nemcsak az egyes oszlopok közé, hanem az első oszlop elé és az utolsó oszlop mögé is odailleszti a program. A parancsot ki lehet adni az input file-ban bárhol, a \halign parancsokon kívül (a mérték alapértelmezés szerinti beállítása nulla) is, de akár a \halign parancs mintasorán belül is. Ez utóbbi esetben az oszlopok közötti távolságot a kiadás helyétől változtatja meg a parancs.

Sokszor előfordul, hogy a táblázat fejrésze és a többi rész között egy kicsit több helyet szeretnénk kihagyni, mint az átlagos sorok között. Sőt,

---

[3] Ime egy példa arra, hogy mennyire nehéz az egyes fogalaknak megfelelő elnevezést találni. Nyilván mindenki tudja, mit értünk a táblázat fejrészén, annak elleneére, hogy ugyanezt a dolgt a nyomdákban táblafejnek nevezik, máshol pedig esetleg a táblázat fejlécének.

időnként vonallal választjuk el a fejrészt az alatta következő soroktól. Ennek megoldására szolgál a \noalign parancs, amit a \halign változójában bármelyik \cr után kiadhatunk. Egy olyan részletet határoz meg, amely nem a következő sorba kerül, hanem a megadott helyen, a táblázat két sora közé. Így ha közvetlenül a fejrész után kiadjuk a \noalign{\smallskip} parancsot, akkor az egy kis extra helyet eredményez ott. Az ugyanott kiadott \noalign{\hrule} parancs pedig egy vízszintes vonallal választja el a fejlécet a táblázat törzsétől. Felhasználva az eddig tanultakat az alábbiak szerint módosíthatjuk táblázatunkat:

```
\tabskip=1em plus 3em minus 1em
\halign to 5cm{\hfil#\hfil&\hfil#\hfil\cr
\noalign{\hrule\smallskip}
Az oldal&Szedéstükör\cr
\noalign{\smallskip\hrule\smallskip}
A4 &172mm$\times$243mm \cr
A5 &108mm$\times$162mm \cr
B5 &126mm$\times$189mm \cr}
```

Az oldal	Szedéstükör
A4	172mm×243mm
A5	108mm×162mm
B5	126mm×189mm

Az oldal	Szedéstükör
A4	172mm×243mm
A5	108mm×162mm
B5	126mm×189mm

Valójában persze itt nem egy, hanem kettő \halign parancsot adtunk ki ugyanabban a sorban, amelyek csak egy kicsit különböznek egymástól. Mivel rugalmas mértékűnek határoztuk meg a \tabskip mértéket, lehetőségünk van arra, hogy a \halign által elfoglalt vízszintes távolságot is beállítsuk, csakúgy, mint a \hbox parancsok esetében. Ennek hatása, hogy a fenti példában a két, csaknem egyforma táblázat mégis különböző, a jobbolali szélesebb, jobban széthúzott. Egész pontosan a következő parancsokat adtuk ki:

```
\tabskip=1em plus 3em minus 1em
\halign to 5cm{ ... } \hskip1cm \halign to 6cm{ ... }
```

ahol a táblázatok megadása természetesen már az előbb leírt módon történt.

Az eddigi táblázatkezelő utasításokkal már elég sok dolgot meg tudunk csinálni. A továbbiakban következő módosítások bonyolultabbak lesznek, de még további lehetőségek előtt nyitják meg az utat. Először egy olyan dolgot említünk, amelyek ugyancsak megkönnyíti a munkát. Egy korábbi példában láttuk, hogy a \halign parancs mintasorában mindent megadhatunk, ami a későbbi sorokban közös, és így sok sor esetén alaposan lerövidíthetjük a gépelést. Csak az okozhat gondot, ha van egy vagy két olyan sorunk, amelyek eltérnek a többi sor általános jellemzőitől, és így miattuk nem használhatjuk ezt a módszert. Ezen segít az \omit parancs, amelyet a \halign egy sorának valamely elemében kiadva a program eltekint azon az egy helyen az oszlopnak a mintasorban megadott általános jellemzőjétől, helyette pontosan azt fogja felhasználni, amit az adott elem helyén talál. Próbáljuk meg ezt alkalmazni az állandó példánkra:

```
\halign{\indent\hfil#\hfil&\quad#mm$\times$&#mm\cr
Az oldal alakja&\omit\qquad\qquad\hidewidth Szedéstükör
\hidewidth&\omit\cr
A4&172&243\cr A5&108&162\cr B5&126&189\cr}
```

Az oldal alakja	Szedéstükör
A4	172mm×243mm
A5	108mm×162mm
B5	126mm×189mm

Figyeljük meg, hogy az \omit parancson kívül még a \hidewidht parancsot is használjuk a táblázatban. Ez, az oszlop tartalamának mindkét oldalára kitéve arra szolgál, hogy az adott szövegrészlet szélességét „elrejtsük", azaz a programot utasítsuk, hogy az oszlop szélességének kiszámításakor a parancsok között szereplő szöveget ne tekintse. Mivel a fenti esetben a „Szedéstükör" szó két oszlop helyén szerepelt, de csak egy oszlopba tudtuk beírni, így próbáltuk ezt a hatást kiegyenlíteni. Ugyancsak a „Szedéstükör" szó középre zárását szolgálja az előtte megadott két \qquad utasítás is. Elismerjük, hogy a végeredmény talán nem tökéletes és a módszer elég bonyolult. A továbbiakban erre a feladatra egy egyszerűbb megoldást is találunk majd. Itt most csak azt próbáltuk szemléltetni, miként használható az \omit parancs. Figyeljük meg, hogy a fejrészben még a legutolsó oszlopba is ki kellett tennünk egyet, hiszen ha azt az oszlopot egyszerűen üresen hagyjuk, a program úgy tekinti, oda a mintasorban meghatározott mm

szövegrészletet szánjuk. Ha egy oszlopban nem szeretnénk használni a mintasorban megadott általános részletet, az oszlop tartalmát mindig a \omit paranccsal *kezdjük*.

Térjünk most rá arra, hogy hogyan lehet a fenti bekezdésben felmerült problémát megoldani, azaz valamely sorban több oszlopot összevonni. Ha valahol a & jelet egy \span parancs helyettesíti, akkor a program az előtte és utána következő oszlopokat és szövegrészleteket is összevonja, az együttesen kapott szöveget egy olyan széles helyen helyezi el, amekkorát a két oszlop együttesen meghatároz. Ezen parancs segítségével tehát a következő módon oldhatjuk meg a problémánkat:

```
\halign{\indent\hfil#\hfil&\qquad#mm$\times$&#mm\cr
Az oldal alakja&\omit\qquad \hfil Szedéstükör
\span\omit\hfil\cr
A4&172&243\cr A5&108&162\cr B5&126&189\cr}
```

Az oldal alakja	Szedéstükör
A4	172mm×243mm
A5	108mm×162mm
B5	126mm×189mm

ahol az \omit \hfil Szedéstükör \span\omit\hfil\cr részletben a két \omit azt jelenti, hogy a két utolsó oszlopban nem vesszük figyelembe a mintasorban előre meghatározott általános jellemzőket, a span azt, hogy a két oszlopot ideiglenesen egynek tekintjük, és a két \hfil pedig azt, hogy a szöveget ezen belül középre zárjuk ki. Vigyázzunk, hogy a \span paranccsal összevont oszlopokban se feletjsük el kiadni a \omit parancsot, ha nem kívánjuk igénybe venni az oszlop általános jellemzőit.

Mint látjuk, az \omit és \span parancsokat egyszerre is használhatjuk. A kettő kombinációja a \multispan parancs, amely után egy számot megadva (többjegyű szám esetén kapcsos zárójelek között) azt mondhatjuk meg, hogy egyszerre hány oszlopot szeretnénk összevonni. A parancsot mindig az összevont oszlopok közül a legelsőnek a legelején adjuk ki, és csak utána adjuk meg az így létrejött széles oszlop tartalmát. Ez a parancs már magában foglalja a \omit parancsokat is, így azokat nem kell kiadni, mint a sima \span esetében. A példánk ezzel így módosulhat:

```
\halign{\indent\hfil#\hfil&\qquad#mm$\times$&#mm\cr
Az oldal alakja&\multispan2\qquad \hfil Szedéstükör\hfil\cr
A4&172&243\cr A5&108&162\cr B5&126&189\cr}
```

Az oldal alakja	Szedéstükör
A4	172mm×243mm
A5	108mm×162mm
B5	126mm×189mm

Az eddigi példákban a táblázat soha nem került pontosan az oldal közepére zárva, pedig ez a táblázatok szokásos elhelyezése. Idáig csak azt a megoldást használtuk, mint bizonyára sok olvasónak feltünt, hogy a táblázat mintasorának első oszlopában a `\indent` parancsot kiadva, minden sort egy bekezdésnyivel beljebb kezdtünk. A táblázatokat középre zárhatjuk oly módon, hogy egy `\centerline` parancs változójába tesszük őket. Ilyenkor a táblázat valóban középre kerül, de a fölötte és alatta kihagyandó függőleges beosztásról magunknak kell gondoskodnunk. Sokkal hatékonyabb eljárás, ha a matematikai módnak a képletek kiemelésére szánt `$$...$$` parancsát használjuk,egy `\vbox`-szal kiegészítve:

```
$$\vbox{\halign{\indent\hfil#\hfil&\qquad#mm$\times$&#mm\cr
Az oldal alakja&\multispan2\qquad \hfil Szedéstükör\hfil\cr
A4&172&243\cr A5&108&162\cr B5&126&189\cr}}$$
```

Az oldal alakja	Szedéstükör
A4	172mm×243mm
A5	108mm×162mm
B5	126mm×189mm

Az előzőekben megtanultuk, hogyan lehet táblázatban vízszintes vonalat húzni. Ebben a szakaszban utoljára azt tekintjük át, mit tudunk tenni a függőleges vonalakkal. Alapjában véve nagyon egyszerű a megoldás: ahol a táblázatban függőleges vonalat szeretnénk, tegyük ki a `\vrule` parancsot, amely az adott sorban az adott helyen nagyjából a sor magasságának megfelelő függőleges vonalat húz. Persze általában folyamatos vonalat szeretnénk húzni, azaz minden sorban ki szeretnénk tenni a függőleges vonalszakaszt. Már láttuk, hogy ezt legegyszerűbben úgy tehetjük meg, hogy a programra bízzuk, azaz a mintasorban is megadjuk.

Egyetlen dologra kell figyelnünk: az egyes sorok mellé húzott vonal olyan magas legyen, mint a sor, hogy folyamatos függőleges vonalat kapjunk. Ezt általában a program is elvégzi, de minden esetben rakjuk ki a \strut parancsot a biztonság kedvéért. Ez nem más, mint a

```
\hbox{\vrule height8.5pt depth3.5pt width 0pt}
```

rövidítése. Mint kiolvasható, az általa megadott doboz szélessége nulla, így az nem látható, de a magassága és mélysége által egy állandó magasságot biztosít az egyes soroknak. Elegendő, ha a fejrészben egyszer kiadjuk, annak hatására minden sorba belekerül.

A \strut parancs azonban arra az esetre van tervezve, ha tízpontos betűnagyságot használunk. Ez a könyv viszont tizenegy pontos betűnagysággal lett szedve, tehát kicsit nagyobb láthatatlan vonalakat használunk a következő táblázatban. Mindezek ismeretében a következő módon keretezhetjük be a példánkat (felhasználva a már korábban ismertetett vízszintes vonalakat):

```
\vbox{\halign{\strut\vrule#&\hfil#\hfil&\vrule#&\quad
#mm$\times$&#mm&\vrule#\cr
\noalign{\hrule}
\vrule width0mm height11pt depth4pt&Az oldal alakja&&
\multispan2\hfil Szedéstükör\hfil&\cr
\noalign{\hrule}
\vrule width0mm height11pt&A4&&172&243&\cr&A5&&108&162&\cr
&B5&&126&189&\cr
\noalign{\hrule}}}
```

Az oldal alakja	Szedéstükör
A4	172mm×243mm
A5	108mm×162mm
B5	126mm×189mm

Figyeljük meg, hogy a mintasorban hogyan definiáltuk a \vrule parancsokat. Ezek után a program automatikusan kiteszi azokat minden sorban, csak arra kell vigyáznunk, hogy ne felejtsük el a megfelelő & jeleket használni.

A szakasz befejezéseként a táblázat egy még bonyolultabb formáját és annak előállítását közöljük, további példaként. Látható, hogy egy olyan vízszintes vonal meghúzása, amely nem terjed a táblázat teljes szélességében, mekkora erőfeszítést kíván. Ezt már csak a TEX-ben járatos felhasználóknak ajánljuk.

```
\tabskip=0pt
\halign{\vrule#&\quad\hfil#\hfil\quad&\vrule#&\quad
\hfil#mm\hfil\quad&\vrule#&\quad\hfil#mm\hfil\quad&\vrule#\cr
\noalign{\hrule}
\vrule width0pt height11pt depth3pt &Az oldal&&\multispan3
\lower3pt\vbox{\hbox to 52mm{\hfill
Szedéstükör\hfill}\vskip3pt\hrule}&\cr
\vrule width0mm height10pt depth4pt&alakja&&\omit\quad\hfil
szélessége\hfil\quad&&\omit\quad\hfil magassága\hfil\quad&\cr
\noalign{\hrule}
\vrule width0mm height10pt depth3pt&A4&&172&&243&\cr
\noalign{\hrule}
\vrule width0mm height10pt depth3pt&A5&&108&&162&\cr
\noalign{\hrule}
\vrule width0mm height10pt depth3pt&B5&&126&&189&\cr
\noalign{\hrule}}
```

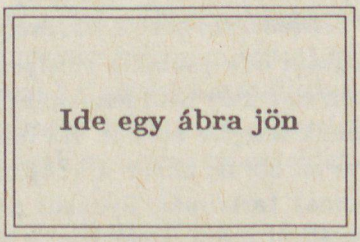
Az oldal alakja	Szedéstükör	
	szélessége	magassága
A4	172mm	243mm
A5	108mm	162mm
B5	126mm	189mm

6.3. Ábrák

Az ábrák számára kihagyandó hellyel több probléma is felmerülhet. A legfontosabb, hogy egy ábrát nem lehet több oldalra tördelni, így a programnak nem pontosan ott kell kihagynia az ábra helyét, ahol megadtuk, hanem az

ahhoz legközelebbi, megfelelő helyen. Mint majd látjuk, megadhatunk olyan ábrákat, amelyeket a kiadásukhoz közel bárhol az oldalon, vagy csak oldal tetején, vagy pedig egy egész oldalra kívánunk kiszedetni. A program ennek megfelelően kikeresi az első olyan helyet, ami a parancs után áll és megfelel a kívánalmaknak. Az első esetben például csak az akadályozhatja meg, hogy a program a parancs által adott helyre szedje ki az ábrát, ha onnantól számítva az ábra vége már egy következő lapra esik. Ha az oldal tetejére szeretnénk helyezni az ábrát, a program azt azon oldal tetejére helyezi, amelyiken a parancsot kiadtuk. Ettől csak akkor tér el, ha az oldalon már előzőleg is kiadtunk ilyen parancsot, vagyis már egy ábra elfoglalta az oldal tetejét. Ha ezek után már nem marad elegendő hely az ábrának, akkor a következő oldalra kerül. Végül, ha teljes oldalra szedett ábrát akarunk létrehozni, azt a program a parancs utáni első teljes oldalra szedi.

Tulajdonképpen nincs sok értelme, hogy azt tárgyaljuk, a program miként viselkedik a fenti parancsokra. Fogadjuk el, hogy megérti, mit szeretnénk, és annak megfelelően a legjobb megoldást választja az ábra elhelyezésére. Nézzük meg inkább, hogyan is hozzuk létre az ábrákat. Az eddigiekben hol az ábra létrehozásáról, hol csak az annak szükséges hely kihagyásáról beszéltünk. Ezt azért tettük, mert a kettő között nincsen lényegi különbség. Amennyiben az ábra olyan anyagot tartalmaz, amit a TEX programmal is létre tudunk hozni, akkor elkészíthetjük, amennyiben viszont nem, akkor a méretének megfelelő üres helyet kell kihagyni. Pontosabban ekkor a program szempontjából az ábra üres lesz.



6.1. ábra. Ábraaláírás

Mielőtt az egyes konkrét módozatokra térnénk rá, még tisztáznunk kell, hogy a program az itt leírt (egyszerű) parancsok segítségével csak olyan ábrákat képes létrehozni, amelyek szélessége megegyezik az oldal szélességével. Természetesen itt nem arra az esetre gondolunk, amikor két vagy több ábra kerül egymás mellé, amelyek összes szélessége már megegyezik a szövegével (hiszen akkor a program szempontjából egyetlen, teljes szélességű ábrával is számolhatunk), hanem arra, az itt is szemléltetett esetre, amikor a keskenyebb ábra

mellé kívánjuk a szöveget tördelni. Az ilyen bonyolult feladat megoldására csak az vállalkozzon, aki már nagyon érti a TEX egyéb részleteit. Általános szabályként azt kövessük, hogy ha egy szövegben (könyvben, cikkben, stb.) mindössze egy-kettő ilyen „nem normális" ábra szerepelne, az azok létrehozásához szükséges fáradság nem fizetődik ki, inkább hagyjuk el illetve módosítsuk azokat. Amennyiben viszont sok ilyen típusú, hasonló elrendezésű ábrára van szükségünk, tényleg kérjük ki valamely jobban hozzáértő személy tanácsát.

Ennyi bevezetés után térjünk rá a konkrét parancsokra. Amennyiben egy, az oldalon bárhova elhelyezthető ábrát szeretnénk létrehozni, azt a `\midinsert` és `\endinsert` parancsok közé kell zárnunk. Például a

```
\midinsert \vskip6cm
\centerline{\it 6.1. ábra. Ábraaláírás}
\endinsert
```

parancs egy 6 centiméternyi magas helyet hagy ki az ábra részére és az alá kiszedi középre a *„6.1. ábra. Ábraaláírás"* felíratot. Mint megfigyelhetjük, nem kell dobozokat létrehoznunk, sőt nem is tanácsos, mivel azt a `\midinsert` parancs magától megteszi. Hasonlóan, a program maga gondoskodik az előtte és utánna kihagyandó extra helyről, így a benne megadott 6 centiméter az ábra valódi magasságát jelöli, és utána nem kell megadnunk egy `\vskip ...` parancsot. A `\midinsert` és `\endinsert` parancsok egy környezetet határoznak meg, így a közöttük kiadott `\it` parancs rajtuk kívül már nem érvényes. *Vigyázzunk arra, hogy a* `\midinsert` *és a még eztán következő egyéb hasonló parancsokat csak két bekezdés* között *használjuk.*

A fenti parancs hatására tehát a program oda próbálja elhelyezni az ábrát, ahol a parancsot találja. Amennyiben ez a hely nem lenne megfelelő, akkor azt átviszi a következő oldalra és annak tetejére helyezi. Ha mindenképpen egy oldal tetejére szeretnénk tenni az ábrát, akkor a `\topinsert ... \endinsert` parancsok közé zárjuk annak tartalmát. Ilyenkor a program először azon oldal tetejére igyekszik az ábrát tenni, amelyiken a parancsot találta, majd ha ez nem sikerül, a következő oldalra. Ha viszont egy teljes oldalra szánunk egy ábrát, akkor azt a `\pageinsert ... \endinsert` parancspárral adjuk meg. Ilyenkor a tartalmát a program a következő oldalra helyezi.

A következő tudnivalókat jegyezzük meg ezekről a parancsokról:

- Az ábrát kijelölő két parancs közé tetszőleges, akár több bekezdést felölelő szöveg is kerülhet. Ugyancsak alkalmazhatjuk a máshol használatos parancsokat, amelyek érvényessége csak az itt meghatározott környezetre korlátozódik.
- A parancsokat soha ne adjuk ki egy bekezdésen belül, mindig csak két bekezdés között.
- Semelyik ábra sem tartalmazhat több anyagot, mint amennyi egy oldalra fér, hiszen az a legnagyobb egység, ahova a program el tudja helyezni azokat. Ellenkező esetben a program a már ismert `Overfull \vbox` üzenettel válaszol.
- Ha néhány ábrát meghatározó parancs túl közel kerül egymáshoz, elképzelhető, hogy egy-kettő csak a következő oldalak egyikére fér el. A program ilyen esetben is igyekszik a legjobb elhelyezést megtalálni számukra, de mindig csak olyan sorrendben szedi ki őket, amilyen sorrendben az őket meghatározó parancsok következnek.
- Amennyiben túl sok ábrát használunk egy szövegben, és azok túlságosan nagy méretűek, vagy a felhasználásukról túl sok megkötést teszünk (azaz a `\midinsert` helyett a `\topinsert` vagy `\pageinsert` parancsokat használjuk megadásukra), akkor a program nem talál a megadásukhoz közel megfelelő helyet számukra, és azok meglehetősen távol kerülnek a hivatkozási helyüktől. Ilyen esetben, ahelyett, hogy a programot tennénk felelőssé a hibáért, gondoljuk végig, lehetséges-e az általunk támasztott kívánalmak kielégítése. Amennyiben nem, próbáljunk az ábráknak más elhelyezést adni.
- Az előző eset következménye lehet az is, hogy a szöveg befejezésekor a program még nem tudott minden ábrát elhelyezni. Ilyenkor, ha csak az `end` parancsot használtuk a file lezárására, kimaradhatnak ábrák. Ha viszont a `\bye` paranccsal fejeztük be a file-t, a még ki nem szedett ábrák az utolsó szövegoldal után folyamatosan kiszedésre kerülnek.

6.4. Lábjegyzetek

A lábjegyzetek azért is kerültek ebbe a fejezetbe, mert a TEX gyakorlatilag úgy kezeli őket, mint valami speciális ábrát. Az egy oldalon összegyűlő

lábjegyzeteket az oldal végén helyezi el egy, a `\topinsert`-hez hasonló parancs segítségével. Ehhez azonban egy speciális algoritmus szükséges. Ebben a könyvben is többször előfordul, hogy a lábjegyzetnek csak úgy lehet megfelelő helyet találni, ha két oldalon keresztül tart. Ennek oka egy olyan, az oldal aljához közel kiadott `\footnote` parancs, amelyikben a lábjegyzet tartalma olyan hosszú, hogy nem fér ki az oldal aljára. Szerencsére ezzel a problémával nem kell törődnünk, a program automatikusan megoldja.

A lábjegyzetet a már ismert `footnote` paranccsal hozhatjuk létre. Ne felejtsük el, hogy ha számozni szeretnénk őket, akkor az első változóban a kitevőbe kell tenni a számot. Tehát ekkor a `\footnote{$^2$}{...}` mintájára kell eljárnunk. Természetesen használhatjuk a lábjegyzeteknél szokásos †, ‡, vagy § jeleket is, ilyenkor nem szükséges matamatikai módot alkalmazni a lábjegyzetet előállító parancs első változójában. A második változó, a lábjegyzet szövege szinte minden, a TEX által ismert dolgot, parancsot tartalmazhat. Így lehet benne matematikai képlet, `\centerline` parancs, több bekezdés egymás után, stb. Az egyetelen dolog, ami nem szerepelhet itt, az előző szakaszban leírt, ábrát definiáló `insert` típusú parancs.

Ha megfigyeljük a hátsó borítón található lábjegyzetet és a könyvben alkalmazott lábjegyzeteket, láthatjuk, hogy lényeges különbség van köztük. Igaz, hogy a könyv teljes egészében az `mplain` felhasználásával készült, de néhány parancsot — köztük a lábjegyzetet előállító `\footnote`-ot — felülírtunk benne. A borító viszont tényleg csak az `mplain`-t használja, így az ott található lábjegyzet stílusa megegyezik azzal a stílussal, amit az Olvasó a program komolyabb átírása nélkül előállíthat. Tehát a lábjegyzet a szövegben használt betűnagyságot használja, egy rövid vonallal lesz elválasztva a szövegtől, és az első bekezdése előtt tartalmazza az azonosító jelét. A könyvben használt lábjegyzet lényegében csak a betűnagyságban tér el ettől.

Egyetlen fontos paraméter van, amelynek beállításával meg kell ismerkednünk a lábjegyzettel kapcsolatban. A `\dimen\footins` mérték[4] adja meg azt, hogy mekkora lehet legfeljebb az a rész az oldalon, amelyet a lábjegyzetek foglalnak el. Az alapértelmezés szerinti beállítása 8 inch, azaz gyakorlatilag a teljes oldal. Ez a könyv viszont úgy készült, hogy 20pc-re állítottuk

[4] Ne törődjünk azzal, hogy ilyen fura módon van megadva. Kezeljük ugyanúgy, ahogy mondjuk a `\hsize` mértéket.

be az értékét. Megfigyelhetjük, hogy egyetlen oldalon sem foglalnak el a lábjegyzetek ennél több helyet. Ha szükséges, a program inkább a következő oldal aljára tördeli a később következő lábjegyzeteket.

7. fejezet

Új utasítások definiálása

7.1. Mik azok a makrók?

Már a könyv eddigi fejezeteiben is többször utaltunk arra, hogy a TEX program segítségével új utasításokat, úgynevezett „makrókat" is lehet definiálni, akárcsak sok más, nem szedésre használt programnyelvben. Ez a lehetőség nagyon nagy mértékben megkönnyítheti a munkát bizonyos esetekben, és néha szinte pótolhatatlan. Gondoljunk például arra, hogy egy színház havi programjának a szedésekor ugyanazt a 4–5 címet kell jó néhányszor leírni. Mennyivel megkönnyíti ilyenkor a munkát, ha mondjuk az `\emb` parancs hatására a program automatikusan kiszedi: „Madách: Az ember tragédiája".

Hasonlóan sokat segíthet, ha egy újság szedésére használva a programot az újság alapvető külalakját előre megírjuk és egy vagy több utasításként definiáljuk. Ilyenkor ezeket egy úgynevezett makró file-ba gyűjthetjük, aminek mondjuk a `makr.tex` nevet adva azt minden újság szedésekor az

```
\input makr
```

utasítással beolvashatjuk. Így előzőleg definiálhatunk és a `makr.tex` file-ba helyezhetünk egy `\cim` utasítást, amely mondjuk az évfolyam számát és azon belül a lapszámot kéri változóként és ezek ismeretében kinyomtatja az újság fejlécét. Ennek birtokában, a `makr.tex` file beolvasása után a `cim` parancs automatikusan kinyomtatja a címet, mivel az `\input` hatására a TEX beolvasta és „megértette" a `makr.tex`-ben tárolt, általunk korábban definiált utasításokat.

Természetesen vigyázni kell a makrók definiálásával, hiszen a számítógép memóriája véges, és így csak véges sok utasítást lehet előre definiálni. Ugyancsak meggondolandó, hogy ezeknek a file-oknak a beolvasása sok időt vehet igénybe, túl nagy és túl általános makrócsomagok megírása esetleg nagyobb késést okoz, mint amennyi hasznot. Persze ezektől a lehetőségektől nem szabad igazán megijedni. Aki már annyira tud a TEX-hel bánni, hogy veszélyesen nagy makrócsomagokat képes írni, az azt is tudja, mekkorák a túl nagyok és hogyan lehet elkerülni őket. Tekintsük példaként a TEX `plain` alakját, pontosabban annak az `mplain` , magyarosított változatát. Ez szintén egy makrócsomag, amely az egyébként eléggé „gyermekcipőben" járó TEX-et megtanítja a legalapvetőbb dolgokra, és amely a TEX körülbelül háromszáz alapvető (úgynevezett primitív) utasításával szemben nagyjából hatszáz makró definícióját tartalmazza. Ez az igen nagy méretű makócsomag is még gyorsan betöltődik a gép memóriájába[1].

Aki jobban odafigyel, észreveheti, hogy a fenti bekezdésben egy kicsit csaltunk. Az `mplain` ugyanis, legalábbis a TEX szerint, nem egy makrócsomag, hanem egy úgynevezett `format`, amit mi magyarul az előző bekezdésekben formátumként emlegettünk. A két dolog közötti különbség nem nagy, nem is pontosan definiálható. Mindössze arról van szó, hogy a nagyobb, egységes rendszerbe foglalt makrócsomagokat szokás formátumoknak is nevezni. Lényegesebb különbség, hogy a TEX rendelkezik egy speciális algoritmussal, amelynek a segítségével ezeket a nagy makrócsomagokat speciális módon tárolja (úgynevezett formátum, vagy `fmt` file-okban) és a szokásosnál sokkal nagyobb sebességgel tölti be. Enélkül az algoritmus nélkül a `plain` vagy `mplain` formátum betöltése sokkal több időt venne igénybe. Azon kívül, hogy időt takarítunk meg vele, van még néhány egyéb, nélkülözhetetlen előnye is a formátumoknak. Talán a legfontosabb, hogy a program elválasztási algoritmusát csak ezekben lehet megadni, illetve megváltoztatni. Mivel pedig elválasztás nélkül nem működik a program, minden futáskor legalább egy — és a zavarok elkerülése végett csak egy — formátumnak jelen kell lennie.

[1] Erről mindenki meggyőződhet, hiszen a TEX minden egyes futtatásakor az `mplain` formátum is betöltődik, ahogy azt üzenet is jelzi.

7.2. A legegyszerűbb definíciók

Ebben a fejezetben azonban nem a makrócsomagok ismertetése, hanem az új utasítások definiálásának a leírása a célunk. Az előbbit az itteninél sokkal részletesebben megtalálhatjuk a 9. fejezetben (Makrócsomagok), illetve a Függelék „TEX programok a gyakorlatban" című részében. Térjünk tehát vissza az első példánkhoz: Az ember tragédiájához. A

```
\def\emb{Madách: Az ember tragédiája}
```

utasítás hatására a program megteszi azt, amit szeretnénk, azaz valahányszor egy \emb parancsot talál az inputban, azt „Madách: Az ember tragédiája"-ként szedi ki. Ez azt jelenti, hogy az \emb parancs innentől kezdve ugyanúgy szerepel a TEX memóriájában, mint mondjuk a \ant vagy \kelt, de természetesen csak abban az inputban, amelyben a fenti definíció szerepel, és csak a definíció után.

A program a definíciók elfogadásakor úgy viselkedik, ahogy azt elvárnánk tőle. Egy parancs definícióját attól a pillanattól kezdve ismeri, hogy azt megadtuk neki. Tehát az

```
\emb
\def\emb{Madách: Az ember tragédiája}
```

részlet teljesen értelmetlen a TEX számára, mivel az \emb parancsot azelőtt adtuk ki, hogy definiáltuk volna. Helyesen így kell eljárnunk:

```
\def\emb{Madách: Az ember tragédiája}
\emb
```

Egy már definiált parancsot minden további nélkül felülírhatunk, azaz új definíciót adhatunk egy korábban már használt parancsnévnek. Ilyenkor a második definíció pillanatáig az első által adott értelmezés van érvényben, onnantól kezdve pedig a másodiké. A

```
\def\emb{Madách: Az ember tragédiája}    \emb
\def\emb{Emberek a havason}    \emb
```

részletben az első \emb parancs kiírja, hogy „Madách: Az ember tragédiája", míg a második, hogy „Emberek a havason".

Ha megnézzük a plain.tex vagy mplain.tex file-t, láthatjuk hogy ezek jobbára nem mások, mint a fenti \def típusú utasítások rengeteg egymás utáni — és pillanatnyilag számunkra még jó részt érthetetlen — alkalmazásai. Például a \normalbaselines utasítás, amely beállítja az alapértelmezés szerinti sortávolságot a következőképp van definiálva:

```
\def\normalbaselines{\baselineskip\normalbaselineskip
\lineskip\normallineskip \lineskiplimit\normallineskiplimit}
```

Itt a kapcsos zárójelen belül szereplő normal-lal kezdődő utasítások előre beállított távolságértékek, így a definíció értelme az, hogy a lineskip, baselineskip és lineskiplimit változókat beállítja az előre adott értékekre.

7.2.1. A definíciók megadása

Ideje megismerkednünk a makró-definiálás pontos szabályaival, noha mostanra mindenkinek vannak már elképzelései az előző példák alapján. Tehát először megadjuk a \def parancsot, ami természetesen a definíció rövidítése. Ez adja meg, hogy utána egy új parancs definíciója következik. Aztán rögtön megadjuk az új parancs általunk választott nevét, a TEX szokását követve a \ karakterrel bevezetve. Ez a név bármi lehet, ami megfelel a programnak, mint utasításnév, akár egy előzőleg definiált név, vagy pedig egy eredeti TEX utasítás neve is. *Ezzel azonban nagyon vigyázzunk, ilyenkor az utasítás eredeti értelmezése egyszer és mindenkorra elvész.* Ezt persze — szerencsére — úgy kell érteni, hogy az éppen használt input file-ban vész el az értelmezés. Ha mégis szükségünk lenne rá, akkor az inputból a hibás definíciót kiiktatva visszanyerjük az eredeti utasítást. A következő fejezetben megtaláljuk a \show parancs használatát, amely abban segít, hogy megtudjuk, az új utasítás nevének kiszemelt jelsorozatot használja-e már a program mint egy másik parancs nevét.

Miután megadtuk az új utasítás általunk választott nevét, már csak az értelmezését kell megadnunk, amit *mindig* kapcsos zárójelek közé zárunk. Ez nem jelenti azt, hogy az értelmezésben nem használhatunk kapcsos zárójeleket, mindössze annyi megkötést ad, hogy azoknak, mint egyébként a

programban általában is, megfelelően kell egymásba ágyazódniuk, és mindnek meg kell hogy legyen a párja a legkülső, az értelmezést behatároló kapcsos zárójeleken belül. Az értelmezés az a szövegrészlet vagy TEX utasítás, illetve ezeknek egy sorozata, amit a TEX a parancs kiadásakor annak helyén a szövegbe illeszt, illetve ott végrehajt. Így az `\emb` parancs hatására nem utasítás kerül végrehajtásra, csak a megadott szövegrészletet kerül az outputba pontosan arra a helyre, ahova a program az `\emb` parancs helyén álló szövegrészletet illesztené. Ezzel szemben a `\normalbaselines` parancs hatására az általa kijelölt három értékadás történik meg. Összességében mindkét utasítás ugyanazt eredményezi, mintha helyettük a definíciójukban szereplő, a kapcsos zárójelben kiadott karaktersorozat szerepelt volna az inputban.

A `\def` parancs ugyanúgy működik, mint bármely másik TEX parancs. Így egy definíciót akár egy szón belül is megadhatunk, mint a következő példában:

```
\def\emb{Madách: Az ember tragédiája}gédia
```

Ennek ellenére semmi sem szól amellett, hogy így tegyünk, és mindenképp sokkal áttekinthetőbb, ha egy-egy definíciót mindig új sorban adunk meg. *Az egyetlen dolog, amire nagyon kell vigyáznunk, hogy egy parancs definíciójában ne használjuk fel a parancs nevét,* mint az az alábbi példában történik:

```
\def\emb{\emb\ Az ember tragédiája}
```

ugyanis ez valóban tragédiához vezethet (legalábbis a szegény program számára).

A definíció azon tulajdonsága, hogy már egyszer definiált parancsneveket minden további nélkül felhasználhatunk újra kellemetlenségeket okozhat, de ezek kis figyelemmel elkerülhetők. Ezzel szemben néha kifejezetten hasznos lehet ez a tulajdonság. Segítségével ugyanis elérhetjük, hogy azonos jellegű dolgoknak ugyanazt a nevet adjuk, de az inputban szerepük esetleges módosulásának megfelelően változtassuk a definíciójukat. Például a `plain` formátumban az előző fejezetben említett `\narrower` utasítás a következő módon van definiálva.[2]

[2] Mint látható, sok, eddig már említett utasítás is makró, nem pedig alapvető, tovább már nem bontható TEX utasítás. Ránézésre nincsen rá mód, hogy egy utasításról

```
\def\narrower{\advance\leftskip by \parindent
\advance\rightskip by \parindent}
```
[3]

Mint azt már leírtuk — vagy a definícióból is kiolvashatjuk — ezen parancs hatására a kiszedett sorok a szokásosnál beljebb kezdődnek és előbb végződnek az oldalon.

Ha valamely okból egy könyv szedése közben ennek a mértékét meg szeretnénk változtatni, nem kell feladnunk a `\narrower` parancs használatát, csak újra kell definiálnunk, mondjuk úgy, hogy

```
\def\narrower{\advance\leftskip by 2\parindent
\advance\rightskip by 2\parindent}
```

Itt a 2-es szorzó a `2\parindent` kifejezésekben azt eredményezi, hogy innentől kezdve a sorokat az eddigi mérték kétszeresével kezdi beljebb, és ugyanannyival előbb fejezi be a program ugyanannak a `\narrower` parancsnak a hatására.

7.2.2. Az utasítás neve

A `\def` paranccsal definiált új utasításnevek ugyanazoknak a szabályoknak kell hogy eleget tegyenek, mint a TEX eredeti primitív, illetve a `plain` formátum makró utasításainak nevei. Ezt a szabályt ugyan már egyszer ismertettük, de itt az új helyzetre való tekintettel — mivel már mi is adhatunk utasításneveket, jobban kell figyelnünk azok formai követelményeire is — még egyszer, teljes pontossággal leírjuk. A parancsneveknek két fajtája van, mindkettő a \ karakterrel kezdődik. Az egyik esetben a \ jelet egyetlen más jel követi, ami nem betű. Ilyen esetben a program automatikusan

---

eldöntsük, primitív-e vagy pedig makró, de erre általában nincs is szükség. A definiálása pillanatától kezdve egy makró pontosan olyan szerephez jut, mint bármelyik primitív utasítás.

[3] Az eredeti definícióban a by szavak nem szerepelnek, mivel nélkülük is használható az `\advance` parancs. Itt csak azért használjuk, mert az előző fejezetekben így tanultuk a parancs használatát. Valójában mind a két út helyes.

tudja, hogy a parancs neve abból az egyetlen jelből áll, és így a parancs nevét akár közvetlenül is követheti bármely más jel. A másik esetben a \ jelet tetszőleges hosszú, betűkből álló sorozat követi, de ilyenkor a parancs neve után közvetlenül nem állhat egy másik betű, hiszen akkor a TEX nem venné észre, hol van vége a parancsnak, és hol kezdődik az azt követő szöveg. Ha a parancs neve utáni első jel egy betű lenne az inputban, akkor azt egy szóközzel kell megelőzni, amit viszont a program nem vesz figyelembe. Az eddigiek szerint tehát a `\?`, `\$`, `\3`, `\naezmi`, `\valami` mind megfelelő parancsnevek, viszont a `\34`, `naezmi?`, `\va,lami` nem használhatók parancsok neveiként.

MATEX Talán már feltűnt, hogy nem használtunk sem példaként, sem valódi utasítások neveiként olyan szavakat, amelyek magyar ékezetes betűket is tartalmaznak. Ennek oka, hogy noha a felhasználó számára ezek a speciálisan magyar betűk egyébként teljesen az angol társaikkal megegyezően viselkednek, *a program nem tekinti őket betűknek.* Még pontosabban megfogalmazva, a `plain` formátum számára nem lehetnek betűk, hiszen ekkor éppen úgy kell őket megadni, mint ahogy a 2. fejezetben leírt összes többi (spanyol, német, stb.) ékezetes betűt, és egy `\'a` sorozat nyilván nem lehet betű. Az `mplain` formátum által megadott MATEX-ben pedig ugyan minden szempontból teljes jogú betűként viselkednek egyébként, *a programban elfoglalt szerepük szerint mégsem betűk. A legbiztosabb megoldás tehát, ha magyar ékezetes betűket nem használunk az általunk definiált parancsok neveiben* (az előre definiált parancsok esetén sem találunk ilyet). A program csak az angol ábécé betűit kezeli betűkként, és így csak azokat fogadja el a parancsok nevében. Azok, akik nem ismerik az angol ábécét az angol betűket legbiztosabban úgy ismerhetik fel, hogy annak veszik az egytagú, ékezetek nélküli betűket (és amennyiben valaki ékezetes betűknek tekinti az *i* és *j* betűt, akkor még azokat is). Ezek egyébként megegyeznek a legtöbb Magyarországon forgalmazott mikroszámítógép billentyűzetén található gyárilag felfestett betűkkel.

Milyen legyen tehát egy újonnan definiált parancs helyes neve? Álljon csupa, az előző bekezdésben leírt betűből, és ha lehet valamely módon emlékeztessen a parancs funkciójára, hogy könnyebb legyen megjegyezni. Ennél többet nem érdemes mondani róla, hiszen mindenki maga dönti el, milyen forma marad meg a legjobban a memóriájában. Habár ez a könyv nem ismerteti az összes, a `plain` és az `mplain` formátumban használt TEX parancsot, és így előfordulhat (de eléggé valószínűtlen), hogy valamelyiket nem ismer-

vén annak nevét használjuk egy általunk definiált parancs neveként, ez nem okoz problémát. Attól a pillanattól kezdve, hogy definiáltuk a parancsot, az már az általunk előírt módon fog viselkedni. A nem ismert parancs neve pedig nem hiányozhat, hiszen ismeretlenül nem is használhattuk. Az egyetlen kivétel, ha az egy le nem írt parancs ugyan, de valamely általunk ismertetett, a `plain` vagy `mplain` formátumban található makró használja. Így a makró használatakor mi is — tudtunkon kívül — felhasználjuk. Ennek valószínűsége ugyan nagyon kicsi, de ha mégis előfordul, komoly problémákat okozhat. Többek között ennek elkerülését is szolgálja, ha a parancsnevek emlékeztetnek a parancs funkciójára, és így „magyarosan" hangzanak, szemben az eredeti angol nevekkel. Ha kétségeink vannak egy parancs nevét illetően, ellenőrizzük azt a következő fejezetben ismertetett `\show` parancs segítségével.

Természetesen előfordulhat olyan helyzet, amikor egyetlen karakter jobban emlékeztet egy parancs funkciójára, mint egy szó; nem beszélve arról, hogy egyszerűbb legépelni is. Például ha olyan könyvet szedünk, amiben rengeteg párbeszéd van, amelyek kezdő sorát beljebb kezdjük és egy gondolatjellel is jelezzük, ugyanakkor nem akarunk rengeteg üres sort hagyni az inputban, hasznos lehet a

```
\def\-{\par ---}
```

definíció megadása, hiszen segítségével csak a `\-` karaktersorozatot kell minden párbeszéd elején megadni. Ugyanakkor nem szabad elfelejtenünk, hogy ilyen parancsnévből csak rendkívül korlátozott számú áll a rendelkezésünkre (szemben a parancsszavak gyakorlatilag kimeríthetetlen készletével), és azok legnagyobb részét is felhasználta már előttünk valamire a TEX. Így a `\-` például a lehetséges elválasztási helyek kijelölésére szolgál, és a használata egy új parancs neveként a továbbiakban megfoszt minket ettől a lehetőségtől.

MATEX Ha valakinek már meglehetősen nagy programozási gyakorlata van és valamiért elengedhetetlenül szükségesnek érzi, hogy magyar ékezetes betűket is használjon az általa definiált új parancsok neveiben, akkor ebből a bekezdésből megtudhatja az erre vonatkozó szabályokat. Mindenesetre a szabályok megadása ellenére is fenntartjuk azt a véleményünket, hogy ennek sok haszna nem lehet, viszont annál több kár származhat belőle. Természetesen mindaz, ami itt következik, csak a MATEX verzióra érvényes. Tehát a szabály: a program azon parancsneveket, ame-

lyekben magyar ékezetes betűk is szerepelnek két részre osztja. Az első csoportba tartoznak azok, amelyekben mindjárt az első betű ilyen. Ekkor, mivel a TEX nem tekinti ezt betűnek, nem parancsszót definiáltunk, hanem csak egy parancsjelet. Ilyenkor a program már az első betűjéből is azonosítja a parancsot. Nem tekinti azonban hibának, ha a definícióban még egyéb jelek is követik az első betűt, amennyiben a parancs minden felhasználásakor is használjuk azokat. Ha ilyenkor még egy parancsot definálunk, aminek a neve ugyanazzal a magyar ékezetes betűvel kezdődik, akkor onnantól kezdve már ez lesz a parancsjel jelentése, és az előző parancs használata hibajelzést eredményez.

Talán jobban érthetők a szabályok, ha példával illusztráljuk őket. A

```
\def\ékez{á,é,í,ó,ö,ő,ú,ü,ű}
```

definíciót használva az \ékez parancs hatására a MTEX kinyomtatja a magyar ékezetes kisbetűket. Ilyenkor, noha a program a parancsot már annak \é kezdetéből azonosította, ha csak ennyit adunk meg neki, hibajelzést ad. Csak a definiált módon, azaz \ékez-ként szabad meghívni a parancsot. Tegyük fel, hogy később szükségünk van egy másik utasításra, amit mondjuk így definiálunk:

```
\def\év{az 1980-as években}
```

Természetesen ezt is azonosítja a program már az \é részéből, így a defínició tulajdonképpen nem egy teljesen új parancs definiálása, hanem a már létező \é parancs újradefiniálása. Így ha most adjuk ki a \ékez parancsot, akkor azt a TEX a \é első részéből az év paranccsal azonosítja, és mivel nem az előírt módon fejezzük be, hibaüzenetet ad.

A magyar ékezetes betűket is tartalmazó parancsnevek második csoportja azon nevekből áll, amelyeknek az első betűje még az angol ábécében is szerepel, és csak a továbbiakban tartalmaz ékezetes betűt. Talán már sejtjük, hogy ilyenkor a program a parancsot már abból a részéből azonosítja, amely az első ékezetes betűt megelőzi. Így a \szélesség és \oldalszám parancsneveket már a \sz illetve \oldalsz kezdeteikből megérti a program, de ebben az esetben is azt várja, hogy a megadott módon fejezzük be őket, ellenkező esetben hibaüzenettel reagál. Ilyen esetekben is fennáll annak a veszélye, hogy ha később egy \szám utasítást definiálunk, akkor attól a pillanattól a \sz kezdetből már ezt találja meg

a program, és így a `\szélesség` utasítást többet nem használhatjuk. A TEX tehát egy kicsit ahhoz a programozási nyelvekhez hasonlít, amelyek a változóneveket azok néhány első karakteréből azonosítják, és így a gyakorlatlan programozó számára néha megoldhatatlan rejtélyt jelentően keverik össze a változókat.

7.3. Változók megadása

Az idáig megadott példákban ugyan elég hasznosak voltak az újonnan definiált utasítások, de azért igazából nem okozott volna nagy gondot, ha nem tudjuk használni őket. Az igazi hasznuk akkor jelentkezik, amikor változókkal rendelkező új parancsokat is tudunk definiálni. Például készíthetünk olyan utasítást, amelyik középre zárja ki a változójában megadott szövegrészletet a következő minta szerint:

```
\def\kzp#1{\line{\hfil#1\hfil}}
```

A definícióban a #1 a makró első paraméterét jelöli. Ha viszont az utasítást a

```
\kzp{Ezt zárjuk ki középre.}
```

mintájára kiadjuk a szövegben, akkor a `Ezt zárjuk ki középre` részt már az utasítás változójának nevezzük. Tehát ha a paramétert a definícióban szereplő, a parancsot helyettesítő részletben valahol megadjuk (akár több esetben is), akkor a parancs használatakor a változó oda kerül, ahol a paraméter jele áll.

Az így definiált parancs változója ugyanúgy viselkedik, mint a már megismert változók. Ha nem zárunk semmit kapcsos zárójelek közé a parancs után, akkor az első, nem a szóközzel megegyező jel lesz a változó. Ha viszont ez a { jel, akkor a változó az ez utáni első karaktertől terjed az ezen nyitó zárójelnek megfelelő záró kapcsos zárójelig, azt már nem beleértve. Tehát a következő négy példában:`\kzp{\bf alma}` `\kzp alma` `\kzp{{\bf alma}}` `\kzp{{al}ma}}` a változók rendre `\bf alma`, a, `{\bf alma}` illetve `{al}ma`. Figyeljük meg, hogy ha betű követi a parancsszót, ebben az esetben is ki kell hagyni egy szóközt. Egy másik fontos jelenséget is szemléltetnek a

példák. A harmadik példa esetében használt dupla zárójelezés nem felesleges, ugyanis az első példában a \bf parancs általános érvényű lesz, tehát a \kzp parancs változóján kívül az azt követő szövegre is érvényben marad, míg a harmadik példa esetében a második zárójelpár biztosítja a \bf parancs lokális értelemezését. A TEX minden esetben csak egyetlen zárójelpárt „fejt le" a megadott változó körül, a maradékot behelyettesíti a parancsba a paraméter helyére.

Egy parancsnak több paraméterét is definiálhatjuk, és így használatakor több változóját is megadhatjuk. A parancs paramétereit a definícióban a parancs neve után adjuk meg a #1, #2, ..., #9 karaktersorozatok felsorolásával. Ez máris megadja, hogy egy új utasításnak legfeljebb kilenc paramétere lehet. Ennél többet azonban nem is lenne tanácsos egy parancson belül felhasználni. A paraméterjelekből tehát tízig akárhány megadható, de csak nagyság szerinti sorrendben használhatjuk őket. Így a #3 jel egy definícióban csakis a #2 után szerepelhet. Az utasítás neve után szereplő, a kapcsos zárójelekben megadott részletben viszont, ami az utasítást hivatott helyettesíteni a meghívásakor, már akármilyen sorrendben megadhatjuk a definiált paramétereket, akár ki is hagyhatjuk némelyiket (akkor persze nem sok értelme volt a megadásuknak).

Az eddigiek alapján már tudjuk, hogy a TEX a parancsszavak utáni szóközt nem veszi figyelembe. Ezek szerint például a `\def\kzp#1...` és `\def\kzp #1...` definíciók megegyeznek. Vigyáznunk kell azonban a paraméterjelek közötti és utáni szóközökkel, ugyanis ezeknek fontos jelentőséget tulajdonit a program. Megint eltekintünk attól, hogy teljes részletességében és bonyolultságában ismertessük a szabályokat, csupán a lényegi pontokra térünk ki. Azok számára, akiknek nincs még megfelelő gyakorlata a TEX-ben való programozásban úgyis csak azt tudjuk ajánlani, kerüljék el a most következő rész buktatóit, és csak viszonylag egyszerű makrókat írjanak, *amelyekben a paraméterjelek szorosan követik egymást*, mint a következő példában:

```
\def\minta#1#2#3{\line{#1\hfil#2\hfil#3}}
```

(Gyakorlásképp fejtsük meg, hogy mit csinál ez a makró)

A továbbiakban talán a legegyszerűbb lesz, ha a plain TEX egy létező definícióját, a \proclaim utasítást tekintjük, amely (majdnem) a következő módon van definiálva:

```
\def\proclaim #1. #2\par{\medbreak
\noindent{\bf#1.\enspace}{\sl#2}\par}
```

Mint látható, ebben az esetben már szinte kusza módon követik a jelek egymást a definíció azon részén, amelyről idáig úgy tudtuk, hogy a paraméterjelek számára van fenntartva. Először is szögezzük le: az a szabály, miszerint a makró értelmezése a neve és a paraméterjelek listája után kapcsos zárójelek között következik mindenkor érvényes. Tehát a jelen definícióban a paraméterek listája a következőképp néz ki (ha eltekintünk a `proclaim` utáni, a program által is figyelmen kívül hagyott szóköztől): `#1. #2\par`. Valójában ez a lista még bonyolultabb is lehetne, csak arra kell vigyázni, hogy *ne tartalmazzon kapcsos zárójeleket,* és természetesen, hogy az első # jelet az 1, a másodikat a 2, stb. kövesse. Azután, amikor a megfelelő #1, #2, stb. jelsorozatok előfordulnak a makró definíciójában (a # jelet mindig egy számjegynek, illetve — itt le nem írt szabályok szerint — esetleg egy ugyanolyan # jelnek kell követnie), a program az adott helyekre helyettesíti a parancs végrehajtásakor a változókat. A paraméterlistában előforduló egyéb jelek pedig azt modják meg a TEX-nek, hogy milyen környezetben fogjuk használni a változókat.

A `\proclaim` parancs `#1. #2\par` paraméterlistája például azt jelenti, hogy a parancs első változóját mindig egy pont (.) és egy szóköz követi, majd pedig a második változó következik, amelyet egy `\par` parancs (vagy azzal ekvivalens üres sor) zár le. Ilyen esetekben a program nem veszi figyelembe, ha kapcsos zárójelek közé zárjuk az általunk kívánt változót. Ha a `\proclaim` parancsot így adjuk ki:

```
\proclaim {Ez az első változó} Ez a második változó.
Itt pedig már a folytatás következik ...
```

akkor — minden igyekezetünk ellenére — az első változó a

```
{Ez az első változó} Ez a második változó
```

szövegrészlet lesz, a második pedig az `Itt pedig` kezdetű résszel kezdődik, és egészen addig tart, amíg egy üres sor vagy `\par` parancs nem adódik. Figyeljük meg, hogy ebben az esetben az első változóban alkalmazott kapcsos zárójelek nem tüntek el a változóban, hiszen azok most nem a változó kijelölését szolgálják. Viszont azok a jelek, amelyek a definícióban a paraméterlistában a paraméterek egymástól való elhatárolását szolgálták —

esetünkben a .␣ sorozat, illetve a \par parancs — nem jelentkeznek egyik változóban sem, ilyenkor a TEX ezeket „nyeli le". Éppen ezért a jelek — ha alkalmazni szeretnénk őket — a definícióban újra megjelennek.

A fenti szabály alól egyetlen kivétel van: ha a kapcsos zárójelek (véletlenül) egy változó két oldalára kerülnek, de nem ők jelölik ki a változót, mint a következő példában:

```
\proclaim {Ez az első változó}. Ez ...
```

Ilyenkor a zárójelek mégsem jelennek meg a változóban. Ebben az esetben az első változó tehát pontosan a Ez az első változó szövegrészlet. A kapcsos zárójelek azon tulajdonsága, hogy egy szükebb környezetet hoznak létre, dolgokat érdekes hatással lehet a változókra. Ha a kapcsos zárójeleken belül fordul elő az a jelsorozat, ami a definícióban mint a változót határoló jelsorozat szerepel, akkor azt szinte „elrejti" a definíció elől. Például a \proclaim parancs

```
\proclaim {Ez az első változó. Innentől a második változó}
```

használata valótlant állít, hiszen hiába zárja le az általunk hitt első változót egy .␣ jelsorozat, azokat a közrezáró kapcsos zárójelek hatására a TEX nem veszi figyelembe. Az igazi első változó tartalmazni fogja mindazt, ami a zárójeleken belül található, a zárójelekkel együtt, és még az utánuk következő szövegből is annyit, ami az első, szabadon álló .␣ jelsorozatig következik.

Mit is csinál valójában tehát ez a proclaim utasítás? A pontos elemzéséhez idézzük fel mégegyszer a definícióját:

```
\def\proclaim #1. #2\par{\medbreak
\noindent{\bf#1.\enspace}{\sl#2}\par}
```

A szerzők szándéka szerint ez az utasítás tételek, definíciók, állítások és egyéb matematikai jellegű kiemelések szedésére szolgál. Első változója ennek a neve és — ha van — száma míg, a második a szövege. A TEX a matematikában szokásos módon a tétel nevét és számát félkövér betűkkel szedi ki, a lezáró pontot — amit a változók értelmezésekor „lenyelt" — visszahelyezi, egy \enspace-nyi kis helyet hagy ki, majd dőlt betűkből kiszedi a tétel szövegét. Végül, mivel a parancsot követő új bekezdést is eltüntette ideiglenesen, automatikusan új bekezdést kezd.

7.4. Egyéb definiáló utasítások

7.4.1. Utasítások definiálása más módon

A program eddigiekben leírt lehetőségei nagymértékben megkönnyíthetik feladataink megoldását rendkívül hatékony utasítások definiálásával, feltéve, hogy soha nem hibázunk és nem is felejtjük el megfelelő módon körülhatárolni a változókat. Mivel azonban emberek, és nem számítógépek vagyunk, ez sajnos nem várható el tőlünk. Így időről időre előfordul, hogy nem rakjuk ki az egy változó elejét vagy végét jelölő jelet vagy jelsorozatot. Már kevés tapasztalat is elég ahhoz, hogy megtanuljuk, az olyan hibákat, amelyeket a program előfordulásukhoz közel jelez, sokkal könnyebb megtalálni és kijavítani. A fenti típusú hibát viszont elvileg csak a végén kellene észrevennie a programnak, hiszen mi a szöveg akár teljes hátralevő részletét is kijelölhetjük egy parancs változójának. Természetesen a file végén majd kapnánk egy hibaüzenetet, hogy valahol elfelejtettük kitenni egy nyitó kapcsos zárójel párját, vagy valamely más módon kijelölt változót nem zártunk le, de ilyenkor már általában bottal üthetjük a hiba nyomát. Ennek elkerülésére a TEX nem engedélyezi, hogy egy közönséges utasítás változójában bekezdést lezáró üres sor vagy \par utasítás szerepeljen, ilyenkor

```
! Runaway argument
```
(Befejezetlen változó)

hibaüzenetet ad. Mivel a file-ban időről-időre új bekezdést kezdünk, a hibásan megadott változókat még azok előfordulásához közel észrevehetjük. Ennek a megoldásnak az egyetlen hátránya, hogy így olyan parancsok változójában sem adhatunk meg új bekezdést, ahol ez esetleg szükséges lenne. A probléma megoldását a TEX \long utasítása jelenti, amit a \def előtt kiadva olyan parancsot hozunk létre, amely változójában vagy változóiban már üres sor vagy \par is szerepelhet. Tehát a

```
\long\def\kurziv#1{{\it #1\/}}
```

definíció által megadott \kurziv utasítás változója, amelyet jól láthatóan kurzív betűkből szed ki a parancs, már több bekezdést is tartalmazhat.

A definíció egy másik, kevésbé hatékony módja a `\let` parancs. Hatására az őt követő, egyenlőségjellel elválasztott két parancsszó azonos jelentést kap. Tehát a

```
\let\ez=\az
```

hatására a (még nem feltétlenül definiált) `\ez` utasítás teljesen egyenértékű lesz a már létező `\az` utasítással. Így a parancs használatához a `\az` utasításnak vagy egy eredeti TEX utasításnak kell lennie, vagy pedig előzőleg definiálnunk kell. A parancs másik használata, amikor egy újonnan definiálandó utasítást egyetlen jellel azonosítunk a `\let\egy=1` mintájára. Ilyenkor az `\egy` utasítás majdnem teljesen egyenértékű lesz az 1 számjeggyel, habár néhány esetben előfordulhat, hogy a TEX nem fogadja el számjegyként.

A `\def` utasításnak a `\long` változatán kívül még egyéb változatai is léteznek. A teljesség igénye nélkül sorolunk még fel néhányat. Ha például egy definíciót a `\outer` kiegészítés előz meg, akkor a vele definiált parancsot nem használhatjuk változókban, egyéb definíciók megadásakor vagy a később következő feltételes utasítások feltételeiben, és egyéb hasonló helyeken. Ha véletlenül mégis ilyen helyen szerepelne, a TEX leáll és

```
!Runaway argument (Befejezetlen változó)
```

vagy

```
!Incomplete conditional (Nem értékelhető feltétel)
```

típusú hibaüzenetet ad. Ez a `\def` változat hasznos lehet olyan parancsok definiálásakor, amelyeket csak magukban szeretnénk használni, és tervezésükkor nem törődünk vele, hogy milyen hatást váltanának ki, amennyiben az előbb felsorolt helyeken adjuk ki őket. Így hibás felhasználásuk nemcsak a már kinyomtatott szövegben, furcsa külalakként jelentkezne, hanem a program futásakor is. Jó példája ennek a `\proclaim` parancs, amelyet nyilván csak arra terveztek, hogy magában használjuk és így a következő módon definiálták:

```
\outer\def\proclaim #1. #2\par{\medbreak ...
```

A `\long` és `\outer` lehetőségeken kívül a `\global` is megelőzheti a `\def` parancsot, amennyiben annak érvényességét azon a környezeten kívül

is érvényesíteni akarjuk, ahol kiadtuk. A \let utasítással megadott definíciónak is használhatjuk a \global változatát. A \global\def röviditése a \gdef parancs (a TEX egy primitív utasítása), amely teljesen ekvivalens vele. Ahelyett, hogy használatára adnánk példát, éppen az ellenkezőjét mutatjuk be, milyen hatást érhetünk el azáltal, hogy ezen változat használata nélkül a TEX csak az adott környezeten belül érvényesíti a definíciót. A 7.2. szakaszban bemutattuk, hogyan definiálhatjuk újra a \narrower parancsot. Ha azonban ezt az új értelmezést csak a szöveg egy behatárolt részén szeretnénk érvényesíteni, akkor az adott részt és a \narrower újradefiniálását kapcsos zárójelek közé zárjuk. A második zárójellel elhagyjuk a környezetet és vele együtt a \narrower ideiglenes új definíciójának érvényességét is. Ha azonban valamely más okból kerül egy parancs definíciója egy környezeten belül (például egy másik parancs érvényességét kívánjuk korlátozni), de a parancsot általános érvényűnek szánjuk, akkor a \global változatot kell felhasználnunk.

A \global változat, az előző kettő \long és \outer változatoktól eltérően minden olyan esetben használható, amikor valaminek a jelentését vagy értékét megadjuk vagy változtatjuk, és természetesen minden ilyen esetben azt eredményezi, hogy a megadás vagy változtatás általános érvényű lesz és nem korlátozódik az éppen érvényes környezetre. Így például a \global megadható az \advance, és a \let parancsok előtt vagy például a \hsize=127mm típusú értékadások előtt is.

7.4.2. Új paraméterek, mértékek és dobozok definiálása

Új utasítások definiálásakor nagyon sokszor előfordulhat, hogy új paraméterekre, mértékekre vagy dobozokra és azok tárolására is szükségünk van. Tegyük fel például, hogy matematikai szöveg szedésekor minél inkább automatikusan szeretnénk a tételeket szedni. A \proclaim parancs mintájára definiálhatunk egy \tetel parancsot:

```
\outer\def\tetel #1. #2\par{\medbreak
\noindent{\bf#1. Tétel\enspace}{\sl#2}\par}
```

amelynek csak a tétel számát és szövegét kell megadni, valahogy így:

```
1. Pitagorasz tétele: A derékszögű háromszög átfogójára
```

```
emelt négyzet területe ...
```

és kiszedi a tételt így:

1. Tétel *Pitagorasz tétele: A derékszögű háromszög átfogójára emelt négyzet területe ...*

Még könnyebb lenne a dolgunk, ha a program a tételek számozását is automatikusan végezné. A 4. fejezetben már láttuk, hogyan történik az automatikus oldalszámozás. Ennek mintájára könnyedén meg lehet valósítani a tételek automatikus számozását is, amennyiben a rendelkezésünkre áll egy, a \pageno-hoz hasonló, a tételek számát nyilvántartó paraméter. Ilyen paramétert kaphatunk a \newcount utasítással, ahol mi adjuk meg a paraméter nevét. Legyen mondjuk a tételeket számoló paraméter a \ttelno, amit tehát a \newcount\ttelno parancs hoz létre. Segítségével a file elején a

```
\newcount\ttelno  \ttelno=0
\outer\def\tetel #1\par{\medbreak \advance\ttelno by 1
\noindent{\bf\number\ttelno. Tétel\enspace}{\sl#1}\par}
```

kis „makrócsomag" kiadása megvalósítja célunkat. Az első sor létrehozza és nullára állítja \ttelno parmétert, míg a másik kettő az új \tetel-t definiálja. Ez annyiban különbözik az előzőtől, hogy annak első paramétere helyét most a \ttelno vette át. A \tetel minden kiadásakor először megemeli a \ttelno értékét eggyel és ezt az új értéket használja a tétel számaként. Új parancs még a \number primitív TEX utasítás, ami az utána következő paraméter értékét szedi ki számmal.

Új paraméterhez hasonlóan szükségünk lehet új mértékekre is. Például ha ideiglenesen meg kívánjuk változtatni az oldal magasságát és szélességét, a régi mértékeket valahol tárolnunk kell, amennyiben vissza akarunk térni hozzájuk. A \newcount mintájára a

```
\newdimen\rmag  \newdimen\rszel
```

utasítások létrehoznak két új, egyelőre még értékkel nem rendelkező mértéket. Ha ezek birtokában a szövegben az alábbi részletet szerepeltetjük

```
\rmag=\vsize \rszel=\hsize
\vsize=12cm \hsize=5cm ... szövegrészlet ...
```

```
\vsize=\rmag \hsize=\rszel
```

akkor az első sor szerint a `\rmag`, `\rszel` mértékek felveszik az adott magasságot és szélességet, a második sorban a magasságot és szélességet 12 illetve 5 centiméterre állítjuk, és így a **szövegrészlet** ilyen méretű oldalakra kerül, majd a harmadik sor visszaállítja az eredeti méreteket.

Tekintsük még át az új dobozok létrehozásának lehetőségét. Mint már mindenki sejti, a `\newbox` utasítás egy új, dobozok tárolására szolgáló változót hoz létre, aminek a `\setbox` utasítással adhatunk értéket. Használatukat a következő példa jól szemlélteti. A

```
\newbox\hasab  \setbox\hasab=\vbox{\hbox{a}\hbox{b}\hbox{c}}
```

sor hatására a `\hasab` változóban a megadott doboz — azaz egy olyan doboz, amelyik az a, b és c betűket tartalmazza egymás felett — kerül. Azután a file-ban bárhol, mondjuk a `\raise3pt\box\hasab` utasítást kiadva a program az adott helyre teszi, 3 ponttal megemelve, a `\hasab` tartalmát. Ilyenkor azonban az adott doboz is ürül, tehát a `\hasab` újabb használata előtt újra kell definiálnunk egy `\setbox` paranccsal. Ha egy így megadott doboz tartalmát többször fel szeretnénk használni, akkor a `\box` helyett a `\copy` paranccsal valahova *átmásoljuk* a tartalmát, amely nem üríti ki a dobozt. Azaz a doboz tartalma a parancs helyén is megjelenik, de a továbbiakaban is felhasználható.

7.5. Feltételes utasítások

Az eddigiekben megismert makródefiniálási lehetőségek is nagyon hasznosak lehetnek néha, de azért igazság szerint — ha jól meggondoljuk — nélkülük is megtehetnénk ugyanazt, amit velük, mégha időnként jóval több munka (értsd adatbevitel) árán is. Igazán hatékonnyá akkor válnak az utasítások, amikor adott feltételek teljesülésétől vagy megsértésétől függően másként lehet megadni őket. A TEX erre is képes a (programozásban már gyakorlatot szerzettek által jól ismert) `\if` típusú utasítások segítségével. Ezen utasítások általános alakja a következő

```
\if(feltétel)(ha igaz)\else(ha nem igaz)\fi
```

ahol a (ha igaz) részt tekinti a program — a benne szereplő utasításokkal és azok végrahajtásával együtt — ha a feltétel igaznak bizonyul, és hasonlóan a (ha nem igaz) részt akkor, ha a feltétel nem teljesül. Amennyiben nem kívánjuk megadni a (ha nem igaz) részt, azaz a programnak semmit sem kell végrehajtania ha a feltétel nem igaz, akkor a `\else` részt is elhagyhatjuk. *Nagyon vigyázzunk arra, hogy ne felejtsük el kitenni a parancsban a* `\fi` *utasítást*, amely jelzi a program számára, hogy hol van vége annak a résznek, amit csak a feltétel igaz vagy hamis volta szerint kell végrehajtania.

Az utasításban a (ha igaz) és a (ha nem igaz) rész tetszőleges szöveg vagy utasítás, illetve ezek kombinációja is lehet, amit a TEX akárhol máshol elfogadna, míg az `\else` és `\fi` rész éppúgy a parancshoz tartzik, mit az `\if`. Az `\if`(feltétel) rész több lehetséges alakot vehet fel, amelyek közös vonása, hogy mind az `\if` jelsorozattal kezdődnek. Az egyik ilyen lehetőség, ha `\ifnum`-mal kezdjük a parancsot, utána két számot adunk meg, amelyeket a program összehasonlít és ellenőrzi, hogy a megadott reláció fennáll-e köztük. Ilyenkor természetesen a számokat nemcsak közvetlenül adhatjuk meg, hanem paraméterek segítségével is. Például az

```
\ifnum\pageno > 1
```

parancskezdet hatására a TEX ellenőrzi, hogy a `\pageno` paraméterben tárolt oldalszám nagyobb-e, mint egy. Ha nagyobb, a feltétel igaznak bizonyul, ellenkező esetben hamisnak. Itt természetesen a > jel helyett = vagy < jelek is állhatnának, amikor is a program azt ellenőrizné, vajon a két szám egyenlő-e, illetve az oldalszám kisebb-e egynél.

Általános szokás, hogy a címoldalra nem rakunk oldalszámot, míg a további oldalakra igen. A `plain`, mint már írtuk, az oldalszámot az oldal lábsorába, középre kizárva szedi. Tegyük fel, hogy ez megfelel nekünk, csak a címlapon szeretnénk letiltani, amit természetesen az itt megadott utasítások segítségével tehetünk meg. A negyedik fejezet elején már láttuk, hogy a lábsor tartalma a

```
\footline{\hss\tizant\folio\hss}
```

paranccsal van megadva. Ezt szeretnénék felülírni az első oldal esetében. Az `\ifnum` parancsot használva ezt így tehetjük meg:

```
\footline{\ifnum\pageno = 1 \hfil \else
```

```
\hss\tizant\folio\hss \fi}
```

Amennyiben az oldalszám egy, akkor a lábsorban mindössze a `\hfil` utasítás szerepel, azaz üres lesz, míg ellenkező esetben, azaz az összes további oldalra a lábsor tartalma megegyezik a `plain` formátum által definiáltal. A fenti `\footline` parancsban az összes szóköz felesleges (de nem hibás) és csak azért szerepel, hogy jobban olvasható legyen a parancs, kivéve azt, ami az 1 után áll. Ugyanis ha az 1-et közvetlenül követné a következő parancs és annak az első jele a kifejtés után egy számjegy lenne, akkor a program azt is a számhoz venné és így az `\ifnum`-ban szereplő második szám nem 1, hanem egy eggyel kezdődő többszámjegyű szám lenne. *Hasonló problémák elkerülése érdekében tanácsos minden utasításban szereplő — numerikus értéket, tehát nem mértéket jelentő — számot mindig egy szóközzel lezárni.*

A konkrét példa után rátérünk a lehetséges `\if`(feltétel) részek (nem teljes) felsorolására. A parancsszó után (ami tehát mindig `\if`-fel kezdődik) megadjuk a feltételt is általános formájában, majd megmondjuk, hogy a TEX mikor tekinti a feltételt igaznak. Végül egy–egy példát adunk minden lehetőségre.

- `\ifnum`(első szám)(reláció)(második szám)
  A két szám paraméterrel, számjegyekkel vagy egyéb módon van megadva úgy, hogy a program egyértelműen azonosítani tudja. A reláció < = vagy > lehet. A feltétel akkor igaz, ha a két szám között a megadott reláció fennáll. Példa: `\ifnum\pageno = 1`

- `\ifodd`(szám)
  A számot ugyanolyan módon adhatjuk meg, mint az előző esetben. Ez a változat a (szám) paritását vizsgálja. Ebben az esetben csak egyetlen változó szerepel a feltételben. A feltétel akkor lesz igaz, ha a szám páratlan. Példa: `\ifodd\pageno`

- `\ifdim`(első mérték)(reláció)(második mérték)
  A két mérték, a számokhoz hasonlóan, bármely módon meg lehet adva, csak az a fontos, hogy a program egyértelműen megtalálja őket. A reláció itt is < = vagy > lehet és a feltétel akkor lesz igaz, ha a végül kapott két hosszmérték között a megadott reláció fennáll. Természetesen a két hosszmértéknek nem feltétlenül kell azonos mértékegységben megadottnak lennie. Például: `\ifdim\hsize > \vsize`

- `\ifvmode` vagy `\ifhmode` vagy `\ifmmode`
  Ezek a változatok magukban hordozzák a feltételt, nem kell a parancs szokásos (feltétel) részét megadni. Azt vizsgálják, hogy maga a program milyen módban van. Az első akkor igaz, ha a TEX függőleges vagy belső függőleges módban van, a második akkor, ha vízszintes vagy belső vízszintes módban, míg a harmadik akkor, ha matematikai módban. Például: `\ifhmode`

A következő, utolsónak ismertetett változat kissé eltér az előzőktől, hiszen itt két lehetőség helyett többet is tekint a program.

- `\ifcase`(szám)(ha a szám 0)`\or`(ha a szám 1)`\or` ... `\or`(ha a szám n)`\else`(ha a szám bármi más)`\fi` Ekkor, mint látható, összesen $n+2$ eset lehetséges, attól függően, hogy a megadott (szám) paraméternek az értéke 0, 1, ... vagy n — ez már az $n+1$. eset — vagy ezek közül egyik sem, amikor is a (ha a szám bármi más) eset lesz érvényes. Ezen változat esetén is elmaradhat az `\else` parancs és az azt követő rész, azonban ha más eseteket akarunk elhagyni, akkor is ki kell tenni az azokat határoló `\or` parancsrészeket.

MATEX Az `\ifcase` parancs használatára jó példa a `matex.tex` makrócsomag `\kelt` utasítása.[4] Ennek hatására a kiadása helyén megjelenik a napi dátum[5] a magyar szabvány szerinti kiírásban. Például ezen könyv utolsó lenyomata — amely már egyenesen a nyomdába ment — 1988. március 14. készült. Az előbbi dátumot a `\kelt` parancs hatására maga a program helyezte el a szövegben. Persze a makrót nem lehetett volna megírni, ha a TEX-nek nem lenne három beépített, primitív paramétere, a `\year`, `\month` és a `\day`, amelyek mindenkor az éppen adott év, hónap illetve nap számát tartalmazzák a megfelelő sorrendben. Tehát 1988. március 14. a `\year` értéke 1988, a `\month` értéke 3 és a `\day` értéke 14 volt. Ezen paraméterek birtokában a `\kelt` parancs a következő módon adható meg:

```
\def\kelt{\number\year. \ifcase\month\or január\or február\or
...\or október\or november\or december\fi \space\number\day.}
```

[4] A `matex.tex` makrócsomagot részletesen a 9. fejezetben fogjuk ismertetni.

[5] Ez természetesen csak abban az esetben igaz, ha a számítógép ismeri azt. A legtöbb IBM PC/XT/AT kompatibilis gépben azonban van elemről üzemeltetett óra vagy az adott DOS implementáció megkérdezi a dátumot a bekapcsoláskor.

A definíció első és utolsó része nem igényel túl sok magyarázatot, csak talán annyit, hogy a \year és \day paraméterek értékét megint a \number paranccsal írattuk ki. A dátumban azonban Magyarországon a hónap nevét betűvel írjuk ki, természetesen magyarul, míg a hónapnak csak a számát tartalmazza a \month paraméter. Így az \ifcase parancsot kell használnunk a hónap nevének kiíratására. Figyeljük meg, hogy mivel 0. hónap nincsen és így a \month 0 értékéhez nem kellett semmit hozzárendelnünk, a parancsban a (szám)-ot megadó \month paramétert közvetlenül követi egy \or parancs. Utána \or-okkal elválasztva egymást követik a hónapok nevei, és mivel a 12. hónap után megintcsak nincs más lehetőség, nem adtuk meg az \else parancsrészletet.

Előfordulhat, hogy az eddig ismertetett \if lehetőségekkel nem tudunk megoldani egy adott problémát. A TEX lehetőséget nyújt arra, hogy magunk is definiálhassunk \if típusú parancsokat. A \newif\ifvalami parancs kiadásakor három parancs egyszerre, az \ifvalami, a \valamitrue és a \valamifalse. Ezek közül az \ifvalami a néhány bekezdéssel ezelőtt említett \if(feltétel) részt helyettesítheti a feltételes parancsokban — tehát ilyenkor nem létezik igazi (feltétel) rész — és a feltétel aszerint igaz vagy hamis, hogy utoljára a \valamitrue vagy \valamifalse parancsot adtuk-e ki, amelyek közül az első igazra állítja be a feltételt, míg a második hamisra. Természetesen a valami helyén bármely olyan jelsorozatot szerepeltethetünk, ami parancsnevekben megengedett. A használat jobb megértéséhez talán nem árt egy valóságos példa. Tegyük fel, hogy az előző szakaszban adott \tetel parancsot nemcsak tételek, hanem állítások kinyomtatására is használni kívánjuk, és a két struktúra csak a név használatában tér el. Adjuk ki a \newif\iftetel parancsot, és módosítsuk a \tetel definícióját a következőképpen:

```
\outer\def\tetel #1\par{\medbreak \advance\ttelno by 1
\noindent{\bf\number\ttelno. \iftetel Tétel\else Állítás \fi
\enspace}{\sl#1}\par}
```

Íly módon ha a \teteltrue parancsot kiadjuk a szövegben, akkor a definícióban adott \iftetel feltétel az utána következő kiértékeléskor igaznak bizonyul és a **Tétel** részlet kerül szedésre, míg egy esetleges \tetelfalse parancs után következő kiértékeléskor a feltétel hamisnak adódik, és a definíció **Állítás** része lép életbe, azaz a parancs az **Állítást**-t szedi ki.

7.6. Lábsor és élőfej könyvekben

Valójában a címben megadottnál kicsit általánosabb kérdésekre válaszolunk ebben a szakaszban. Több oldalas dokumentumok szedésénél sokszor előfordulhat, hogy az oldalak egyes jellemzőit vagy paramétereit nem állandóra kívánjuk beállítani, hanem bizonyos fokig az oldaltól, illetve az oldalszámtól szeretnénk függővé tenni. Ilyen esetekre a legjobb példa viszont a címben megadott lábsor és élőfej. Ezen könyvön is jól látható, hogy a páros és páratlan oldalakon különbözik az élőfej (a lábsor pedig csak azért nem, mert az mindkét esetben üres). Természetesen ennél bonyolultabb problémák is felléphetnek, amelyek mindegyikére itt nem térhetünk ki. Mindenesetre néhány példán szemléltetjük, hogyan oldhatók meg ezek a kérdések.

Kezdjük tehát a címben megadott esettel, azon belül is e könyv oldalaival. Mint láthatjuk, itt a lábsor minden eseteben üres, míg az élőfejben a páros, baloldali oldalakon balra zárva szerepel az oldalszám, majd jobbra zárva az „7. Fejezet Új utasítások definiálása" szöveg (a különböző betűtípusokkal most ne törődjünk). Ugyanott a páratlan, jobboldali oldalakon balra zárva szerepel a „7.6. Lábsor és élőfej könyvekben" felirat és jobbra zárva az oldalszám. Ezeket a változatokat természetesen most is a `\headline` paranccsal adhatjuk meg, a feltételes utasításokat is használva, a következő módon (feltételezve, hogy antikva betűket használunk az élőfejben).

```
\headline{\tizant\ifodd\pageno 7.6. Lábfej és ... \hfil\folio
\else \folio \hfil 7. Fejezet ... \fi}
```

A parancs hatására tehát a baloldali oldalakon az a részlet jelenik meg az élőfejben, amit először adtunk meg, míg a jobboldali oldalakon a második. Ne felejtsük el, hogy a program mindig a lábsor és az élőfej kiszedése után,de még a következő oldal szedése előtt emeli az oldalszámot, így az oldal szedése során mindvégig az az oldalszám van érvényben, amit majd az oldalnak ad.

Ha már itt tartunk, megmutatjuk a `\folio` parancs definícióját is, amely, mint emlékszünk, aszerint írja ki az oldalszámot arab vagy római számokkal, hogy mi az előjele a \pageno paraméternek.

```
\def\folio{\ifnum\pageno < 0 \romannumeral-\pageno
\else\number\pageno \fi}
```

Sok magyarázatot nem kell fűznünk a definícióhoz. Egyetlen fontos tudnivaló, hogy a `\romannumeral` parancs római számokkal írja ki a változójaként megadott számot. Itt pedig a `-\pageno` a változója, tehát a valódi oldalszámot írja ki, természetesen római számokkal.

Kicsit már bonyolultabb probléma a következő. Mint az előző szakaszban említettük, néha egy szöveg első oldalán más helyre, mondjuk a lap aljára középre zárva szokás tenni az oldalszámot. Amíg ezt csak egy oldallal szeretnénk végrahajtani, nincs nagyobb probléma, az előző szakaszban láttuk a megoldást. Ott azonban erősen kihasználtuk, hogy az első oldal volt a többiektől különböző. A könyvek esetében ezt minden fejezet vagy olyan nagyobb, önálló jobboldali oldalon kezdődő egység esetén címoldalán el kell érnünk. A probléma abból adódik, hogy előre nem tudhatjuk, az adott rész éppen melyik oldalra esik majd. Az előző szakaszban adottak mintájára, de kicsit bonyolultabban a következő módon járhatunk meg. Bevezetünk a `\newcount\fejezetoldal` utasítással egy új paramétert, amely mindig az éppen írt fejezet oldalszámát tartalmazza. Ezt úgy érjük el, hogy minden fejezet legelején kiadjuk a `\fejezetoldal = \pageno` parancsot. Ezek után már csak a következő módon kell átdefiniálni az élőfejet és a lábsort.

```
\footline{\ifnum\pageno = \fejezetoldal \hss\tizant\folio\hss
\else \hfill \fi}
\headline{\ifnum\pageno = \fejezetoldal \hfill \else
\tizant\ifodd\pageno 7.6. Lábfej és ... \hfil\folio
\else \folio \hfil 7. Fejezet ... \fi\fi}
```

Még nehezebb problémát jelent, ha a bal margót szeretnénk megváltoztatni az oldalszámtól függően. A könyvek szedésekor mindig az oldal külső szélén van nagyobb margó. Az eltérés körülbelül 1,5 centiméter. Tehát ilyen esetekben a jobboldali oldalakon mindig 1,5 centiméterrel nagyobb margót kell hagynunk. Ilyen esetekben a `\hoffset` paraméter megadásakor nem használhatjuk a feltételes parancsokat, mivel annak mértékét közvetlenül kell megadni. Tehát az egyetlen megoldás, hogy a `\hoffset` értékét többször adjuk meg, mindig az adott feltételeknek megfelelően, az oldalszámtól függően. Mondjuk azt szeretnénk, hogy a páratlan oldalakon a `\hoffset`

értéke 35 milliméter legyen, míg a páros oldalakon 20 milliméter. A szöveg elején minden további nélkül definiálhatjuk és ki is adhatjuk a \marg parancsot, amely ezeket az értékeket állítja be.

```
\def\marg{\ifodd\pageno\hoffset=35mm \else\hoffset=20mm \fi}
\marg
```

Csakhogy ez a parancs mindössze egyszer, a szöveg legelején állítja be a margó nagyságát, és a későbbiekben csupán a szöveg ismeretében nem tudhatjuk, hogy a program mikor fog oldalt váltani, azaz mikor kell megint kiadnunk a parancsot. Ezen, és sok más, hasonló problémán a következő módon segíthetünk. A TEX a \output parancs hatására végzi el a 4. fejezetben leírt módon az egyes oldalak előállítását minden egyes olyan alkalommal, amikor már elegendő anyag gyűlt össze az oldalhoz. Természetesen ezt a parancsot nem kell meghívnunk, a TEX mindent automatikusan végez. Nem is kell ismernünk a parancsot a továbbiak megértéséhez sem, csak annyit kell tudnunk, hogy a plain formátumban

```
\output{\plainoutput}
```

és az mplain formátumban

```
\output{\plainoutput\pagebody}
```

alakban van definiálva. Figyeljük meg, hogy itt nem definícióról van szó, a \plainoutput szerepe nem más, mint hogy a változójában szereplő utasításokat végrehajtassa. Mindkét esetben a \plainoutput parancs az \output változója, csak míg a plain esetében ennek magának már nincsen változója, addig az mplain-ben megadtuk neki a \pagebody változót. A \plainoutput parancs elvégez mindent, amit elvárunk tőle, majd megemeli az oldalszámot és visszaadja a vezérlést a program fő részeinek a következő oldal szedéséhez. Tehát a közvetlenül utána kiadott parancsok már az új oldal szedésére felkészülve találják a programot, de úgy, hogy ahhoz még nem kezdett hozzá. Nyilvánvalóan ez a legmegfelelőbb pillanat sok, az oldal külalakját az oldalszám vagy egyéb paraméter szerint változtató parancs — köztük a fentebb definiált \marg parancs — kiadására. Így a problémát úgy oldhatjuk meg, hogy az \output utasítást átdefiniáljuk a következők mintájára:

```
\output{\plainoutput\marg}
```

vagy

```
\output{\plainoutput\pagebody\marg}
```

aszerint, hogy a `plain` vagy `mplain` formátumot használjuk. *Vigyázzunk azonban,* ne használjuk meggondolatlanul ezt a módszert, mivel nagyon sok baj okozója lehet. Például semmiképp sem használható arra, hogy a `\hsize` paramétert megváltoztassuk vele, hiszen a régi oldal szedésének lezárásakor már esetleg egy jó rész ki van szedve az új oldalból a régi `\hsize` érték felhasználásával.

7.7. Lábjegyzetek automatikus számozása

Egy hosszabb lélegzetű irásmű esetén sokszor előfordulhat, hogy egyes részleteket nem a szövegben elfoglalt helyük szerinti sorrendben írunk meg. Ilyenkor a később megírt részletek zavart okozhatnak a számozásban. A program automatikusan számozza az oldalakat, a később ismertetett MTEX makrócsomag megteszi ezt a fejezetekkel és egyéb szövegegységekkel, de a lábjegyzetek számozását egyik sem végzi el. Viszont nagyon kényelmetlen lehet, ha egy, a második oldalra váratlanul bekerült lábjegyzet miatt az egész könyvben át kell számozni a lábjegyzeteket. Ezen segíthetünk, ha bevezetjük az automatikus számozásukat.

Először is a `\newcount` paranccsal hozzunk létre egy új számlálót — mondjuk nevezzük `\labjszam`-nak —, ami a lábjegyzetek számát tartalmazza majd. Szükségünk lesz még egy parancsnévre, amely a `\footnote` funkcióján kivül az automatikus számlálást is végzi. Ez jó alkalom arra is, hogy a lábjegyzetek létrehozására egy olyan nevet találjunk, amely már magyarul jelzi feladatát. Legyen tehát a lábjegyzeteket látrehozó új parancs neve `\labj`, amelyet a következő módon definiálunk:

```
\def\labj#1{\global\advance\labjszam by 1
\footnote{$^{\number\labjszam}$}{#1}}
```

Az így definiált parancs a `\footnote` első változójába a felső index magasságába emelt éppen érvényes lábjegyzetszámot teszi, aminek növelését

is elvégzi. Látható, hogy ha valahonnan újra szeretnénk kezdeni a lábjegyzetek számozását (vagy egyáltalán csak el szeretnénk kezdeni), akkor ott a `\labjszam=0` parancsot kell kiadnunk. A `\labj` parancs egyetlen változója a lábjegyzet szövege, amit az a `\footnote` második változójának ad tovább.

Egy dolgot érdemes még megfigyelni a definícióban. Az egyik a `\global` parancs használata, amivel azt érjük el, hogy a lábjegyzetek számozása akkor is folyamatosan történik, ha egy lábjegyzetet egy környezeten belül hozunk létre. Használata nélkül a

```
{\narrower Keskeny szövegrészlet ... \labj{Ide jön az
első lábjegyzet} befejezés.} Széles szöveg ...
\labj{Második lábjegyzet}
```

részletben a két lábjegyzet azonos számot kapna, hiszen hiába emelné meg az első `\labj` parancs a `\labjszam` értékét eggyel a `\narrower` számára kitett kapcsos zárójeleken belül, annak új értéke érvényét vesztené a zárójeleken kívül.

Az így létrehozott `\labj` parancs a lábjegyzet már jellemzőinek a beállítására is szolgál. Ha például az `mplain` adta alapértelmezés helyett jobban szeretnénk nyolcpontos betűkkel szedni a lábjegyzetet, akkor a

```
\font\nyolcant=hamr8
\def\labj#1{\global\advance\labjszam by 1
\footnote{$^{\number\labjszam}$}{{\nyolcant #1}}}
```

definícióval ezt megtehetjük. Figyeljük meg, hogy itt a `\footnote` második változóját két zárójel közé zártuk, hogy a `\nyolcant` parancs hatását csak a lábjegyzet területére korlátozzuk.

8. fejezet

Hibajavítás

Ebben a fejezetben megpróbáljuk összefoglalni azokat az ismereteket, amelyek a TEX futtatása során fellépő leggyakoribb lehetséges hibák kijavításához szükségesek. Ilyen kérdésekkel már foglalkoztunk előzőleg, főleg a 3. fejezetben, de máshol is, egy-egy konkrét utasítás vagy utasítástípus kapcsán.

8.1. A hiba megtalálása

Nézzük először, hogy mit lehet kiolvasni egy hibaüzenetből, hogyan lehet belőle megtalálni a hibát. Egyik legegyszerűbb és leggyakoribb hibaüzenet az

```
! Undefined control sequence.
1.12 \hgill
          \kelt\vskip10pt
?
```

ahol (és minden más hibaüzenetben is) az első sor a hiba jellegének a meghatározását tartalmazza, míg a második a hibás szövegrészlettel kezdődik, vagy arra utaló információkat tartalmaz. Az első sor azt mondja meg, hogy a program definiálatlan parancsnevet talált. A második sor, amely más hibaüzenetekben esetleg hátrább is csúszhat, már a hibás sort tartalmazza. Ez elsőként mindig annak a sornak a számát mutatja, amelyben a hiba van. A

sorszámot az éppen feldolgozott file-ban kell érteni. Azaz, ha egy file 55. sora egy `\input makro.tex` parancsot tartalmaz, és a program a `\makro.tex` file 12. sorában találja a hibát, akkor a hibaüzenetben sorszámként a `1.12` szerepel, ami semmilyen utalást nem tartalmaz arra a tényre vonatkozóan, hogy a hibás file-t az előző file 55. sorában olvastuk be.

A sorszám meghatározása után következik a hibás sor tartalmának a megadása, mégpedig általában két sorba törve. Az első sorban található az a rész, amit a program már beolvasott, míg a második sorban a file-ban levő sor befejezése. A fenti hibaüzenet esetén a TEX a `\hgill` rész beolvasása után találta meg a hibát és állt le. Ez általában nagy segítséget jelent a hiba megtalálásában, hiszen ha kis szerencsénk van (és nem túlságosan ravasz, bonyolult hibáról van szó), akkor a program éppen a hibás rész után fog leállni, mint esetünkben is. A hibát tehát az értelmetlen `\hgill` parancs okozta, ami nyilván egy elgépelés eredményeként, a `\hfill` parancs helyett került a file-ba. A hibaüzenetnek ezen második része nem feltétlenül következik rögtön az első sor után. A program, amennyiben bonyolultabb hibát talál, a két sor között a hibáról még számos egyéb információt is közölhet, mint később látni fogjuk. Ezzel szemben általános szabály, hogy a hibás sor után már csak egyetlen sor következik, amelynek elején egy kérdőjel található, majd a villogó kurzor, jelezvén, hogy a program valamilyen választ vár. Így tehát e hibaüzenet két legfontosabb része — legalábbis a kezdők részére — az első sora és a vége. Az ezen részek közötti információ — amennyiben jelen van — csak akkor szükséges, ha az eddigiek birtokában nem vagyunk képesek a hiba megtalálására.

Nézzünk most egy árnyalatnyival bonyolultabb hibaüzenetet, amely már talán ismerős egy kicsit.

```
! Illegal unit of measure (pt inserted).
<to be read again>
                   c
<to be read again>
                   n
1.2 \hsize=5.5cn

?
```

Itt tehát az első sor után, de még a hibát tartalmazó sor kijelzése előtt egyéb információt is kapunk. A hibaüzenet jelentése egyébként, hogy a program meg nem engedett — tehát számára értelmezhetetlen — mértékegységet talált, és ha nem történik javítás, akkor azt pontnak, azaz a pt mértékegységnek tekinti majd. A hibás sort kiíró rész jelzi, hogy mindez az input file 2. sorában történt, miután a program beolvasta a `\hsize=5.5cn` részt. Ebből azonnal világos számunkra, hogy a hibát a cn-nek gépelt rész okozta, ami nyilván `cm` akart lenni. Az ezen sor és az utolsó, a kérdőjelet befoglaló sor közötti üres sor tartalmazza a hibás sornak a `\hsize=5.5cn` utáni részét, ami tehát üres. Végül az eddig tárgyalt részek közé eső szakasz egyéb, fontos információt tartalmaz. Ezekre a sorokra általában jellemző, hogy a < és > jelek közé zárt szöveg nem a hibás file-ban található, nem is a hiba jellegének megállapításához nyújt segítséget, hanem az utána következő, a hibával már konkrétan összefüggő részről tartamaz információt. Ezek közül az egyik legáltalánosabb az itt előforduló

```
<to be read again>
```

amely azt mondja ki, hogy a program, noha egyszer már beolvasta a c és n karaktereket, azokat a hiba elhárítása után még egyszer beolvassa majd, hacsak ellenkező értelmű utasítást nem kap.[1]

Egy újabb, még mindig viszonylag egyszerű hibaüzenet a következő:

```
! Missing $ inserted.
<inserted text>
                $
<to be read again>


1.16

?
```

[1] Ugyanis úgy gondolja, hogy a felhasználó egyszerűen kifelejtette a pt mértékegységet és a szám után következő szöveg nem egy elütött másik mértékegység, hanem már fontos és pontos része az inputnak. A továbbiakban látjuk majd, hogy a TEX-nek (majdnem) mindig meglesz az elképzelése arról, hogy hogyan kellene a hibát kijavítani, amit végre is hajt majd ellenkező értelmű utasítás hiányában.

Ilyen üzenetet kapunk akkor, ha elfelejtettünk kitenni egy, a matematikai módot nyitó vagy záró $ jelet. A hibaüzenet pontos jelentése, hogy a program egy szerinte hiányzó $ jelet illesztett a file-ba a megadott helyre. Az input file hibás sorát mutató mindkét sor (az `1.16`-ot tartalmazó és az azt követő sor) most üres, jelezve, hogy a program a hibát az üres 16. sorban találta. Ez azt jelzi, hogy egy elfelejtett záró $ jelről van szó, hiszen matematikai módban nem fordulhat elő új bekezdést előíró üres sor vagy (ahogy azt a TEX értelmezi) \par parancs. Erre utal a kicsit előbb található újra beolvasandó \par kiírás is. Végül az `<inserted text>` kiírás szó szerinti jelentése „beillesztett szövegrészlet", amely természetesen az utána következő $ jelre vonatkozik.

Előfordulhat, hogy a hiba nem közvetlenül az input azon sorában található, ahol a TEX megtalálja. Ez leggyakrabban akkor fordul elő, amikor egy utasítás változójában van a hiba, mint a következő példában is. (Amelyben a hibaüzenet végéről már elhagytuk a kérdőjelet tartalmazó sort. A továbbiakban mindig így teszünk, hacsak nincsen a magyarázatban fontos szerepe a kérdőjelnek, illetve a rá adott válasznak.) Tegyük fel, hogy az input file-ban a

```
\centerline{Itt egy kis \hiba van.}
```

sor szerepel, és a \hiba utasítás *nincsen* definiálva az inputban. Ennek eredményeként a következő hibaüzenet jelenik majd meg a képernyőn:

```
! Undefined control sequence.
<argument> Itt egy kis \hiba
                           van.
\centerline #1 ->\line {\hss #1
                              \hss}
1.15 \centerline{Itt egy kis \hiba van.}
```

Az utolsó sor után van még egy üres sor, ami megint azt jelenti, hogy a program csak a teljes 15. sor beolvasása után találta meg a hibát. A második sor `<argument>` részének jelentése „változó", amely arra utal, hogy az itt következő rész egy másik parancsnak a változója. Az utána következő részlet a konvencióknak megfelelően ott törik meg új sorra, ahol a program a hibát sejti. A következő sorok pedig azt a parancsot és a kifejtését mutatják, amelynek a változójáról szó van, megint a hibás helyen megtörve. A hibaüze-

netből tehát az olvasható ki, hogy egy `\centerline` utasítás változójában van a hiba, amely változó az

```
Itt egy kis \hiba van.
```

szövegrészletet tartalmazza, annak is a `\hiba` részével van baj, és a hiba jellege az, hogy a program egy definiálatlan parancsnevet talált. Ha ezekhez még hozzávesszük, hogy a hiba a kiírt 15. sorban van, igazán nem nehéz megtalálni, hogy végül is mi a baj.

Természetesen nem lehet azt állítani, hogy minden hiba megtalálása ilyen egyszerű. Előfordulhatnak igen bonyolult, rafinált problémák is. De még a legegyszerűbb hiba megtalálása is nehéz lehet az angolul nem értő felhasználók számára. Mi itt annyiban tudunk segíteni, hogy a későbbiekben megadjuk a leggyakoribb hibaüzenetek listáját, azok magyar fordítását, illetve jelentését. De előbb tekintsük át, hogyan lehet a megtalált hibákat kijavítani. Erre két okunk is van. Egyrészt kétségtelenül ez az egyszerűbb része a feladatnak, és így talán sikerül egy kis vigaszt nyújtani az éppen elkeseredő olvasónak, másrészt a folyamat egyes részei a hibafelismerést is segíthetik.

8.2. A hiba kijavítása

Amikor a program hibát talál, akkor a kiírása után, a legutolsó, a kérdőjelet tartalmazó sorban a kurzor villogva ott marad, és a program felfüggeszti a futását, amíg valamilyen választ nem kap tőlünk. A legegyszerűbb válasz a (return) gomb megnyomása, amellyel nagyjából azt hozzuk a program tudomására, hogy „menj tovább a legjobb belátásod szerint, rád bízom a hiba kijavítását". Ilyenkor a program mindenképp tovább fut, az egyszerűbb hibákat valahogy kijavítja, (és ilyenkor a legtöbb esetben, ha nem is közvetlenül a hiba után, de nem sokkal később visszatalál a „rendes kerékvágásba",) a bonyolultabbakat azonban az esetek többségében nem tudja igazán megoldani, szinte rögtön hibát jelez újra. Ilyenkor még néhányszor próbálkozhatunk a (return) billentyű lenyomásával, de ha sok hibát talál a program ugyanazon a környéken (amit mindig meg tudunk állapítani a kijelzett sorszámokból) akkor valami mást kell tennünk.

A másik legegyszerűbb válasz a hibaüzenetre az x vagy X (általában mindig egyetlen betű a válasz, amit akár kisbetűként, akár nagybetűként megadhatunk), amit megint egy (return)-nel kel lezárni. Ennek hatására a program befejezi a futását, visszaadja az irányítást a DOS-nak és a szövegszerkesztőnk segítségével nekikezdhetünk a hiba kijavításának. Ennek módozataira persze konkrét tanácsot nem tudunk adni, hiszen a hiba megtalálása és felismerése egyben azt is jelenti, tudjuk, milyen változtatást kell megtennünk az input file-ban. Az pedig, hogy azt hogyan tesszük, már nem a TEX programon múlik, hanem a szövegszerkesztőn. Az egyetlen fontos dolog, amire emlékeznünk kell, hogy a program mindig elteszi a log file-ba mindazt az információt, amit a képernyőre kiír, és a válaszokat, amiket mi adunk a képernyőn. Ennél általában kicsit még többet is kerül a log file-ba, külön kérésre pedig még sokkal több információ (erre majd később térünk ki bővebben.) A log kiterjesztésű file neve mindig megegyezik annak a filenak a nevével, amelyen először kezdtük el futtatni a TEX-et. Ha a program futása során több más (például \input-tal adott) input file-t is feldolgoz, ezekhez nem készít új \log file-t.

Így tehát a hibajavításnak a legegyszerűbb (és a könyv szerzői által ajánlott) módja a következő:

1. Futtassuk a TEX-et az inputon addig, amíg lehet. Ha nem talál hibát, készen vagyunk. Ha talál, kényszerítsük a folytatásra a (return) billentyű lenyomásával mindaddig, amíg ennek értelmét látjuk.
2. Szakítsuk meg a program futását a hibaüzenetre adott X válasszal és javítsuk ki a log file-ban tárolt hibákat a szövegszerkesztő segítségével. Kezdjük előlről a futtatást.

A TEX IBM PC/XT/AT-re írt implementációi azonban megengedik az interaktív hibajavítást is. Vannak, akik jobban szeretik ezt a módszert, hiszen az előző használatával előfordulhat, hogy a program általunk nyilvánvalónak ítélt hibát másképp értelmez, és rosszul javít. Ennek eredményeként esetleg nem tudjuk folytatni a futtatást, le kell állnunk. Ha mindez a file elején történik, vagy adott file-on belül sokszor megesik, rengeteg időt veszíthetünk. Ezen segít, ha a hibát azonnal kijavítjuk a program leállítása nélkül és úgy engedjük a folytatást. *Ne feledjük azonban, hogy ilyen esetekben a javítást nem az input file-ban hajtottuk végre és így az csak az egyszeri futtatásra érvényes.* Így a program lefutása után ugyanúgy végre kell hajtanunk az előző bekezdésben leírt hibajavítást, mintha a hibákat nem

javítottuk volna ki a futás közben. Természetesen a log file most is rendelkezésünkre áll, sőt, mivel tartalmazza az interaktív javításainkat, részben már azt is leírja, milyen módon kell kijavítanunk a hibákat az inputban. Egyetlen hibája ennek a módszernek az, hogy az emberek hajlamosak az egyszer már kijavított hibákról elfeledkezni.

Az interaktív hibajavítás során a hibaüzenet utolsó sorában szereplő kérdőjelre adott válasz határozza meg, hogyan viselkedik a program. Ha a kérdőjelre egy másik kérdőjellel válaszolunk, megkapjuk a lehetséges válaszok listáját:

```
Type <return> to proceed, S to scroll future error messages,
R to run without stopping, Q to run quietly,
I to insert something, E to edit your file,
1 or ... or 9 to ignore the next 1 to 9 tokens of input,
H for help, X to quit.
```

Ezek, kicsit bővebben kifejtve a következő lehetőségeket rejtik, a számítógép adta sorrendben felsorolva:

1. A (return) billentyű megnyomása, amelynek hatását már leírtuk.
2. Az S billentyű (és természetesen az azt követő (return) billentyű) hatására a program a következő hibánál már nem áll meg, csak kiírja a képernyőre, és úgy tekinti, mintha egy (return)-nel válaszoltunk volna rá. A programfutás gyorsaságától függően ezek a hibaüzenetek esetleg el is tűnhetnek a képernyőről — különösen ha túlságosan sűrűn fordulnak elő a hibák az input file-ban — de mindenképp megtalálhatók lesznek a log file-ban. A program válasza ebben az esetben a következő:

   ```
   OK, entering \scrollmode...
   ```

3. Az R billentyű hatására (a továbbiakban nem írjuk ki a választ mindig lezáró (return) billentyűt) az előző esethez hasonlóan viselkedik a program, de még kevésbé hajlamos a leállásra. Míg például az előző esetben leáll, ha nem talál egy file-t, amit \input-tal akarunk behívni, addig itt már ez sem befolyásolja. A program válasza ebben az esetben a következő:

   ```
   OK, entering \nonstopmode...
   ```

4. A Q billentyű hatására éppen úgy viselkedik, mint az R billentyűére, de ilyenkor a hibaüzeneteket már csak a log file-ba helyezi el, a képernyőre nem írja fel. A program válasza ebben az esteben a következő:

```
OK, entering \batchmode...
```

5. Az I billentyű után akár közvetlenül megadhatunk egy, a TEX-nek szánt szövegrészletet, akár az I billentyűt egy (return)-nel lezárva megvárhatjuk, amíg a program azt bekéri. Az első esetben a következő minta szerint járunk el:

```
! Undefined control sequence.
1.12 \hgill
\kelt\vskip10pt
? I\hfill (return)
```

ahol az utolsó sorban szereplő jeleket a ? kivételével mi írjuk be. A második esetben a következőképp néz ki a képernyő:

```
! Undefined control sequence.
1.12 \hgill
\kelt\vskip10pt
? I(return)
insert>\hfill(return)
```

ahol az I és \hfill rész a válaszunk. Ez a válasz akkor helyes, ha a hibás részt a program nem értelmezi semminek, hanem egyszerűen átugorja, egyéb utasítás hiányában. Ez történik például az itteni példában, azaz ha definiálatlan paracsnévre talál. A programhoz ily módon adott sorok végére ne tegyünk felesleges szóközöket.

6. Ha azonban a program a hibás részt például szövegrészletnek értelmezi, mint abban az esetben, amikor a cm mértékegységet véletlenül cn-nek írtuk be, akkor a fenti válasz előtt a hibás részeket törölni kell, hiszen anélkül azt a program még egyszer beolvasná, mint ahogy figyelmeztet is rá. Ezt elkerülhetjük, ha megadjuk annak a néhány következő jelnek és parancsnak az együttes számát, amelyeket nem akarunk beolvastatni a programmal. Ilyenkor ezeket átugorja, anélkül, hogy tekintettel lenne rájuk, majd újra megáll és újabb utasítást vár. Az így megadott szám azonban nem lehet 100-nál nagyobb. Példánkban a helyes válasz tehát 2, mire a következő jelenik meg a képernyőn:

```
? 2
<recently read> n
1.2 \hsize=5.5cn
?
```

ami után az előző esetben leírt módon folytathatjuk a hibajavítást a `cm` mértékegység megadásával.

7. A `H` billentyű arra utasítja a programot, hogy próbáljon meg még több segítséget adni. Erre csak annak lehet igazán szüksége, aki nagyon ritka hibával találkozik és angolul is ért, hiszen a következő szakaszban leírjuk a gyakori hibaüzenetek jelentését és a hibák megszüntetésének a módját, aminél több információt ezek a segítő üzenetek sem adnak.
8. Az `X` billentyű hatására, mint már mondtuk, leáll a program futása.
9. Az `E` billentyű hasonlóan viselkedik az előző esethez. Elméletileg ez még fel is kellene, hogy készítse az input file-t arra, hogy az adott hibát tartalmazó sornál javíthassuk, de a gyakorlatban az IBM PC/XT/AT-ken ez nem jelent mást, mint a program leállítását és a

   `You want to edit file input.tex at line ...`

   üzenet kiírását, aminek a jelentése: „Az input.tex file-t kell szerkesztenie a ... sorban". Itt természetesen `input` helyett annak a file-nak a neve szerepel, amelyen a program éppen dolgozott és kiíródik a megfelelő sor száma is.

Még egyetlen módon befolyásolhajuk a programot futás közben. A legtöbb programhoz hasonlóan a TEX is leáll, ha a (Ctrl)–C vagy (Ctrl)–(Scroll Lock) billentyű-kombinációkat lenyomjuk. Egyes implementációknál csak ezeknek egymás után háromszori lenyomása hatásos, tehát inkább használjuk ezt a megoldást, ha valamiért le kívánjuk állítani a program futását. Azonban ezek hatására sem áll le azonnal a program, csak kis idő elteltével, általában akkor, amikor egy-egy oldal végére ér. Még egy kivétel van: ha `Q` választ adunk egy hibaüzenetre, azaz a program `\batchmode`-ba lép, már ennek hatására sem áll le, mindenképp végigfut.

Az `S`, `R` és `Q` válaszoknak megfelelő hatást nemcsak futás közben, a billentyűzetről érhetjük el, hanem az input file-ból is. A program válaszának megfelelő `\scrollmode` `\nonstopmode` és `\batchmode` parancsok kiadásától kezdve futás közben éppen úgy viselkedik a program, mintha a megfelelő

válaszokat adtuk volna egy hibaüzenetre. Ha pedig az `\errorstopmode` parancsot adjuk ki valahol az input file-ban, akkor a program visszaáll az eredeti, alapértelmezés szerinti állapotába, azaz megint megáll minden hiba után.

8.3. A leggyakoribb hibák

Ebben a szakaszban felsoroljuk a TEX leggyakoribb hibaüzeneteit, azok jelentését, és a TEX reakcióját a különböző lehetséges válaszokra, ábécé sorrendben.

`! Counter too large.`
Szó szerinti jelentése: Túl nagy paraméter. Azaz a program valamely paraméternek olyan értéket talált, amit már nem tud kezelni. Ez elég valószínűtlen, ha mégis előfordulna, majdnem biztos, hogy sajtóhiba okozta.

`! Double subscript.`
A program futása során matematikai módban dupla alsó index jelölést talált, azaz valami `$x_1_2$` típusú szövegrészletet. Mivel ennek nem tudja a pontos értelmét meghatározni, továbbengedése esetén `$x_1{}_2$`-ként értelmezi, azaz x_{12}-t fog kinyomtatni. Az x_{2_3}-hoz `$x_{2_{3}}$`-t kell megadni az input file-ban.

`! Double superscript.`
A program futása során matematikai módban dupla felső index jelölést talált, azaz valami `$x^1^2$` típusú szövegrészletet. Mivel ennek nem tudja a pontos értelmét meghatározni, továbbengedése esetén `$x^1{}^2$`-ként értelmezi, azaz x^{12}-t fog kinyomtatni. Az x^{2^3}-hoz `$x^{2^{3}}$`-t kell megadni az input file-ban.

`! Extra alignment tab has been changed to \cr`
Egy táblázatban a definiált oszlopok számához képest túl sok & jelet adtunk meg egy sorban. Ha nem reagálunk a hibaüzenetre, a program egyszerűen kicseréli egy `\cr`, azaz sorvége parancsra az extra jelet. Ez általában nem a legszerencsésebb megoldás, hiszen a hiba a legtöbb esetben vagy abból

fakad, hogy véletlenül tényleg eggyel több oszlopot adtunk meg esetleg a begépelendő távolság elnézésével, vagy véletlenül a szükségesnél kevesebb oszlopot definiáltunk, vagy pedig elfelejtettünk egy `\cr` jelet kitenni. Minden esetben a TEX reakciójánál komolyabb javításra van szükség, de kis odafigyeléssel nem nehéz megtalálni a hibát.

`! Extra }, or forgotten $.`
A szó szerinti jelentése, hogy a program egy felesleges } vagy $ jelet talált valahol. Általában azt jelenti, hogy elfelejtettük egy {, $ vagy $$ párját kitenni. Amilyen egyszerű a hiba, olyan nehéz időnként megtalálni, mivel a hiba megtalálása rossz esetben sokkal később történhet, mint ahol a hiba valójában van. Mindenesetre mindig visszafelé kell keresni. A válasz hiányában a program törli a megtalált, szerinte felesleges } jelet. Ez néha valóban a jó megoldás, de előfordulhat, hogy tényleg egy (vagy kettő) $ hiányzik.

`! Extra }, or forgotten \endgroup.`
Nagyon hasonló az előző hibaüzenethez, az okok is ugyanazok lehetnek. Előfordulhat azonban, hogy egy környezetet nem a $ jellel, hanem valamely mással kellett volna lezárnunk, és azt felejtettük el. Válasz hiányában a program most is a } jelet törli.

`! Fatal format error; I'm stymied`
Az `fmt` formátum file, amit a programnak megadtunk, nem jó. Feltehetően egy másik TEX verzióval készült. A másik lehetőség, amely főleg a MicroTEX programcsomag használata esetén fordulhat elő, hogy a felhasznált `tex.exe` file már tartalmaz egy formátumot, de mi mégis be akarunk tölteni egy másik formátum file-t a `tex &form` mintájára.

`! Font ... not loaded: Not enough room for left.`
A szövegben több betűtípust szeretnénk felhasználni, mint amennyit egyszerre a program be tud tölteni. Ha nem csináltunk valami nagyon nagy hibát, akkor a hibaüzenet oka csak a benne meghatározott tény lehet. A kipontozott rész helyén megadott betűtípust a program már nem tudja betölteni. Ilyen esetekben tanácsos a szöveget akkora részenként feldolgozni, hogy egy-egy részben a szükséges betűtípusok száma ne legyen túl nagy. Másik lehetőség, hogy a felhasznált `tfm` file-ban definiáljuk a szükséges fontok egy részét, és utána elkészítjük belőle megint az `fmt` típusú formátum-file-t.

Az ebben már definiált fontokat nem kell az input file-ban megint definiálni, így több helyünk marad más fontok definiálására.

`! I can't find file '...'.`
A program nem találja a kipontozott részben megadott file-t, amire pedig szüksége lenne. Majdnem biztos, hogy egy `\input` utasítással adtuk meg a kijelölt file-t, de a program nem találja valamiért. Általában vagy elfelejtettük egy nem tex kiegészítőjű file kiegészítését megadni, vagy elgépeltük a file nevét, vagy pedig valamiért hiányzik a file az adott alkönyvtárból. A hibaüzenet után a program mindig kiírja a következő sort:

`Please type another input file name:`

ami arra szólít fel bennünket, hogy adjuk meg egy másik, az általunk helyesnek vélt file nevét, amit be kell olvasnia. Ebben az esetben a program egy csökönyös szamárhoz hasonlítható, nem elégszik meg a hibajavításkor máskor felhasználható lehetőségek egyikével sem, csakis egy olyan file nevét hajlandó elfogadni, amelyiket be tudja olvasni. Ha valamely okból ilyen végképp nem áll a rendelkezésünkre, adjuk meg a DOS `nul` file nevét, ami egy nem létező, fantom üres file-t jelöl, de amit a TEX elfogad. Ha a file, amit be szeretnénk olvastatni, létezik, de véletlenül egy másik alkönyvtárban van, az input file meghatározásakor használhatjuk a szokásos DOS parancsokat annak meghatározására.

Még egy, ritkán előforduló, de szinte kideríthetetlen oka lehet ennek a hibaüzenetnek. Ha valamiáért túlságosan sok file-t szeretnénk egymásba ágyazva beolvastatni és kinyitva tartatni, az így megadott file-ok száma meghaladhatja a DOS által egyszerre kezelni tudott nyitott file-ok számát. Ilyenkor a file fizikailag jelen van ott, ahol mi gondoljuk, de a DOS nem engedi a TEX számára a megnyitását, így a program nem érzékeli, és ezt a hibaüzenetet adja. A probléma megoldása: vagy kevesebb file-t használunk egyszerre, vagy — ha értünk hozzá — a DOS `config.sys` file-jában nagyobbra állítjuk be az egyszerre nyitva tartható file-ok számát.

`! I can't go on meeting you like this.`
A program nem képes folytatni a futást, mert valamely korábbi hiba teljesen összezavarta. Ritka hibaüzenet, ha mégis előfordul, javítsuk ki sürgősen az addig megtalált hibákat.

! Illegal parameter number in definition of ... .
A kipontozott részen megadott perancs definíciójában nem megengedett paraméterjelet használtunk. Ez előfordulhat a \def\semmi#1{#2} típusú hibák esetén, amikor elírjuk a paraméter számát, vagy egyéb hasonló esetekben. Kicsit bonyolultabb hibát jelent, ha jóval a hiba megtalálása előtt van a kérdéses definíció, amit elfelejtettünk egy } jellel lezárni.

! Illegal unit of measure (pt inserted)
Az előző szakaszokban már ismertetett hibaüzenet. Jelentése: nem megenegedett mértékegységet használunk. Ha egy

Missing number, treated as zero

hibaüzenetet követ, annak a hibának a részét jelöli (lásd később). Egyébként a már leírt hibát jelzi, kijavítása is a leírtak szerint történik. Leggyakoribb oka az lehet, hogy a 0 hosszt O-nak írjuk mondjuk Opt helyett.

! Interruption
A programot leállítottuk a szokásos (Ctrl)–C vagy (Ctrl)–(Scroll Lock) billentyű-kombinációk alkalmazásával.

! Misplaced alignment tab character &
Az & jel valahol ott szerepel, ahol nem kellene. Ezt a jelet csak a táblázatokban használhatjuk. Ha nem adunk rá választ, a program figyelmen kívül hagyja. Ha & jelet szerettünk volna kiíratni, használjuk a \& parancsot. Ha pedig úgy gondoljuk, jó helyen van az & jel és nem értjük a hibaüzenetet, ellenőrizzük, nem használtunk-e egy } jelet valahol előtte, amely lezárja egy környezet érvényességét.

! Misplaced \cr
Teljes mértékben hasonló az előző hibához. Valamiért rossz helyen használtunk egy \cr parancsot.

! Missing control sequence inserted
A hibát az okozta, hogy egy definíciót \def semmi módra adtunk meg, azaz a program nem találja a parancsnevet a \def parancs után. Válasz hiányában egy nem létező, ún. \inaccessible parancsnevet illeszt a kérdéses helyre, ami által a definíció nem sok értelmet nyer, de talán még egy ideig futhat a program.

`! Missing number, treated as zero.`
A program ezen a helyen egy számot vár, amit valamiért elfelejtettünk kirakni. Ha nem adunk választ a hibaüzenetre, nullának fogja tekinteni. Amennyiben az adott helyen hosszmértéknek kellenene állnia és a mértékegységet is elfelejtettük, a már ismert hibaüzenettel ezt is jelzi majd a program.

`! Missing { inserted`
`! Missing } inserted`
A program egy szerinte hiányzó { vagy } jelet illeszt a megadott helyen az inputba válasz hiányában. Nagyon nehéz meghatározni a hiba okát, általában jóval a hibaüzenet előtt található. Ilyenkor általában hasznos dolog megnézni a létrehozott szöveget, abból sok hiba kiderülhet.

`! Missing $ inserted`
A program úgy gondolja, hogy egy $ jel hiányzik, és válasz hiányában el is helyezi. Valóban egy ilyen jel hiányzik valahonnan, de nem biztos, hogy pont az adott helyről. Általában akkor fordul elő ez a hiba, amikor elfelejtjük a csak matematikai módban használható parancsok előtt kiadni a matematikai módba váltás $ jelét. Ha annak a párját, a metematikai módot lezárót sem raktuk ki, számíthatunk egy másik hibaüzenetre is. Másik lehetséges oka, hogy a lezáró $ jelet nem raktuk ki és egy üres sort, vagy \par utasítást talált a program.

`! Not a letter.`
Egy \hyphenation utasításon belül (amellyel a kivételes elválasztású szavakat adhatjuk meg) olyan jelet adtunk meg, amit a program nem tekint betűnek. Ha a plain formátumot használjuk, akkor csak az angol ábécé betűi a betűk. Az mplain formátum felhasználása esetén csak a magyar ábécé betűit szabad felhasználni ilyen helyen.

`! Paragraph ended before ... was complete.`
Erre a hibára már utaltunk az előző fejezetben. Általában egy

`Runaway argument?`

üzenet is megelőzi, és azt jelenti, hogy egy parancs változójában egy új bekezdést definiáltunk, amit nem lenne szabad.

`! TeX capacity exceeded, sorry.`
Ez egy nem túl sok jót jelentő hibaüzenet. Azt jelenti, hogy a TEX valamely, az üzenet további részében meghatározott memória-részlete betelt. Persze az esetek többségében nem az okozza az üzenetet, hogy a programnak tényleg kicsi lenne a memóriája a szöveg feldolgozásához. A következő esetek lehetségesek:

`buffer size` Túl hosszú sorokat adtunk meg az inputban a programnak, állítsuk rövidebb sorhosszra a szövegszerkesztőt. Másik lehetséges oka a hibaüzenetnek az lehet, hogy a könyv későbbi részében leírt `matex` makrócsomag szakaszokat definiáló parancsaiban túl hosszú címeket adtunk meg.

`exception dictionary` Túl sok elválasztást adtunk meg a `\hyphenation` parancsban kivételként.

`hash size` Túl sok parancsot definiálunk.

`input stack size` Mint egy előző hibaüzenet esetén is, túl sok input file-t szeretnénk egyszerre kinyitni. Ebben az esetben viszont már nem segít a DOS parancs, a TEX beépített korlátai miatt nem bír több file-t kezelni egyszerre.

`main memory` A program nem bírta a memóriában tárolni a felhalmozódott információt. Leggyakoribb oka, hogy nem tördeljük bekezdésekké a szöveget, a program nem talál lapemelésre alkalmas helyet, és így előbb-utóbb tényleg betelik a memóriája.

A felsoroltakon kívül még sok egyéb memória-részlet is betelhet, de ezekkel itt nem foglalkozunk. Ha ilyen hiba előfordulna, úgyis egy szakértőt kell megkérni, hogy segítsen.

`! Text line contains invalid character.`
Az input file-ban olyan jelek vannak, amiket nem szabadna tartalmaznia. Mivel a szokásos IBM billentyűzetről ilyet nem tudunk megadni a `plain` formátumnak, illetve a `maek.com` program használatával ilyet nem tudunk megadni a `mplain` formátumnak, ilyenkor biztosan a felhasznált szövegszerkesztő a hibás. Nézzük meg a DOS type parancsa segítségével a hibás input

file-t és azonnal fel fog tűnni a hiba. A megoldás csak a szövegszerkesztő kicserélése lehet.

`! That makes 100 errors; please try again.`
A program 100 hibát talált és feladta a futást. Javítsuk ki a megtalált hibákat és futtassunk megint.

`! Undefined control sequence.`
Már jól ismert hibaüzenet. Olyan parancsnevet használunk, amelyet nem definiáltunk előzőleg. Ha nem adunk választ, a program egyszerűen nem veszi figyelembe ezt a parancsot. A leggyakoribb ok természetesen az lehet, hogy elgépeltük egy parancs nevét.

`! Use of ... doesn't match its definition.`
A kipontozott részen egy parancs neve található, amelyet nem olyan módon használunk, ahogy a program elvárná tőlünk. Általában nagyon ritkán fordul elő, kivéve, ha magyar betűket is használunk a parancsnevekben. Az ilyen esetek okát az előző fejezeteben ismertettük. Aki azt nem érti, de figyelmeztetésünk ellenére is használ ilyen parancsneveket, az megérdemli sorsát. A hibát természetesen úgy javíthatja ki, hogy a parancsnévből törli a magyar betűket.

`! You can't use 'macro parameter character #' in ... mode.`
A # jelet használtuk valahol a programban, holott azt csak makró utasítások paramétereinek a jelölésére használhatjuk. Ha a # jelet szeretnénk kiíratni, használjuk a \# parancsot.

A hibaüzeneteken kívül a program még figyelmeztető üzeneteket is adhat. Ezek azonban az esetek többségében arra vonatkoznak, hogy a program túl sűrű vagy ritka, illetve túl hosszú sorokat vagy oldalakat talált, amelyeket már leírtunk a 4. fejezetben. Az esetleges csúnya sorok kijavításának módozatai is ott szerepelnek.

A szakasz befejezéseként megadjuk még azoknak a lehetséges szövegeknek a rövid listáját is, amelyek a hibaüzenetek részeként,a TEX-nek a hibában előforduló egyes részletekre vonatkozó viselkedését adják meg.

`<argument>`
Az utána következő részlet egy parancs változójában szerepel.

`<inserted text>`
Az utána következő részletet a program illesztette be, az nincsen az input file-ban.

`<output>`
Az utána következő részlet a program ún. output rutinjának a része, azaz akkor hajtja végre a TEX, amikor a program a feldolgozott szöveget kiírja az outputba.

`<recently read>`
Az utána következő részletet olvasta be legutoljára a program.

`<to be read again>`
Ami utána következik, azt még nem dolgozta fel a program, habár már beolvasta. Ha azonban nem adunk ellenkező értelmű utasítást, még egyszer be fogja olvasni, és természetesen értelmezi is majd.

8.4. Bonyolultabb hibák megtalálása

Sajnos minden óvatosság ellenére is előfordulhat, hogy nehezen megfejthető hibát találunk az inputban. Ezek egyik oka lehet, hogy egy korábbi, még ki nem javított hiba olyan általunk nem látott károkat okozott, ami akár sokkal később is hatással lehet. Ezért ha valamely hiba okát nem találjuk, mindig javítsuk ki az előtte előforduló hibákat, és próbálkozzunk újra a program futtatásával. Ha szerencsénk van, az eddig érthetetlennek látszó hiba is megoldódik. Természetesen számolni kell azzal a lehetőséggel is, hogy ez a módszer sem oldja meg a problémát. Ilyenkor egy másik, már említett próbálkozás lehet, hogy arra próbáljuk kényszeríteni a programot, minél több oldalt hozzon létre az outputból, beleértve azt az oldalt is, ahol a megfejthetetlen hibát találta. A feldolgozott szövegre tekintve az esetek nagy többségében sokkal inkább megtalálható a hiba valódi helye.

Mint azt már tapasztalhattuk, és részben le is írtuk, a program sokféle információ megadásával (mind a képernyőre, mind pedig a `log` file-ba) segíti a hibakeresés, illetve programkövetés feladatát. Mindig kiírja például a már

létrehozott oldalak oldal számát [és] jelek közé. Így, ha egy hiba kijelzése után egy újabb oldalszám megjelenik a képernyőn, szinte biztosak lehetünk benne, hogy a hibás rész már az outputban is jelentkezik. Ettől eltérő eset csak akkor fordulhat elő, ha a hiba annak a bekezdésnek a belsejében van, amely az oldal alján megtörik, és a bekezdésnek a hibát tartalmazó része már a következő oldalra esik. Ekkor még egy oldalt ki kell várni, hogy a hiba az outputban is megjelenjék.

Az output megvizsgálása természetesen nemcsak a hibák kijavításakor szükséges, hanem akkor is, ha a program hibát nem találva lefut és létrehozza a szöveget. Hiszen ilyenkor csak abban lehetünk biztosak, hogy a TEX nem talált hibát, abban viszont nem, hogy mindent úgy is állított elő, mint ahogy azt előre elterveztük. Így aztán igen nagy a valószínűsége, hogy tipográfiai hibák még bőven lesznek a szövegben. Ezek egy részét igen könnyű kijavítani, mások viszont nehezebb feladatot jelenthetnek, mint a program által jelzett hibák. Általános szabályt lehetetlen adni az ilyen típusú hibákra. Segítségként azt mondhatjuk, hogy a következő bekezdések az ilyen hibák kijavítására is vonatkoznak. Még egy viszonylag általános hibaforrás: ellenőrizzük, hogy a szerintünk nem helyesen működő utasítást nem zártuk-e olyan környezetbe, amely már nem tartalmazza a kívánt részt. És ne feledjük, hogy a külalakot befolyásoló sok utasításnak nem a kívánt hatás helyén, hanem a bekezdés, vagy oldal lezárásakor kell érvényben lennie, illetve az akkor érvényben levő paraméter-értékeket használja a program. Amennyiben ilyen jellegű problémára gyanakodhatunk, próbáljuk a hibakereséséhez segítségül venni a 4. fejezetet.

MTEX A magyar ékezetes betűk sajnos újabb nehézséget jelentenek a hibakereséskor. A legnagyobb baj inkább formai, mint lényegi: a képernyőre való kiíráskor már nem olyan módon szerepelnek, mint az input fileban, hanem a meglehetősen fura ^^F[2] vagy hasonló formában. Remélhetőleg ezt előbb-utóbb meg lehet majd szokni, de addig sajnos nehezebben olvashatóvá teszik a hibaüzeneteket. Ez azonban nem befolyásolja semmiféle káros módon magának a programnak a futását. A másik, elég sokszor nem hangsúlyozható eltérésük az angol ábécé betűitől abban áll, hogy nem — illetve csak nagy szakértelemmel és óvatossággal — használhatók parancsok neveiben. Ha ilyesmire gyanakszunk a hibaüzenetből, még egyszer nézzük át gondosan az input file-t.

[2] Ez éppen az á betű megjelenítése a képernyőn.

A következő módszer, amit az ismeretlen hiba felderítésére használhatunk, az input legyszerűsítése. Ha a hiba egy olyan nagyobb szövegben fordul elő, amelyen a programnak már lényegesen meghosszabbodik a futási ideje, akkor célszerű először egy olyan rövidebb kísérleti input file-t létrehozni, amely biztosan tartalmazza a hibás részt. Utána ebben a részben elkezdhetünk részleteket törölni egészen addig, amíg a hiba is meg nem szűnik. A törlésen kívül egyéb módon is egyszerűsíthetjük a szöveget, például a bonyolultabb struktúrák egyszerűbbre való lecserélésével, a magyar ékezetes betűk kiiktatásával, stb. Több-kevesebb kísérletezés árán egyértelműen meghatározhatjuk azt a legkisebb szövegrészletet, amely még a hibát tartalmazza. Ilyenkor már abból is sok következtetést levonhatunk, hogy melyik az az első utasítás vagy parancs, amelynek a kihagyása már megszünteti a hibát. Amennyiben ez még mindig nem lenne elegendő információ, a továbbiakban nyilván lényegesen egyszerűbb lesz a hiba megtalálása a rövidebb szövegben.

Sajnos vannak olyan hibák is, amelyeket még a fenti módszerrel sem lehet meghatározni. Ilyen például, ha egy a program által használt, de általunk nem ismert parancsnevet írunk felül. Például a TEX `plain` verziója sokszor használja az `\active` paramétert, amely jelentésére, használatára nem tértünk ki ebben a könyvben. Előfordulhat, hogy valaki éppen ennek a parancsnévnek ad új definíciót, amely szinte megfejthetetlen hibához vezethet[3], főleg az `mplain` formátum használata esetén. Ez azonban csak nagyon ritkán fordulhat elő, ugyanis egyrészt általában ezek a parancsnevek angol szavak rövidítései, és így valószínűleg nem gyakran választjuk majd őket parancsok neveinek, másrészt a makrócsomagok készítői éppen az ilyen bonyodalmak elkerülésére általában a `@` jelet is felhasználják a segédparancsok neveiben, amit mi nem tudunk megtenni. A példa inkább csak annak a bemutatására szolgál, hogy milyen elképzelhetetlen hibák is felléphetnek.

A nagyon bonyolult hibák megtalálásához a program többfajta segítséget is kínál. Nemcsak a hibákat javíthatjuk interaktívan, hanem a programot is futtathatjuk. Ha a szokásos futtatással ellentétben nem adunk meg neki input file-t, csak formátumot, akkor a program betöltődése után egy sor elejei * jellel jelentkezik be és további utasításokat vár. Ilyenkor akár soronként is begépelhetnénk a szöveget, de ennek nem sok haszna lenne, hiszen ezzel éppen a szövegszerkesztő nyújtotta előnyöktől esnénk el. Van azonban

[3] A vállalkozó kedvű olvasók megpróbálkozhatnak ezzel.

egyetlen utasítás, amit így érdemes használni, mégpedig a \show. Ez kiírja az utána megadott parancsnév definícióját, a következő példák szerint. Ha a parancs egy TEX primitív, mint például az input, akkor csak megismétli a parancs nevét, így

```
*\show\input (return)
> \input=\input.
```

Ha a parancsnév még nincsen definiálva, akkor csak ezt közli velünk:

```
*\show\semmi
> \semmi=\undefined
```

Ha viszont a parancsnév egy makróutasítást takar, akkor ezt, és a makró definícióját is a tudomásunkra hozza a program:

```
*\show\bigbreak
> \bigbreak=macro
\def\bigbreak{\par\ifdim\lastskip<\bigskipamount
\removelastskip\penalty-200\bigskip\fi}
```

Így, ha valamely parancs definíciójával kapcsolatban kételyeink vannak, használhatjuk ezt a módszert ellenőrzésre.

A másik módszer, amivel a végrehajtott parancsok megmutatására kényszeríthetjük a programot, a \tracingmacros és \tracingcommands paraméterek használata. Amennyiben az első értéke pozitív (tanácsos ilyenkor egynek választani), a program a log file-ban minden makró végrehajtása esetén elteszi annak kifejtését, míg ha a második pozitív, minden parancs — akár makró, akár primitív — végrehajtását hasonló módon dokumentálja. Az alapértelmezés szerinti beállításuk természetesen 0, de a szövegen belül bárhol átállíthatjuk őket. Ha viszont egy hosszabb szövegben végig 1-nek tartjuk az értéküket, ne lepődjünk meg, ha nagyon hosszú log file-t kapunk. Még egy dologra hívjuk fel a figyelmet: az mplain formátum a magyar betűket, mint említettük, nem betűkként, hanem parancsokként tekinti, így a fenti paraméterek akármelyikének a pozitív értéke minden egyes magyar betűt külön dokumentál a log file-ban.

Végül, sajnos ki kell térnünk arra a lehetőségre is, hogy egy hibát nem sikerül megtalálnunk, vagy kijavítanunk. Még csak azt sem állíthatjuk biztosan, hogy ez mindig a felhasználó hozzá nem értésén múlna. A TEX pro-

ramot és annak magyar `mplain` formátumát is emberek készítették, akik szintén hibázhatnak. És mivel, ahogy az az eddigiek alapján bizonyára kiderült már, ezek a programok meglehetősen sokat tudnak, és ennek megfelelően bonyolultak is, szinte biztos, hogy apró hibák maradtak bennük. Csak abban lehetünk biztosak, hogy ha egy hiba annyira bonyolult, hogy nem lehet megállapítani az okát, akkor kell lennie egy másik módszernek is arra, hogy az elvárt eredményt létrehozzuk. Ilyenkor tehát vagy ezt az esetleg bonyolultabb, de remélhetőleg működő módszert válasszuk, vagy kicsit egyszerűsitsünk az input file-on.

9. fejezet

Makrócsomagok

9.1. Általános tudnivalók

Amint azt az előző fejezetekben már leírtuk, a TEX program nem futhat magában, valamilyen makrócsomagot mindenképpen fel kell használnia. Ezek között többféle szintű létezik. Egyesek alapszinten látják el a feladatukat (közülük egyre szükség van), mások csak kisegítő szerepet játszanak. Ez utóbbiakból egyszerre többet is felhasználhatunk. Az alapszinten működő makrócsomagokat (amelyek általában viszonylag nagy méretűek) a TEX egy különleges módon futtatja le (ezt a programot hívjuk INITEX-nek). Az általa létrehozott információ sűrített formában tárolódik, és ezzel később meggyorsul a makrócsomag beolvasása. Ezek az ún. formátumcsomagok, amelyeket minden implementáció egy speciális, erre a célra fenntartott alkönyvtárban tárol és amelyeket a program a többi, általa felhasznált file-tól az `fmt` kiegészítéssel különböztett meg. A makrócsomagok által ellátott feladatok egy részét csak ezek a formátumcsomagok képesek végrehajtani, de más feladatokat is célszerű velük végrehajtatni. Az első esetre jó példa az elválasztási algoritmus megadása, amely olyan parancsokat és egyéb file-okat igényel, amelyeket csak a formátumfile-ok tartalmazhatnak. A különböző betűtípust definiáló utasításokat ugyan bármelyik makrócsomagban, vagy akár magában a feldolgozandó szöveget tartalmazó file-ban is megadhatnánk, de olyan nagy az időigénye, hogy mindenképp célszerű a leggyakoribb betűtípusokat a formátumfile-ban definiálni, és csak az azokat a konkrét file-ban, amelyeket kivételesen használunk.

Ilyen formátumcsomag lényegében kettő van használatban pillanatnyilag, az ún. plain, és annak módosított változatai, mint amilyen a magyar különlegességeket tartalmazó mplain, illetve a LaTeX makrócsomag használatához adott lplain. Persze ez a kijelentés eléggé elhamarkodott, hiszen az egyéb, gyakori és nagyobb méretű makrócsomagokat is szokás állandó használat esetén különleges módon lefuttatni, és attól fogva formátumként emlegetni. Ilyen nagyobb csomag még a főleg matematikai felhasználók számára készült AMSTeX, illetve annak egy segédprogramja, az ún. AMSPPT stílus. Az AMSTeX használatához például kifejezetten ajánlott, hogy hozzunk létre egy olyan formátumot, amely az általunk használt alapvető makrócsomag (tehát a plain vagy mplain) végén tartalmazza a \input amstex utasítást, és így kerül lefuttatásra az INITEX alkalmazásával. Utána, ha még az AMSTeX-hez ajánlott AMSPPT stílust is használni szeretnénk, azt már megadhatjuk a szövegfile elején is egy \input-tal.

Könyvünkben leírtak lényegében a plain illetve mplain formátumok felhasználása esetén érvényesek. Mint már említettük, a kettő között, néhány apróságtól eltekintve, az az eltérés, hogy az mplain megérti és felhasználja a kifejezetten magyar betűket is, és a magyar elválasztás szabályait használja. Mindkettővel felhasználható a később részletezett AMSTeX makrócsomag, de a LaTeX már speciális bánásmódot igényel. Az lplain makrócsomag tulajdonképpen a plain egy olyan változata, amely arra készíti fel a programot, hogy a LaTeX-et fogadja, de magában nem használható. Mivel a LaTeX magyar változata különböző okokból többet jelent, mint hogy az mplain-en végrehajtsuk az lplain-hez hasonló változtatást, a MaTeX programcsomag pillanatnyilag nem használható a LaTeX formátummal. Ennek ellensúlyozására hoztuk létre a MaTeX makrócsomagot, amely kétségtelenül jóval szerényebb, mint a LaTeX, de annak néhány szolgáltatását talán pótolja.

A kialakult hagyomány szerint az egyes makrócsomagok neveinek köze van a létrehozójukhoz, illeteve a forgalmazóhoz. A LaTeX csomagot Leslie Lamport amerikai szakember állította elő, míg az AMSTeX az Amerikai Matematikai Társaság (AMS) tulajdona. A MATeX makrócsomag elnevezését mindenki kedve szerint tulajdoníthatja a MAgyar TeX vagy a MAgiszter TeX rövidítésének. Az mplain megint csak a hagyományokat követve kapta nevét a lplain mintájára. Az egyetlen kivétel az itteni szabályok alól a plain, amely egy angol szó. Magyar jelentése sima, egyszerű.

Ejtsünk most néhány szót arról a bizonyos „különleges" futási módról. Amennyiben tehát a programmal nem egy meghatározott szöveget kívánunk feldolgozni, hanem valamely alapvető makrócsomagot akarunk formátum alakra hozni, akkor az INITEX-et kell futtatnunk a makrócsomagon. Ennek meghívása a konkrét implementálástól függ. Ilyenkor a program feldolgozza a makrókat, a makrócsomagban megadott betűtípusok méreteit (így azokat a további futtatásokkor már nem kell külön beolvasni), az elválasztási szabályokat, és az egész információt olyan formában tárolja, amely meggyorsítja annak további betöltését. Csak arra van szabály, hogy minek kell mindenképp az így feldolgozott file-okban lennie (pl. az elválasztási szabályoknak), olyan parancstípus nincs, amely ne szerepelhetne itt. Így aztán tetszőleges formátumcsomag végén \input-tal megadhatunk olyan általunk mindig használt egyéb makrócsomagokat, amelyeket úgyszintén ebben a speciális formában kívánunk tárolni a továbbiakban. Csak arra vigyázzunk, hogy azok tényleg a formátumfile-ok végén legyenek, ugyanis csak itt lehetünk biztosak benne, hogy a program már érti az azokban felhasznált utasításokat.

Mint láthattuk, elég sokféle módon neveztük eddig a makrócsomagokat, olykor formátumnak, olykor stílusnak. Arra már utaltunk, hogy a formátum miben különbözik a többitől. Egy formátumot nem is lehet az \input utasítással behívni, csak a program futtatásakor (tehát nem a feldolgozandó file-ban) adhatjuk meg, hogy melyiket kívánjuk használni — noha természetesen a file-t az adott makrócsomag szabályai szerint kell elkészítenünk. Vigyázat, csak olyan formátumfile-t adhatunk meg, amelyik valóban létezik a program által megkívánt formában, azaz `fmt` kiegészítéssel, az erre a célra fenntartott alkönyvtárban! A stílust nem tudjuk hasonlóan markáns jegy segítségével megkülönböztetni a többi makrócsomagtól. Általános esetben csak annyiban különbözik, hogy nem az általában szokásos `tex` kiegészítésű file-ban van, hanem egy `sty` kiegészítésűben. A `stilus.sty` file-t nem is lehet egyszerűen a `\input stilus` paranccsal meghívni, hiszen akkor a program a `stilus.tex` file-t keresné, hanem ki kell írni, hogy `\input stilus.sty`. Ettől az általános szabálytól csak a LaTeX használ kicsit eltérőeket, mint később látni fogjuk.

Inkább a feladatuk szerint különítik el a stílusfile-okat. Az általános makrócsomagok olyan parancsokat tartalmaznak, amelyek nem kötődnek egyetlen speciális kiadványhoz vagy kiadványfajtához. A stílusfile-ok inkább

egy-egy könyv vagy kötet paramétereit, jellemzőit tartalmazzák. E könyv előállításához például a vele együtt forgalomba kerülő `mplain` formátumon és MATEX makrócsomagon kívül egy `konyv.sty` nevű stílusfile-t is alkalmaztunk. Ez tartalmazta például azokat a parancsokat, amelyek az élőfej és a különleges lábjegyzet kiszedését segítették.

9.2. A plain és mplain formátumok

E két makrócsomagról szinte nem kell már semmit említenünk, hiszen az egész könyv róluk szólt. Az első megtalálható minden TEX programcsomagban `plain.tex` néven, míg a másodikat a MATEX programcsomag tartalmazza `mplain.tex` néven. A `plain` által feldolgozott angol nyelvi elválasztási szabályok a `hyphen.tex`, míg az `mplain` magyar nyelvi elválasztási szabályai az `elval.tex` programokban találhatók. Az `mplain` úgy készült, hogy minél több dolog érvényben maradjon a `plain` szabályai közül, így a lényeges eltérést valóban csak a magyar nyelvű felhasználás jelenti. Van néhány egyéb apró különbség is, amelyekre a könyvben kitértünk, mint például az `mplain` magyar „ alsó idézőjele, vagy hogy az `mplain`-ben a felkiáltójelet a `\?` parancs hozza létre.

A fentiek ellenére a két makrócsomag felhasználása szinte azonos módon történik, habár egyáltalán nem felcserélhetők. Ugyanis ha valaki angol nyelvű szöveget az `mplain` segítségével szed ki, akkor (legalábbis az angolok szerint) ugyancsak furán elválasztott angol szavakat kap. Ez azonban még csak a kisebb baj[1]. Sokkal több hibát okoz, ha valaki a könyvben leírtak szerint szedett magyar nyelvű szöveget próbál meg a `plain` segítségével feldolgozni. Ekkor ugyanis a speciásisan magyar magánhangzók helyén görög betűk és a 2. fejezetben bemutatott skandináv ligatúrák fognak megjelenni.

Mindkét makrócsomag egyformán alkalmas arra, hogy az utolsó soraiban valamely nem alapvető, de a felhasználó által esetleg rendszeresen felhasznált (akár általa írt) kisebb makrócsomagot `\input`-tal hozzácsatol-

[1] Az `mplain` megjelenése előtt a TEX-hel magyar nyelvű szöveget csak úgy lehetett előállítani, hogy a felhasználók állandó figyelemmel kisérték a program elválasztási kisérleteit és a magyar szabályoknak megfelelően módosították azokat.

junk, és így inicializáljuk azt az INITEX segítségével. Míg az általában forgalomben levő plain csomag nem tartalmaz ilyen \input utasítást a végén (tehát ha valamit be akarunk vele olvasni, azt magunknak kell végrehajtani), addig az mplain végén ott található az \input matex utasítás, amely tehát beolvassa a matex.tex programot. Ha ezt el kívánjuk kerülni, akkor ebben az esetben ki kell törölni az ezt tartalmazó sort.

9.3. A LaTeX formátum

Aki ismer más szövegformázó programokat is, az az eddigiek alapján láthatja, hogy a TeX többek között a kiemelkedő matematikai szedési lehetőségeivel különbözik a többi ilyen programtól. Ezt a tulajdonságát a plain illetve mplain formátumok részben még erősítik is azáltal, hogy sok matematikai makró definícióját tartalmazzák. Ugyanakkor a TeX nagyon rugalmasan használható (azt is mondhatnánk, programozható) szokványos szövegek szedésére is. Ezt a lehetőséget használja ki a sokrétű LaTeX formátum.

A LaTeX nem egyetlen file, hanem egy igen sok elemet tartalamzó makrócsomag. Az alapvető utasításokat a latex.tex file tartalmazza, de ezen kivül sok kis file-ban tartalmazza az egyes dokumentumok stílusfile-jait. Ezeket azért kell külön-külön kezelnie, mert a bennük tartalmazott parancsok nevei általában megegyeznek, míg a definíciók kissé különböznek. Például abban a stílusfile-ban, amelyikben az alapvető betűnagyság tízpontos, a \large parancs (magyar jelentése nagy) tizenkét pontos betűnagyságba vált. Ezzel szemben tizenkét pontos alapnagyság esetén a \large tizennégy pontos betűket eredményez.

A LaTeX inicializálásához szükségünk van a csak erre a célra készített lplain.tex programra. Ez a plain egy olyan változata, amelyből kihagyták azoknak a makróknak a definícióját, amelyeket a LaTeX nem használ, de amelyek helyett más, hasonló rendeltetésű makrókat definiáltak. Ennek megfelően a LaTeX nem egyszerűen többet tud, mint az ún. plain TeX, hanem mást is. A legnagyobb részlet, ahol nagyjából megegyezik a két formátum, a matematikai utasítások készlete, hiszen a LaTeX nem ezek továbbfejlesztésére készült.

A LaTeX ahhoz, hogy a plain formátumnál egyszerűbbé tegye az oldal külalakjának kezelését, olyan utasításokat vezet be, amelyek annak minden részletét egyszerre határozzák meg. Ezzel természetesen sokkal egyszerűbb egy-egy oldal alakjának beállítása — feltéve, hogy létezik parancs, amely éppen az általunk kívánt kombinációban teszi azt meg. Ez az a jellemzője ennek a programcsomagnak, amely nagy előnye mellett nagy hátrányt is jelent. Ha valamely általunk elképzelt dologra létezik benne egy egyszerű utasítás, akkor azt kiadva nyilván sokkal könnyebb a dolgunk, mintha a plain formátumot használnánk. Ha viszont nincs ilyen parancs, akkor általában sokkal nehezebb dolgunk van, mint a plain használata esetében. Ez a program készítőinek félig kimondott célja is — ugyanis azt szeretnék, ha a tipográfiában nem túl jártas átlagos felhasználó nem bocsátkozna felelőtlen tipográfiai kisérletekbe, hanem az általuk előre meghatározott mintákat követné. Egy amerikai — vagy általában angol nyelvű — felhasználónak ez talán meg is felel, de kevés olyan parancsa van a makrócsomagnak, ami nemzetközi érvényű. Az országok közötti tipográfiai eltérések eredményeként a LaTeX formátum nem nagyon használható Magyarországon. Valójában hazai használatához nem is átírására, hanem újra megírására volna szükség. Ezért a továbbiakban csak egy rövid ismertetést adunk fő jellemzőiről.

Az oldal külalakjának egységes kezelésén kívül — amelyet a már említett különböző stílusok, mint amilyen a könyv, cikk, jelentés, stb. segítségével old meg — automatikusan kezeli és számozza a szövegegységeket, tartalomjegyzéket és indexet készít (utóbbit csak részben önállóan), listákat, táblázatokat és ábrákat kezel. Ez utóbbit valóban egyszerűbben és sokrétűbben teszi, mint a plain formátum. Ebben nagy segítségére van, hogy a plain-nél több betűtípus kezelésére van felkészítve, sőt olyan file-ok is a rendelkezésére állnak, amelyek betűképek helyett például egyenesszakaszokat vagy kördarabokat tartalmaznak. Ezeket felhasználva egyszerűbb ábrák megrajzolására maga a LaTeX is képes, és könnyebben lehet benne megadni, hogy éppen milyen betűtípus (és nagyságot) használjon.

A sok automatikus funkciót a LaTeX a programmal együtt forgalmazott stílusfile-okon kívül egyéb segédfile-ok segítségével oldja meg. Ezeket a program minden egyes futásakor, az adott szövegfile-hoz tartozóan hozza létre, és ide gyűjti az információkat. A stílusfile-ok a LaTeX esetében kicsit eltérnek az általánostól, noha az általában használt sty kiegészítéssel rendelkeznek. A LaTeX speciális módot tart fenn ezek kezelésére, behívásuk

tulajdonképpen automatikusan történik. Mint korábban már említettük, sok makrócsomag különleges szerepet szán a @ jelnek, amely segítségével olyan, csak a makrócsomagban alkalmazott segédutasítást definiálhatunk, amelyet így még véletlenül sem használunk fel később. Éppen emiatt viszont baj származhat abból, ha ilyen makrócsomagot felkészületlenül és ismeretlenül próbálunk felhasználni. Ennek elkerülésére a LATEX a stílusfile-ok esetén automatikusan megengedi a speciális @ jelek használatát.

9.4. A MATEX makrócsomag

A LATEX által nyújtott szolgálatatások részbeni pótlására tartalmazza a MATEX programcsomag a `matex.tex` makrócsomagot, amelyet külön kérés nélkül, automatikusan beolvas és feldolgoz az `mplain` formátum. Ha tehát a függelékben leírtak szerint inicializáljuk a `mplain`-t, akkor az itt leírt utasításokat minden további nélkül használhatjuk.

A MATEX, azon kívül, hogy tudja a `mplain` speciálisan magyar szolgáltatásait, a `\kelt` parancs hatására kiírja a parancs által meghatározott helyre az aznapi dátumot, a magyar helyesírás szabályainak megfelelően. Ehhez természetesen szüksége van arra, hogy a DOS is tudja azt, azaz vagy legyen a számítógépben beépített óra, vagy a gép bekapcsolásakor a felhasználó adja meg. Ezen kívül három csoportba oszthatjuk a programcsomag szolgáltatásait: automatikus szövegegység címzés, azokból összeállított automatikus tartalomjegyzék készítés és félig automatikus tárgymutató készítés, valamint, mivel az indexnél úgyis szükség van rá, két hasábba történő tördelés.

A MATEX négy különböző szintű szövegegységet képes kezelni: fejezetet, szakaszt, pontot és alpontot. Ezek közül a fejezetet mindig automatikusan új, páratlan oldalszámú, azaz jobboldali oldalon kezdi, míg a többit folyamatosan, mindössze az egységtől függő távolsággal az előző egységtől elválasztva. Az egységeket automatikusan számozza is, a fejezeteket egytől, a szakaszokat a fejezeteken belül 1.1.-től, és így tovább. A szakasztól lefelé csak az egység sorszámát és címét írja ki, míg a fejezetek esetében kiírja a fejezet szót is. Ahelyett, hogy itt szemléltetnénk ezeket a parancsokat, mindenki lapozzon tetszése szerint bármelyik fejezethez, szakaszhoz, stb.

ebben a könyvben. A \szakaszm paraméter segítségével beállíthatjuk, hogy az egyes szövegegységek milyen mélységig legyenek számozva. Amennyiben ezt egyre állítjuk, csak a fejezetek kapnak önálló számot, ha kettőre, a szakaszok is (ez az alapbeállítás), és így tovább. Ha nullára állítjuk be, akkor a fejezetek sem kapnak számot, és így a fejezet kiírás is eltünik, csak a fejezet címét tartalmazó sor marad meg.

Ha valamilyen okból meg szeretnénk változtatni a szövegegységek automatikus számozását, akkor azt a \fejezsz, \szaksz, \pontsz és \alpontsz paraméterek megfelelő beállításával tehetjük meg. Ezeket mindig azon szövegegység előtt kell kiadni, aminek nem az automatikusan kialakított számot szeretnénk adni. A paraméter értéke pedig mindig eggyel kevesebb legyen, mint az a szám, amit az adott szövegegységnek szánunk. Ha tehát az 5. fejezetben valamely szakasznak a valójában nem őt megillető 5.7 számot szánjuk, akkor előtte adjuk ki a \szaksz=6 parancsot — feltételezve, hogy a \fejezsz paraméter értéke a fejezetnek megfelelő. Ha viszont az 5.7.2 pontszámhoz annak minden elemét be kell állítanunk, akkor a \fejezsz=5, \szaksz=7, \pontsz=1 parancsokat szükséges kiadnunk.

Ezek után már csak arra van szükségünk, hogy a szövegegységek címeit előállító parancsokat megismerjük. A \fejezet, \szakasz, \pont és \alpont parancsok mind egyetlen változóval rendelkeznek, ami az adott, a parancs nevében meghatározott szövegegység címe lesz. A parancs változója, tehát a szövegegység címe nemcsak szöveget, hanem egyéb TEX utasításokat is tartalmazhat. Így a címekben akár matematikai képletek, vagy egyéb nem szokványos jelek is szerepelhetnek. Az egyetlen kivétel a kezdő és nyitó kapcsos zárójel, amelynek használatát el kell kerülni ezeken a helyeken. Ezek a parancsok automatikusan végrehajtanak minden, a cím kiszedéséhez tartozó utasítást, még egy megfelelő segédfile-ba is elteszik a címeket, amennyiben közöljük a programmal, hogy tartalomjegyzék készítésére is fel kívánjuk használni. Nem szedik ki viszont ezek a parancsok automatikusan az élőfejet és a lábsort, mivel azt minden felhasználó másként kívánhatja. Tehát ezek használatára a 4. fejezetben leírtak érvényesek, illetve a 7.6 szakaszban leírtak szerint változtathatunk a tartalmukon.

Az szövegegységek címeit megadó parancsok és az őket követő első szövegsor között mindig hagyjunk ki egy üres sort. Ennek hatására a program a szöveget egy új bekezdésbe fogja szedni (habár nem ez az oka az üres sor kihagyásának). Ha — mint az néha szokásos — a legelső bekezdést nem

akarjuk beljebb kezdeni a fejezetben, szakaszban, stb., akkor használjuk a \noindent parancsot. Amennyiben két szövegegységcím kerül egymás alá (mondjuk egy szakasz és egy pont címe), nem szükséges üres sort kihagyni köztük.

Ha a tartalomjegyzéket is ki szeretnénk szedetni a programmal, akkor a file legelején adjuk ki a \tartkesz parancsot. Ez arra utasítja a programot, hogy amennyiben címekkel találkozik, azokat rakja el egy erre a célra készített, a file nevével megegyező nevű, de tar kiegészítésű file-ba, illetve az előző futás alatt létrehozott ilyen file-t konvertálja a trt kiegészítésű file-ba. Ha a program találkozik valahol a \tartjegyz paranccsal, akkor az általa meghatározott helyen, új páratlan oldalon kezdve kiszedi minden további utasítás nélkül a tartalomjegyzéket, annak címével együtt. A tartalomjegyzék külalakját megintcsak ebben a könyvben ellenőrizhetjük. A file elején beállíthatjuk a \tartm paramétert, aminek értéke megmondja, hogy milyen mélységig kerüljenek a szövegységek a tartalomjegyzékbe. Alapértelmezése kettő, azaz a fejezet- és szakaszcímek kerülnek csak a tartalomjegyzékbe. Ha viszont négyre állítjuk be, akkor a pontok és alpontok címei is szerepelni fognak.

A tartalomjegyzék mindig a trt kiegészítésű file-ból készül, ami viszont mindig azokat az információkat tartalmazza, amelyek a program előző futásából származnak. Ezek viszont általában az egyes futások után megváltoznak, hiszen egy fejezet kiegészítése azt is jelenti, hogy a következő fejezet néhány oldallal később kezdődik. Ha tehát tartalomjegyzék készítésére is használjuk a programcsomagot, helyes a \tartkesz és \tartjegyz parancsokat már a file elkészítésének a legelején kitenni, és minden futtatáskor alkalmazni. Aztán, ha már egyébként teljesen hibátlan a szöveg, célszerű még egyszer lefuttatni a programot hogy az utolsó változtatások is megjelenjenek a tartalomjegyzékben. Az nem elegendő, ha csak a \tartkesz parancsot alkalmazzuk a munka legelejétől, és a \tartjegyz-et csak a legvégén, hiszen a 3. oldalra kinyomtatott 2 oldalas tartalomjegyzék azonnal hátrább rak mindent 2 oldallal.

A tárgymutató készítése már nem tehető meg az előzőekhez hasonlóan, teljesen automatikus módon. Ha a file elején kiadjuk az \indexkesz parancsot, akkor a program minden \ind parancs hatására kiírja egy külön erre a célra fenntartott file-ba egy-egy külön sorba az \egys parancsnevet, majd a parancs változóját, végül pedig annak az oldalnak a számát, amelyen a

parancs található. Ha tehát valamit be szeretnénk tenni a tárgymutatóba, akkor azt az \ind parancs változójában kell megadni. *Figyelem, amit a parancs változójában megadunk, az csak a speciális file-ba kerül bele, de nem kerül szedésre a szövegben.* Így ha ugyanazt a dolgot szeretnénk a szövegben is kiszedetni és a tárgymutatóban is megjelentetni majd, akkor kétszer kell leírnunk — egyszer magában és egyszer mint az \ind parancs változóját. Ha tehát a program azt találja ezen az oldalon, hogy \ind{index}, akkor a tárgymutató készítéséhez használt, a file nevével megegyező nevű, de ind kiegészítésű file-ba az egy külön sorba kerül, hogy

```
\egys index 228
```

Az \ind parancs változójába, hasonlóan az egyes szövegegységcímekhez, nemcsak szövegrészlet, hanem tetszőleges TEX parancsok is kerülhetnek, kivéve a kapcsos zárójeleket.

Két fontos dologra hívjuk fel a figyelmet az \ind paranccsal kapcsolatban. Az egyik, hogy habár a parancs igyekszik eltüntetni a kiadási helye körüli felesleges szóközöket, azaz a

```
szöveg \ind{index} folytatás
```

szövegrészlet hatására csak egy szóköz jelenik meg a szöveg és a folytatás között, ez nem minden esetben sikerül. Ezért a legbiztosabb megoldás, ha az \ind parancsot mindig az őt körülvevő szavak egyikéhez (de csakis egyikéhez) ragasztjuk. A másik fontos dolog, hogy a ind file-ban a magyar ékezetes betűk a TEX által kiválasztott módon, azaz a ^^F mintájára jelennek meg. A program számára ezek az alakok ugyanúgy érthetők, mintha a ^^F helyén a neki megfelelő á állna. Ha a felhasználót zavarja ez a jelölésmód — mondjuk a file szerkesztésében — akkor az alább szereplő marend program segítségével átalakíthatja a magyar betűket is eredeti alakjukra.

Az \indexkesz parancs hatására tehát létrejön az ind kiegészítésű file. A \tmut parancs viszont egy, a munkafile nevével megegyező nevű, de inx kiegészítésű file-ból hozza létre a tárgymutatót. A \tmut parancsnak két változója van, amelyek közül az első a tárgymutató címe (lehet tárgymutató, vagy index, stb.), míg a második, amely általában vagy nincs jelen, vagy hosszabb, az a szövegrészlet, amit a cím után, de még a tárgymutató szedése előtt ki szeretnénk szedni. Ez a szöveg még a szokásos módon kerül szedésre, míg a tárgymutató maga az erre használatos két hasábos módon.

Amennyiben tartalomjegyzéket is készítünk, akkor abban a tárgymutató is szerepelni fog, mégpedig olyan formában, mintha egy fejezet lenne. Egyébként a címe is a fejezetnek megfelelően kerül szedésre, de természetesen fejezetszám nélkül, és ugyanúgy az első szabad jobboldali lapon kezdődik, mint a fejezetek.

Azt a file-t, amelyből majd a tárgymutató készül, tehát aminek neve megegyezik a fő file nevével, de a kiegészítője inx, a felhasználónak kell létrehoznia az itt megadott szabályok szerint. Amennyiben ilyen file nincs, a program még mindig kiszedi a tárgymutató címét és a \tmut parancs második változójában esetleg megadott szövegrészletet, de figyelmezteti a felhasználót arra, hogy nem találta a megfelelő inx file-t. Ennek ellenére, ha tartalomjegyzéket és tárgymutatót is szeretnénk készíttetni a programmal, célszerű a \tmut parancsot már a munka elejétől alkalmazni (és ilyenkor \tartjegyz parancsot is *mindenképpen*), hogy a tárgymutatónak megfelelő cím is jelentkezzen a tartalomjegyzékben. Ugyancsak célszerű a \ind parancsokat is már a szöveg létrehozásakor, vagy legkésőbb az első javításnál megadni, hogy aztán amikor már majdnem készen vagyunk, a file elején a \indexkesz parancsot kiadva a szükséges információkat kigyűjthessük.

Ha tehát már mindennel készen vagyunk, és az ind file is a rendelkezésünkre áll, akkor készítsük el a \inx file-t. Ebben minden sor az

```
\egys index 229
```

sorhoz hasonló legyen, csak az \egys parancs helyett az \alegys vagy az \alalegys paranccsal is kezdődhet. Ezek a parancsok azt határozzák meg, hogy az utánuk (az oldalszámmal vagy számokkal együtt) megadott indexegység az előtte szereplő legutolsó főegység, illetve alegység alegysége, és természetesen ennek megfelelően lesz kiszedve is. Ezeken a parancsokon kívül, amelyek tehát a tárgymutató egyes elemeire vonatkoznak, tetszőleges más TEX parancsot is alkalmazhatunk az inx file-ban. Így ha a különböző betűvel kezdődő egységek közé a tárgymutatóban szokásos kis extra függőleges távolságot szeretnénk kitenni, nyugodtan alkalmazhatjuk a \vskip utasítást. Csak azt nem szabad elfelejtenünk, hogy amit itt szedünk, az fele szélességben, de két hasábban lesz szedve. A program tehát az alábbiakhoz hasonló dolgokat vár a \inx file-ban:

```
\egys főegység 3,5,6
\alegys alegység 4,5,7,12
```

```
\alalegys az alegység alegysége 5
\egys más 5
\egys megint más 12,34
\egys és megint 23
\egys újabb egység 2,4,8
\alegys ennek is van alegysége 3
```

aminek hatására kiszedi, hogy

főegység 3,5,6
alegység 4,5,7,12
az alegység alegysége 5
más 5
megint más 12,34
és megint 23
újabb egység 2,4,8
ennek is van alegysége 3

Ahhoz azonban, hogy a MATEX által létrehozott ind file-ból a fenti mintájú file legyen, sokat kell dolgoznunk vele. Többek között, az itt bemutatottakkal ellentétben, ábécé sorrendbe kell rendezni az egyes egységeket, azon belül az alegységeket, és még azokon belül az alalegységeket. Aztán a többször szereplő szavakat egyetlen egység alá kell hozni, újra és újra át kell gondolni a tárgymutató szerekezetét, stb. Ha zavar a magyar ékezetes betűk fircsa megjelenése, azokat is át kell alakítanunk. Ezen feladatok közül két olyan van, amelyet a számítógép is meg tud oldani,a betűk átalakítása és a sorbarendezés. Ezért adjuk a programcsomag elemeként a `marend.exe` programot, ami a

```
marend valami.inx
```

módon alkalmazva a fentiek szerint rendezi a `valami.inx` file-t, és a magyar betűket ismét az ismerős alakjukban jelenteti meg a képernyőn. Vigyázzunk, hogy a rendezéskor a file még ne tartalmazzon semmilyen TEX parancsot az `\egys`, `\alegys` és `\alalegys` parancsokon kívül és minden egyes egység külön sorban szerepeljen. Az viszont már a felhasználó szabad elhatározásától függ, hogy a rendezést a inx file létrehozásának elején vagy végén, esetleg a létrehozás során többször is használja.

Egyetlen parancspár van még hátra, a `\ketoszlop` illetve `\ketoszlopvege` parancsok. Az első hatására a program az utána következő szöveget a teljes oldalszélesség helyett csak annak felére szedi, de két hasábba. A második, értelemszerűen, visszaállítja az alapállapotot. Ami a két parancs előtt szerepelt, az a szokásos módon kerül az oldalra, majd a két parancs között szereplő szövegrészlet két hasábban kerül szedésre, utánuk pedig megint a szokásos módon folytatódik a szöveg.

Célszerű, ha ezeket a parancsokat csak bekezdések között adjuk ki, azaz egy bekezdést mindig vagy teljes egészében két hasábba szedünk, vagy teljes egészében egy hasábba. Még ennek ellenére is előfordulhat, hogy a kis szélesség vagy a kevés két hasábba szedett anyag hatására a program a szokásosnál nehezebben találja meg a soremeléseket vagy a hasábok közötti váltás helyét, és így `Overfull` vagy `Underfull` figyelmeztető üzenetet kapunk. Sőt, ilyen üzenet ezen parancsok hatására (és velük együtt az őket felhasználó `\tmut` parancs hatására is) még akkor is előfordulhat, ha utána a kinyomtatott oldalt megvizsgálva nem találjuk, hogy mi is okozhatta. Ilyen esetekben nem kell figylembe venni ezekez az üzeneteket.

A MATEX makrócsomag használatakor létrejön még az érvényes alkönyvtárban egy `kis` kiegészítésű file is, ahol a `kis` a kisegítő rövidítése. Erre a programnak csak futás közben van szüksége, tehát bármikor nyugodtan kitörölhetjük.

9.5. Az AMSTEX és VANILLA makrócsomag

A TEX-ben írt makrócsomagokat osztályozhatjuk a szerint is, hogy milyen alap formátumfile-t használnak. A LATEX makrócsomagról elmondtuk, hogy saját formátum file-ja van, az `lplain`. A kisebb-nagyobb makrócsomagok azonban általában az eredeti — TEX programmal együtt forgalmazott — `plain`-re épülnek. A `plain`-re épülő makrócsomagok között általában nincs komoly összeférhetetlenség, használhatjuk őket együtt is. Említettük, hogy a MATEX makrócsomagnak külön formátumfile-ja van, az `mplain`. Ez azonban olyan módon különbözik a `plain`-től, hogy ez a makrócsomagokat egyáltalán nem, vagy csak nagyon kis mértékben zavarja. Ez igaz az

egyik legnagyobb és legfontosabb makrócsomagra az AMSTEX-re is. Ez a makrócsomag az Amerikai Matematikai Társulat támogatásában készült el és elsődleges célja a matematikai kiadványok, cikkek gyors és szép előállításának megkönnyítése. Bár elsődlegesen matematikai szövegek szedésére szánták, lehetőségei és szolgáltatásai nem merülnek ki a kifejezések, formulák elhelyezésének szakszerűségében.

Sok szempontból párhuzamosan építkezik a LATEX programcsomaggal, és bár más módon de ugyanazokat a problémákat oldja meg, ugyanazokat a szövegformázási lehetőséget kínálja a felhasználónak. Ez tulajdonképpen természetes: más makrócsomagoknál is megfigyelhetjük, hogy igyekeznek igazodni a leggyakoribb felhasználói igényekhez, ezért bizonyos mértékig hasonlítanak egymásra. A LATEX azonban elsősorban irodalmi szövegek szedését könnyíti meg, az AMSTEX inkább bonyolult műszaki, technikai szövegek szedését teszi lehetővé. Míg a LATEX-re több stílus (`.sty`) makrócsomag építkezik, addig az AMSTEX-hez eddig csak egy, az `amsppt` stílus készült el.

Ebben a szakaszban rövid áttekintést adunk az AMSTEX makrócsomagról és a VANILLA-ról, amelyet a robusztus AMSTEX makrócsomag „ízelítőjének is nevezhetnénk, hiszen nem más, mint ügyesen kiválogatott része az AMSTEX makrócsomagnak. Elsősorban azokra a lehetőségekre koncentrálunk, amelyeket a `plain` ill. `mplain` nem tartalmaz. A MATEX makrócsomaggal a LATEX nem használható, de az AMSTEX igen; ezért az eddigiektől eltérően, most a parancsok ismertetésével mutatjuk be, mit kínál az AMSTEX. A szakasz végén röviden bemutatjuk az `amsppt` makrócsomagot is.

9.5.1. Az AMSTEX matematikai jeleiről

- Az AMSTEX görög betűkből is több betűtípust biztosít: például a `\varGamma` paranccsal dőlt görög gammát lehet előállítani.
- A kaligrafikus betűket nem a `\cal`, hanem a `\Cal` paranccsal érhetjük el.
- Logikai formulákban néha az & jelet használják a logikai *és* jelölésére; ezt az AMSTEX-ben a `\and` paranccsal állíthatjuk elő.

- A logikai következtetést szolgálják az `\implies` és az `\impliedby` parancsok.
- A többszörös integrálokat egyetlen paranccsal előállíthatjuk: az `\iint` a kétszeres, az `\iiint` a háromszoros, az `\iiiint` a négyszeres integrálokra vonatkozó parancsok. Azoknál az integrált tartalmazó kifejezéseknél amelyeknél az integráljeleket (az első és utolsó kivételével) pontokkal helyettesítjük az `\idotsint` parancsot használhatjuk.

Ezeken kívül még sok jelet, szimbólumot lehet szedni az AMSTEX-ben, de azokhoz olyan fontok szükségesek, amelyeket a PC-n futó TEX programokhoz nem adnak (bár természetesen ezek is használhatók lennének PC-n is).

9.5.2. Egyéb parancsok

Az AMSTEX makrócsomag használatakor lényegében minden plain-ben definiált parancs a rendelkezésünkre áll, bár néhány kissé másként működik. Azok a parancsok amelyek az AMSTEX-ben nem használhatók — mert más szerkezetek tökéletesen helyettesítik őket — a következők:

`\=` `\>` `\cal` `\mit` `\oldstyle`

- `\smc` paranccsal EGY ÚJ BETŰTÍPUSRA VÁLTHATUNK ÁT.
- `\B` hatására felülvonás ékezetet tesz a következő betűre. A plain-ben erre az `\=` szolgált, de ez itt nincs definiálva.
- `\D` hatására pont ékezetet tesz a következő betűre. A plain-ben erre a `\.` szolgált, ez itt szintén nincs definiálva.
- `\oldnos` paranccsal „régi" stílusú számokat irathatunk ki. Ezt a plain-ben az `\oldstyle` paranccsal tehetjük meg: 1.234.567,890.
- `\buffer` paranccsal a szumma, integráljelek alatt ill. fölött szereplő korlátok elhelyezését lehet szabályozni.
- `\midspace{....}` hatására a TEX .... helyet hagy az előző bekezdés után. Ha ez már nem fér ki az adott oldalra, az egész helyet a következő oldal tetején hagyja ki.
- `\topspace{...}` hasonló a `\midspace` parancshoz, de ez az oldal tetjén hagy ki ... mennyiségű helyet.

- `\caption{...}` parancsot a `\midspace` vagy a `\topspace` után adhatjuk ki, ennek hatására a ... szöveg középrezártan, mint ábraaláírás jelenik meg.
- `\captionwidth{...}` akkor hasznos, ha az ábraaláírás hosszú. Ekkor a paranccsal beállíthatjuk, hogy a szöveget milyen hosszú sorokra tördelje a TEX.
- `\define` segítségével új makrókat, parancsokat definiálhatunk.
- `\redefine` paranccsal már korábban definiált parancsokat — akár a TEX primitíveket is — átdefiniálhatjuk.
- `\predefine` paranccsal szinonímákat hozhatunk létre, ugyan azt a definíciót több parancsnéven is elérhetjük. Érdekessége, hogy ezt a parancsot korábban kiadhatjuk, mint a parancs dafinícióját.
- `\documentstyle{...}` parancs kiválasztja a megfelelő `.sty` file-t. Egyelőre ez csak az `amsppt.sty` file lehet.
- `\syntax` parancsot a forrásfile legelején kiadva a TEX nem szedi ki a szöveget, csak szintaktikus ellenőrzést végez. Ez akkor nagyon hasznos, ha hosszabb — és valószínűleg sok hibával rendelkező — forrásfile-t akarunk szedetni.
- `\mathbreak` parancs hatására a TEX eltöri a matematikai kifejezést ott, ahol ezt kiadtuk.
- `\nomathbreak` parancs hatása éppen ellentétes az előzőével, itt akkor sem fogja eltörni a kifejezést, ha egyébként ezt megtenné.
- `\allowmathbreak` parancs kiadásával — nem kötelező érvénnyel — engedélyezzük, hogy a TEX eltörje itt a kifejezést.
- `\tag ...` parancsot kiemelt kifejezéseknél ($$ jelek között) használhatjuk. Hatására a kiemelt kifejezés a ... címkét kapja.
- `\tagsOnleft` `\tagsOnright` parancsok hatására a kiadási helyüktől kezdve a címkék a bal ill. jobb oldali margóhoz kerülnek.
- `\dsize` parancs a kifejezések betűit, jeleit a kiemelt matamatikai mód méretére állítja be akkor is ha nem kiemelt matematikai módban vagyunk.
- `\tsize` parancs hasonló az előzőhöz, de ez a szövegközben használt matematikai jel, ill. betűméretére vált át.
- `\ssize` parancs az indexben használt jel, ill. betűméretére vált át.
- `\sssize` parancs az index indexében használt jel, ill. betűméretére vált át.

- \Hat \Check \Tilde \Acute \Grave \Dot \Ddot \Breve \Bar \Vec parancsok a plain-ben definiált ékezet létrahozó parancsok megfelelői, de ezekkel ugyanarra a betűre több ékezetet is tehetünk.
- \frac{...}{...} paranccsal törteket adhatunk meg; az első kapcsos zárójelben a számlálót, a másodikban a nevezőt.
- \dfrac{...}{...} hasonló az előzőhöz, de ez a törtet kiemelt kifejezésként állítja elő.
- \binom{...}{...} paranccsal binomiális kifejezést állíthatunk elő.
- \cfrac...\\ ... \endcfrac parancsokkal lánctörtet állíthatunk elő.
- \overset ... \to{...} paranccsal egy egész kifejezéshez írhatunk kitevőt. Az első ... formula kitevőjeként jelenik meg a \to{...} parancs után megadott kifejezés.
- \underset ...\to{...} parancsokkal egy kifejezés indexébe írhatunk egy másik kifejezést.
- \Sp ...\\... \endSp parancsokkal többsoros kitevőket írhatunk. A \\ jelekkel választjuk el az egyes sorokat.
- \Sb ...\\... \endSb parancsokkal többsoros indexeket írhatunk. Itt is a \\ jelekkel választjuk el az egyes sorokat.
- \align ...\\... \endalign

  \alignat ...\\... \alignat

  \split ...\\... \endsplit

  \multline ...\\... \endmultline parancsokkal adhatunk meg többsoros elrendezéseket.
- \matrix ...\\... \endmatrix parancsok mátrixok megadására szolgálnak. Ezeknek — a bezárózárójelek típusától függően — több fajtája használható:

  \pmatrix ...\\... \endpmatrix gömbölyű zárójelet tesz

  \bmatrix ...\\... \endbmatrix szögletes zárójelet tesz

  \vmatrix ...\\... \endvmatrix egyenes zárójelet tesz

  \Vmatrix ...\\... \endVmatrix másfajta egyenes zárójelet tesz

  \smallmatrix ...\\... \endsmallmatrix parancsok hatására a mátrix kisebb méretben jelenik meg.

9.6. Az AMSPPT stílus

A stílus neve az AMS, American Mathematical Society, és a 'preprint' angol szavakból származik. Elsősorban olyan dokumentumok létrehozását segíti, amelyet valamilyen tudományos folyóiratban való közlésre szánnak.

- `\topmatter ... \endtopmatter` parancsok közé írt szöveget a TEX úgy szedi mint a címeket. Ez természetesen lehet más szöveg is, tartalmazhat a valódi címen és szerzőn ... stb kívül más információkat. Ezekre látunk példákat a következő utasításoknál. Ha az `\endtopmatter` parancsot elfelejtjük kiadni, az egész szövegből semmi sem jelenik meg.
- `\title ...\\... \endtitle` parancsok a címek megadására szolgálnak. A `\\` jelekkel befolyásolhatjuk mi kerüljön egy sorba.
- `\author ...\\ ... \endauthor` parancsok a szerző (szerzők) megadására szolgálnak. Ezt a szöveget is sorokra tördeli a TEX, ha kiadjuk a megfelelő helyen a `\\` jeleket.
- `\affil ...\\... \endaffil` parancsokat tudományos cikkek írásánál használhatjuk. Ezeknél a cikk címe után meg szokták adni a szerzők munkahelyét, a kutatóintézetet, vagy tudományos szervezeti tagságát ...stb. Erre használható az `\affil ...\\... \endaffil` parancs.
- `\overlong` parancs akkor hasznos, ha a címek, vagy egyéb középre helyezendő szövegek túl hosszúak. Ekkor a szöveg általában csak a jobb margón nyúlik túl, de ha kiadjuk előtte a `overlong` parancsot, a TEX a címet középre helyezi, és mindét oldalon egyenlő mértékben fog túlnyúlni a margón.
- `\thanks` paranccsal a cikkek elején szereplő köszönetnyilvánításokat adhatjuk meg.
- `\address` paranccsal a szerző (szerzők) címét (címeit) adhatjuk meg.
- `\keywords{...}` parancs arra szolgál, hogy megadjuk az adott tudományos cikkre jellemző — a könyvtári feldolgozást, osztályozást, témák szerinti keresést megkönnyítő — kulcsszavakat.

- `\subjclass` parancs használata megegyezik az előzőével, a különbség csak annyi, hogy ez a 'Mathematics subject classifications' alakájban adja meg a — szabványosított — kulcsszavakat.
- `\abstract{...}` parancs után a cikk kivonatát adhatjuk meg. Ez a címek után a szokásos kivonat formában (kisebb betűvel, rövidebb sorokban, középrezártan) jelenik meg.
- `\nofrills` paranccsal letilthatjuk azokat a kezdő szavakat, amelyeket a TEX automatikusan tesz a `\keywords` (Keywords : ...) `\subjclass` (Mathematics subject classifications :...) `\abstract` (Abstract ...) parancsok hatására.
- `\roster \item ... \item ... ... \endroster` parancsokkal egy felsorolást készíthetünk. A felsorolás egyes elemeit az `\item` paranccsal adhatjuk meg, ezek (1), (2), ... formájú számozást kapnak.
- `\cite` paranccsal hivatkozásokat adhatunk meg. Például `\cite{Ray-Chaudhuri, 87. old.}` hatására ez jelenik meg: [**Ray-Chaudhuri, 87. old.**].
- `\Refs` paranccsal nyitjuk meg a hivatkozott cikkek, könyvek listáját. A parancs hatására kiíródik a 'References' felirat, és az ez után következő listát kisebb betűvel szedi.
- `\enddocument` parancsot kell a forrásszöveg végén kiadnunk.

A. függelék

TEX programok a gyakorlatban

Számos különböző, a TEX-hel kapcsolatos programot kell megismernie a felhasználónak. Ráadásul a különböző felhasználók esetleg különböző TEX verziókkal rendelkeznek. Ez természetesen nem azt jelenti, hogy bárkinek aggódnia kellene amiatt, hogy az eddig leírtak nem érvényesek az ő programjára. A különbségek a programoknak a számítógépbe való betöltéséből és ott a felhasználásra való felkészítésükből adódnak, amit röviden már eddig is inicializálásnak vagy implementálásnak neveztünk. Az általános TEX programnak éppen az a nagy előnye, hogy függetlenül attól, milyen gépen milyen verziója működik, ugyanazon az input file-on ugyanúgy kell futnia. Már az IBM PC/XT/AT számítógépekre is több különböző TEX program létezik, és a hozzájuk csatlakoztatható nyomtatók meghajtására még több ún. driver. Itt nem vállalkozhatunk arra, hogy az összes forgalomban levő program leírását megadjuk, hiszen közülük nem egy esetleg több helyet kívánna, mint ennek a könyvnek a terjedelme. Így tehát most a legelterjedtebb IBM PC implementációk, a PCTEX és a MicroTEX ismertetésén kívül néhány, Magyarországon leggyakrabban használt nyomtató-meghajtót ismertetünk. Természetesen azt is megadjuk, hogy hogyan kell ezekkel együtt használni a MATEX programcsomag elemeit.

Habár minden cég forgalmaz a TEX programjaival együtt néhány nyomtató-meghajtó programot is, ezek általában nem alkalmasak minden, forgalomban levő nyomtató meghajtásához. Vannak olyan vállalatok is, amelyek csak a meghajtók forgalmazásával foglalkoznak, nem árulnak TEX programcsomagokat. Így a felhasználó rákényszerülhet, hogy egy TEX változattal egy egészen másik cég meghajtóját alkalmazza. Ez azonban nem jelenthet semmi problémát, minden kényszer nélkül is nyugodtan fordulhatunk ehhez

a megoldáshoz, ha valamiért egyik vagy másik nyomtató-meghajtó program kedvesebb a számunkra.

Nem szabad azonban elfelejteni, hogy a TEX futása közben nem ismeri az egyes betűk képeit, csak az azokat befoglaló dobozok méreteit, amit a `tfm` file-ból tud meg. A nyomtató-meghajtó program aztán a TEX által meghatározott helyekre teszi az egyes betűket. Ha ezeknek a mérete eltér a `tfm` fileban meghatározottól, akkor a betűk túl közel kerülhetnek egymáshoz, vagy egyik betű felülírhatja a másikat. Így a TEX-nek és a nyomtató-meghajtó programnak ugyanazon `tfm` file-okat kell használnia. A nyomtató-meghajtó `tfm` file-jai viszont már a vele árult betűképeknek megfelelőek, így azokat nem lehet változtatni. Így arra kell vigyáznunk, hogy amennyiben valamely meghajtóval szeretnénk kinyomtatni a szöveget, akkor az ahhoz tartozó `tfm` file-okat kell használnunk a TEX futtatása során is. Ha csak egyetlen nyomtatót használunk rendszeresen, akkor ez nem okoz gondot, minden egyéb megfontolás nélkül annak `tfm` file-jaival inicializáljuk a TEX programjainkat. Ha viszont több nyomtatót is használunk (például egy EPSON–FX és egy lézernyomtatót, vagy egy EPSON–LQ-t) akkor célszerű egyetlen cég nyomtató-meghajtóit választani a célra, amelyek ugyanazokat a `tfm` file-okat használják. Ellenkező esetben ugyanis a különböző printereken való kinyomtatásokhoz különböző TEX implementálásokon kell lefuttatnunk az input file-okat. Erre a problémára még részletesebben kitérünk majd abban a szakaszban, amelyben a meghajtókkal foglakozunk.

A.1. A különböző TEX implementációk

A.1.1. A MATEX implementálása

Az `mplain` formátum, illetve a hozzá tartozó MATEX makrócsomag bármelyik forgalomban lévő eredeti TEX programcsomaggal használható. Az `mplain` inicializálása az `INITEX` program segítségével történik, aminek meghívása viszont programcsomagonként változó. Általános szabályként azt könyvelhetjük el, hogy az `mplain`-t ugyanúgy kell inicializálni, mint a programcsomagban előforduló `plain` vagy akármelyik másik formátumot. Ez alól csak azok a TEX változatok jelentenek kivételt, amelyek úgynevezett

„preloaded", azaz előre betöltött formában tartalmazzák a plain-t. Ilyen változat a MicroTEX, amelynek egyes korai verzióival nem is használható az mplain. Ha viszont az 1.5A1 vagy magasabb verzió van már a birtokunkban, akkor a következő pontokban leírtak szerint használhatjuk az mplain-t.

Egyetlen dologról nem szabad elfelejtkeznünk az mplain implementálása előtt. Az INITEX program is felhasználja már betűtípusok méreteit az mplain.fmt file elkészítéséhez, amiket fmt file-ból vesz ki. Mivel az mplain magyar betűket is használ, a TEX programokkal forgalmazott tfm file-okat még az INITEX futtatása előtt át kell alakítani. Ezt az átalakítást végzik el a B. függelékben, a magyar fontok installálásánál leírt mapix.exe program, illetve az azt felhasználó installáló file-ok. Az átalakított tfm file-ok aztán minden további nélkül felhasználhatók az angol nyelvű implementálásokhoz is, csak a második fejezetben leírtakban térnek el az eredetiektől. *Ne felejtsük el tehát az* mplain *inicializálása előtt a B. függelékben leírtak szerint a* tfm *file-okat átalakítani.* A mapix program használatát, és a MTEX-re vonatkozó egyéb tudnivalókat is a B. fejezet tartalmazza. Miután a tfm file-ok átalakítását elvégeztük, a szorosan vett TEX programok, és velük együtt az mplain inicializását minden további nélkül elkezdhetjük. Természetesen a nyomtatáshoz vagy a képernyőn való megjelenítéshez azokat a file-okat is át kell majd alakítani a mapix segítségével, amelyek az adott kinyomtatáshoz (tehát nyomtatóhoz és annak meghajtójához) tartozó betűtípusok képeit tartalmazzák.

Vigyázzunk nagyon arra, hogy az egyes formátumfile-ok csak azokkal a TEX változatokkal futtathatók, amelyekkel elkészítettük őket. Ellenkező esetben a program hibaüzenetet ad, és (ebben eltér a többi hibaüzenettől) mindenképp leáll. Tehát ne próbáljunk mondjuk egy olyan mplain formátumot a PCTEX-hel együtt használni, amit egy másik gépen valamely MicroTEX változat segítségével inicializáltunk.

A.1.2. A PCTEX programcsomag

A PCTEX csomag a Personal TeX társaság terméke, Magyarországon talán a legelterjedtebb válozata (ha egyáltalán van ilyen) a TEX programnak. Több különböző verziója is forgalomban van, amelyek természetesen a verziószámok növekedésével egyre fejlettebb változatot jelentenek, de a céljainkra

bármelyik megfelel. Azaz amennyiben valakinek bármely verzió a rendelkezésére áll, és eddig működött, akkor azt az mplain-nel is működtetheti majd. A különböző verziók a program sebességében, annak tárigényében, esetleges itt nem részletezett extra szolgáltatásokban különböznek. Nem szabad viszont elfelejtenünk, hogy noha egyes változatok már 512 kbyte memóriával is képesek üzemelni, az inicializáláshoz (amely elkerülhetetlen lépés — esetleg egy másik gépen végezhetjük el) az INITEX program minden esetben 640 kbyte memóriát igényel.

A program felvitelét a merevlemezre az installáló bat kiegészítésű file-ok végzik el. A programlemezek egyikén (általában az elsőn) könnyen megtalálhatjuk azt a nevéről — általában ins, inst vagy install a neve, vagy legalább ezzel kezdődik, illetve tartalmazza ezeket a részszavakat. A leggyakoribb a PCTEX verzió esetén az texinst.bat nevű installáló file. A legtöbb esetben ezt a programot a merevlemez főkönyvtárába kell másolni, és utána elindítani, majd követni a képernyőn megjelenő utasításokat. Amikor a program befejezte működését, a TEX-hel kapcsolatos programok egy PCTEX nevű alkönyvtárban találhatók. Magában az alkönyvtárban csak a tex.exe file, míg a többiek a PCTEX alá tartozó alkönyvtárakban. A TEXTFMS alkönyvtárban vannak a tfm file-ok, a TEXINPUT alkönyvtárban a makrócsomagok, stílusfile-ok, és a program futásához szükséges egyéb kisegítő file-ok[1], míg a TEXFMTS alkönyvtárban a formátumfile-ok. Ez utóbbi majdnem üres, csak a számunkra ismeretlen (de a programnak nagyon fontos) TEX.POO file-t tartalmazza. *Ha a magyar változatot használjuk, az* mplain.tex, matex.tex *és* jelval.tex *file-okat a* TEXINPUT *alkönyvtárba kell bemásolnunk.*

Az installálás után tehát így kell kinéznie a TEX által használt alkönyvtárak rendszerének. Az ábrában nagybetűkkel az alkönyvtárakat, kisbetűkkel a bennük levő programokat jelöljük.

```
ROOT    inst.bat
        PCTEX       tex.exe
                    meghajto.exe
                    TEXTFMS         xxx.tfm
                    TEXFMTS         tex.poo
                    TEXINPUT        xxx.tex
                                    xxx.sty
```

[1] Ebben az alkönyvtárban található file-ok többsége a LATEX futásához szükséges.

```
mplain.tex
elval.tex
matex.tex
```

Amennyiben a PCTEX egy korai változatát használjuk, előfordulhat, hogy miután a programoknak a merevlemezre másolása megtörtént, a PCTEX alkönyvtárban nem egy, hanem két exe file-t találunk, amelyek közül az egyik neve tex512 és a másiké tex640. Ezek e program két olyan változata, amelyek 512 illetve 640 kbyte rendelkezésre álló memória mellet használandóak. Válasszuk ki azt, amelyik a számítógépünknek megfelel, és nevezzük át tex.exe-vé, majd a továbbiakban folytassuk úgy az inicializálást, mintha eleve csak ez az egy tex.exe file-unk létezett volna.

A fentebb leírt állapot nem alkalmas még a program futtatására, hiszen nem áll rendelkezésre egyetlen alapvető formátum sem a TEXTFMS alkönyvtárban. Azokat, amelyeket a későbbiekben használni fogunk, el kell tehát készítenünk. Amennyiben ez a plain, az mplain, vagy a LATEX-et használó lplain, akkor a PCTEX alkönyvtárban kiadjuk a

```
tex plain/i
```

parancsot, ahol a plain helyén természetesen az lplain ill. az mplain is szerepelhet. A program a gép sebességétől függően hosszabb-rövidebb futási idő után leáll[2], és az utolsó sor elején egy csillaggal jelzi, hogy a TEX újabb utasításokra vár. Ekkor ki kell adnunk a csak ebben az esetben használt \dump utasítást, amelynek hatására a program kiírja a merevlemez PCTEX alkönyvtárába a memóriában már létrehozott formátum (tehát fmt kiegészítésű) file-t. Innen aztán azt a TEXFMTS alkönyvtárba kell másolni, és a felesleges melléktermékeket (beleértve az adott helyen már szükségtelen fmt file-t is) ki kell törölni a PCTEX alkönyvtárból.

A program futtatásához most már csak néhány DOS paramétert kell beállítani. Először is, mivel a program mindig az éppen érvényes alkönyvtárban vagy a PCTEX\TEXINPUT alkönyvtárban keresi a neki inputként megadott file-okat, ahol viszont általában nem található a tex.exe file, meg kell adnunk a C:\PCTEX alkönyvtárat az autoexec.bat file PATH parancsában. A másik *igen fontos* paraméter, amit be kell állítanunk, a DOS által egyszerre

[2] De a futása közben mindig kiírja éppen mit csinál — a MATEX esetében magyarul.

kezelhető file-ok száma, mivel különösen a LATEX formátum igen nagy igényű ebben a tekintetben. Ugyancsak nem árt, ha a DOS buffer méretét is nagyobbra állítjuk. A legbiztosabb megoldás, ha *minden olyan számítógépen,* amelyen a TEX akármelyik változatát futtatni kívánjuk, a következő két sort szerepeltetjük a DOS `config.sys` file-jában:

```
files=20
buffers=40
```

Ennek hiányában azt tapasztalhatjuk, hogy a program nem hajlandó megtalálni olyan file-okat, amelyek pedig léteznek és az általunk megadott helyen vannak.

Kicsit bonyolultabb az eljárás, ha egy, az AMSTEX makrócsomagot is tartalmazó formátumot szeretnénk előállítani. Ebben az esetben hozzunk létre egy mondjuk `amplain.tex` nevű file-t (aminek a neve tehát különbözik a `TEXINPUT` alkönyvtárban található `amstex.tex` makrócsomag nevétől) oly módon, hogy a `plain.tex` vagy `mplain.tex` makrócsomagok legvégén, közvetlenül a

```
\catcode`@=12
```

sor előtt beírjuk az

```
\input amstex
```

sort. Utána ezt az `amsplain` makrócsomagot ugyanúgy inicializáljuk, ahogy fentebb a többi `plain` típusú file-t, csak a `TEXFMTS` alkönyvtárba a

```
copy amsplain.fmt texinput\amstex.fmt
```

paranccsal másoljuk át a „dumpolás" után. Mostantól kezdve rendelkezésünkre áll egy `amstex.fmt` formátumfile is.

Mint azt korábban már leírtuk, a TEX programot egy adott formátummal az input file-on a

```
tex &formátum input
```

DOS parancs segítségével futtatjuk, ahol a `formátum` egy a `TEXFMTS` alkönyvtárban szereplő formátumfile, míg az `input.tex` egy, az éppen érvényes alkönyvtárban levő TEX input file. A futás összes eredménye a `log` file-ban az

alkönyvtárban jelentkezik, ahol a futtatást végeztük. Azt hasznos még tudnunk, hogy ha valamilyen okból le szeretnénk állítani futás közben a programot, ennél a változatnál a `Ctrl` és `C` billentyűk háromszori lenyomásával tehetjük meg. Ilyenkor a már ismert `Interruption` hibaüzenet jelenik meg előbb-utóbb (a program csak bizonyos helyzeteiben — leginkább amikor a lemezre történő kiírást végzi — hajlandó elfogadni kívülről érkező utasításokat), amelyre a tanult módok közül tetszőlegesen reagálhatunk, többek között egy `x` válasszal be is fejezhetjük e program működését.

Egyetlen dolgot írunk még le a PCTEX-ről, nevezetesen, hogy hogyan lehet merevlemez nélküli, de két hajlékonylemez-meghajtót tarlmazó gépen működtetni. Habár a gyári leírás nem ajánlja ezt a futtatási módszert, a felhasználók mégis rákényszerülhetnek. Ez persze semmi esetre sem ad teljes megoldást, jóval nehézkesebb, mint a merevlemezes felhasználás, és inkább csak az input file tesztelésére használható, mivel a végeredményt elegendő tárolókapacitás hiányában sem kinyomtatni nem tudjuk, sem a képernyőn megjelentetni. Az alábbi módszer tehát csak a kényszer szülötte, és így nem is biztos, hogy a legszerencsésebb.

A két lemezen való futtatást tehát a következőképpen próbálhatjuk ki: az A meghajtóba helyezzünk el egy olyan lemezt, amelyen egy `PCTEX` alkönyvtárban a `tex.exe` program és egy `TEXTFMS` alkönyvtár szerepel. A `TEXTFMS` alkönyvtárnak tartalmaznia kell azokat a `tfm` file-okat, amelyek az input file-ban szereplő betűtípusokhoz tartoznak. Az előre definiált fontokat az `fmt` file-ok ismerik, tehát csak azon betűtípusok `tfm` file-jait kell megadni, amelyeket egy `\font` utasítás segítségével definiáltunk az inputban. Egyébként ha nem áll rendelkezésünkre túl sok betűtípus, az összes `tfm` file elfér még a lemezen a `tex.exe` file mellett. Ellenkező esetben is egy elég széles válogatásra van helyünk. A B meghajtóba helyezzünk egy olyan lemezt, amely szintén egy `PCTEX` alkönyvtárat tartalmaz, abban az A meghajtó lemezével megegyező `TEXTFMS` alkönyvtárat, egy `TEXFMTS` alkönyvtárat a `tex.poo` file-lal és a felhasználni kívánt formátumfile-okkal, és egy esetleges `TEXINPUT` alkönyvtárat, amelyben azok a stílusfile-ok szerepelnek, amelyeket az input file felhasznál. Szerepelnie kell még a `PCTEX` alkönyvtárban a TEX input file-nak is. Be kell még vennünk az `autoexec.bat` file `path` parancsába az `A:\PCTEX` alkönyvtárat is, és végül a B meghajtó `PCTEX` alkönyvtárából indítjuk el a programot a szokásos

```
tex &formátum input
```

paranccsal.

Láthatjuk, hogy elég sok file-t kell elhelyeznünk ehhez a módszerhez a B meghajtóban levő lemezen — az A meghajtóban levő mindenképpen betelik az ott alkalmazandó file-okkal. Ráadásul ide kerülnek még a futtatás során keletkező dvi és log, valamint egyéb esetleges file-ok is. Tehát csak akkor tudunk a fent leírt módon eljárni, ha a felhasznált formátum nem túl nagy és nem használ túl sok kisegítő file-t vagy az input file kicsi méretű. A LATEX használata esetén például csak igen korlátozott méretű input file-okat futtathatunk.

A.1.3. A MicroTEX programcsomag

Ebben a pontban sok esetben utalunk azokra a dolgokra, amelyet az előző pontban a PCTEX programcsomagról mondtunk el. Így aztán annak az olvasónak is ajánlatos az előző pontot megismernie, akinek a MicroTEX programcsomagja van meg, és igazából csak erre kíváncsi.

A MicroTEX az Addison-Wesley Publishing Company terméke. Korai változatai csak a plain formátummal futottak, ami előre be volt töltve, és így nem volt lehetséges a felülírása. Mivel ez nem alkalmas a magyar formátummal együtt való futtatásra, itt nem is térünk ki rá. A mi céljainknak az 1.5A1 vagy magasabb verzió felel meg, amely az eltérések ellenére sok hasonlóságot is mutat a PCTEX-hel. A merevlemezre történő installáláskor is ahhoz hasonlóan kell eljárni, legalábbis az első lépésekben. Az első eltérés, hogy ajánlatos az install.bat file helyett az mtall.bat file-t használni, amely minden olyan programot felmásol a merevlemezre, amire később szükségünk lehet. Ezután ugyanolyan alkönyvtárak jönnek létre, mint a PCTEX használata esetében, csak az elnevezések lesznek mások. Aminek ott PCTEX a neve, azt itt TEX-nek hívják, az ottani TEXTFMS megfelelője itt FONTS, a TEXINPUT-é INPUTS és végül a TEXFMTS-é FORMATS. Így a PCTEX leírásánál adott ábrának a MicroTEX esetén a következő lesz a megfelelője:

```
ROOT    mtall.bat
        TEX          tex.exe
                     initex.exe
                     preload.exe
                     meghajto.exe
```

ROOT	TEX	FONTS	xxx.tfm
ROOT	TEX	FORMATS	
ROOT	TEX	INPUTS	xxx.tex
			xxx.sty
			mplain.tex
			elval.tex
			matex.tex

Az itt szereplő alkönyvtárak lényegében ugyanazokat a file-okat tartalmazzák, mint a PCTEX-es megfelelőik, azzal az eltéréssel, hogy ebben az esetben a LATEX-et és hozzá tartozó egyéb file-okat később kell installálni. Ezt a LATEX-et tartalmazó lemezekről végezhetjük el, az `install.bat` file alkalmazásával, ha a TEX már installálva van. Lényeges eltérés az is, hogy a programcsomag főkönyvtára, a TEX könyvtár a `tex.exe` programon kívül még egyéb file-okat is tartalmaz, amelyek közül nekünk a későbbiekben az `initex.exe` és a `preload.exe` programokra lesz szükségünk. Végül a `tex.poo` file-t itt beleépítették a `tex.exe` file-ba, nincsen rá külön szükség a FORMATS alkönyvtárban. Ez az alkönyvtár eredetileg tehát üres, csak az általunk készített formátumfile-ok kerülnek majd oda.

A MicroTEX esetében az installálás második lépése, a formátumok inicializálása az INITEX segítségével lényegesen eltér a PCTEX-nél leírtaktól. Először is, amennyiben csak a `plain` formátumot szeretnénk használni, azt nem kell inicializálni, mivel az itteni `tex.exe` file azt tartalamzza. Tehát ebben az esetben a `tex input` parancs egyenértékű a PCTEX esetén kiadott

```
tex &plain input
```

paranccsal. Ha viszont itt is inicializálni kívánjuk a többi formátumfile-t, akkor azt az

```
initex lplain \dump
```

mintájára tehetjük meg. Ugyanezt az eljárást kell követnünk az `mplain` esetén is. Mint látjuk, az az alapvető különbség, hogy itt külön neve van a INITEX programnak és a `\dump` parancsot már az elején kiadhatjuk. Ezek után ugyanúgy járunk el, mint a PCTEX esetében, azaz átmásoljuk az `fmt` file-okat a FORMATS alkönyvtárba.

Ha olyan makrócsomagokat akarunk inicializálni, amelyek feltételezik a plain formátum használatát (mint amilyen az amstex.tex), akkor azt a PCTEX már leírt nehézkesebb eljárásával ellentétben egyszerűen a

```
initex &plain amstex \dump
```

vagy a magyar változat használata esetén a

```
initex &mplain amstex \dump
```

mintájára tehetjük meg. Az első változat esetén természetesen szükségünk van a plain.fmt formátumfile-ra. Eredetileg ez nem létezik, csak a tex.exe file tartalmazza előre feldolgozva. Ez viszont az INITEX számára nem elérhető, így ilyenkor a plain.fmt file-t előzőleg a már ismert minta szerint el kell készítenünk.

A MicroTEX tex.exe programját tehát már az inicializálás nélkül használhatjuk a plain formátummal. Ha olyan makrócsomagokból készítettünk formátumfile-okat, amelyek „nem mondanak ellent" a plain formátumnak, azaz nem tartalmaznak azzal ellentmondó információt, akkor azokat a plain-nel együtt a szokásos

```
tex &formátum input
```

módon használhatjuk. De így az mplain formátumot nem használhatjuk, mivel ebben az esetben nem tudja a program, hogy például melyik (egymástól gyökeresen különböző) elválasztási algoritmust használja, a plain angol, vagy az mplain magyar szabályait. Ezen probléma megoldásához segít a már előbb említett preload.exe program, amellyel tulajdonképpen tetszőleges olyan tex.exe file-t készíthetünk, amely előre betöltött formában tartalmazza az általunk megadott formátumot. (Annak mintájára, ahogy a forgalmazott tex.exe program tartalmazza a plain formátumot.) Ilyenkor nem szükséges a tex &... alakú meghívása a programnak, csupán egyetlen program használatára az előre betöltött formátum is érvénybe lép.

A preload használata a következő: először hívjuk meg a programot a preload megadásával a DOS parancssorában. Amikor bejelentkezik, az első kérdésére adjuk meg annak a file-nak a nevét, amely az alapvető TEX programot tartalmazza. Ez természetesen szinte minden esetben a tex.exe file, esetleg, ha a preload programot egy másik alkönyvtárból hívjuk meg, akkor a \tex\tex.exe módon megadva. A második kérdésre adjuk meg

annak a file-nak a nevét, amelyben a végeredményt tárolni szeretnénk, például `mtex.exe`. Végül a harmadik kérdés annak a formátumfile-nak a nevére vonatkozik amelyet előre betöltött formában szeretnénk használni, mondjuk `mplain.fmt`. Ezekre a válaszokra a program létrehozza a `mtex.exe` file-t, és a továbbiakban a `mtex input` megadása teljesen egyenértékű lesz azzal, mintha `tex &mplain input`-ot írtunk volna a PCTEX-ben.

Ezután az `mplain.tex` és hozzá hasonló formátumokat az ún. `virtex.exe` program létrehozásával is használhatjuk. `virtex.exe`-nek nevezzük a `preload` által létrehozott programot, ha az első kérdésre a `tex.exe`, a másodikra a `virtex.exe` választ, míg a harmadikra üres választ, azaz csak egy return-t adunk. Ebben az esetben ugyanis nem lesz formátumfile, amit előre betöltsön a program, és így egy olyan TEX program jön létre, amelyik teljesen „üres", azaz megfelel a PCTEX `tex.exe` file-jának. Ezt a `virtex.exe` file-t aztán ugyanúgy használhatjuk, mint a PCTEX `tex.exe` file-ját. Többek között használhatjuk vele az `mplain.fmt` file-t a

```
virtex &mplain input
```

mitájára.

Azok a tudnivalók, amelyeket a DOS paraméterek leírásáról adtunk a PCTEX esetében, teljes mértékben érvényben maradnak a MicroTEX-re is. Az egyetlen kivétel, hogy a MicroTEX-nek van egy speciális hibaüzenete annak bejelentésére, hogy az általa kinyitni kívánt file-ok száma meghaladja a DOS által medengedett mértéket. Ezt a

```
Too many files for DOS to handle
```

üzenettel hozza a felhasználó tudtára, szemben a PCTEX más célra is használt (általános) hibaüzenetével:

```
I can't find ... file
```

Egy másik kisebb eltérés a programok között, hogy a MicroTEX a `Ctrl` és `Break` (egyes billentyűzeten ez utóbbi csak `Scroll Lck`-ként van jelölve) billentyűk együttes lenyomására áll le. Ilyenkor ugyanazt teszi (és ugyanúgy kell kezelni is), mint a PCTEX a `Ctrl` és `C` billentyűk együttes lenyomására.

A gyártók a PCTEX-hel ellentétben ajánlják a MicroTEX-et mindössze két hajlékony-lemez meghajtóval rendelkező gépre is, habár az alkalmazandó

megoldás nem tér el lényegesen a PCTEX esetétől. Ugyanolyan szereposztásban használhatjuk a lemezeket, mint az előző esetben, mindössze az azokon szereplő alkönyvtárakat kell megváltoztatni. Azon kívül, hogy itt minden alkönyvtárnak a MicroTEX-ben használatos nevet kell adnunk, ebben az esetben nem szükséges, hogy a lemezeken szrepeltessük a TEX alkönyvtárakat. Helyette minden oda kerülő filet tegyünk a lemez főkönyvtárába. Nincsen szükség továbbá az A lemezen a tfm file-okra és az őket ebben az esetben tartalmazó FONTS alkönyvtárra. Nagyon fontos eltérés még, hogy amennyiben az előző bekezdésekben leírt, előre betöltött formátumokkal ellátott exe file-t használunk, akkor nincsen szükség a B lemezen a formátumokra és az azokat tartalmazó FORMATS alkönyvtárra sem. (A PCTEX-nél használt exe file-ra sincs szükség.) Ezeket a különbségeket figyelmebe véve a PCTEX-hez hasonlóan futtathatjuk két hajlékony lemezzel a MicroTEX-et.

Előfordulhat azonban, hogy az A meghajtóba kerülő lemezre nem fér rá az általunk használni kívánt exe file, amennyiben az előre betöltött formátum túl nagy. Ilyenkor mégis arra kényszerülünk, hogy az A lemezen a virtex.exe file-t használjuk, és a B lemezen adjuk meg a formátumot, mint ahogy a PCTEX esetében tettük. De ez csak abban az esetben fordul elő, ha a formátum túl nagy, akkor viszont esetleg a B lemezen sincsen hely a számára. Ezt a problémát a PCTEX nem tudta megoldani, de a MicroTEX tartalmaz egy speciális futtatási módot a kikerülésére. Ha a szokásos egy & jel helyett kettőt alkalmazunk a

```
virtex &&formatum input
```

mintájára, akkor a program a formatum.fmt file beolvasása után leáll, üzenetet ír ki a képernyőre, megengedi, hogy kicseréljük a formátumot tartalmazó lemezt a B meghajtóban egy olyan lemezre, amelyik az input file-t tartalmazza, és csak a return gomb lenyomása után folytatja futását. *Vigyázzunk azonban, a* virtex.exe *programot tartalmazó lemezt ebben az esetben sem szabad kivenni a meghajtóból.* Általában is nagyon vigyázzunk arra, hogy a program futása közben nehogy valamelyik lemezt kivegyük a meghajtóból.

A.2. A nyomtató programok

Miután valamely TEXváltozattal előállítottuk a dvi file-t, meg is kell jelenítenünk. Ez történhet akár a képernyőre történő kivetítéssel, akár nyomtatót, és a hozzá tartozó speciális programot használva végleges formában. A képernyőn általában csak ellenőrizzük a szöveget, míg a különböző minőségű képet előállító nyomtatókkal megtekintésre is, és nyomdakészen is előállíthatjuk. Minden lehetőséget beleszámítva négy különböző szinten jeleníthetünk meg egy szöveget: a képernyőn, mátrixnyomtatón, lézernyomtatón és fényszedőn. A kép minösége az előző lista sorrendjének megfelelően fokozatosan javul, de természetesen ezzel együtt nő a szükséges technikai és anyagi háttér nagysága is. Ma Magyarországon viszonylag sok IBM/PC munkahely rendelkezik a képernyőn való megjelenítéshez szükséges grafikus kártyával és a legegyszerűbb kinyomtatáshoz használt mátrixnyomtatóval, míg sokkal kevesebb megfelelő lézernyomtatóval, és valószínűleg egyetlen egy sem fényszedő géppel.

A lézernyomtatón és a fényszedőn való előállítást nem tárgyaljuk külön ebben a könyvben. Ennek részben az az oka, hogy ezek még viszonylag kevésbé elterjedtek hazánkban, részben pedig az a tény, hogy részletes ismertetésük túl sok helyet igényelne. Vázlatosan viszont nem elég és nem is szabad leírni ezeket. Egy lézernyomtató túlságosan drága és érzékeny műszer ahhoz, hogy egy felületes leírás alapján esetleg valaki elrontsa. A fényszedő gépekre írt programok pedig még teljesen ismeretlenek az országban. Az esetleges beruházónak azt érdemes megjegyezni, hogy pillanatnyilag a világon a leginkább forgalmazott, dvi file-t lézernyomtatón kiíró programok vagy Hewlett-Packard LaserJet PLUS-t vagy pedig ún. PostScript kompatibilis lézernyomtatót (amilyen például az Apple LaserWriter) használnak[3]. Létezik meghajtó[4] a korábbi Hewlett-Packard LaserJet változatra is, de az külön hardver-beruházást igényel.

[3] Ezek a típusok vannak egyébként is a leginkább elterjedve.

[4] A továbbiakban meghajtónak nevezzük ezeket a programokat

Tehát a továbbiakban mátrixnyomtatókkal és a képernyőn való megjelenítéssel foglalkozunk. Ebbe a témakörbe tartozó programok közül ismertetjük a (legelterjedtebb) PCTEX-hel forgalmazott EPSON–FX és RX nyomtatókhoz, valamint az EPSON–LQ nyomtatókhoz készült meghajtókat, a MicroTEX-hel forgalmazott EPSON–FX és RX nyomtatókhoz készült meghajtót és a képi megjelenítéshez használt (a Textset cég által készített) Preview programot. Ezek természetesen (származási helyüktől függetlenül) minden TEX programmal együtt használhatók, feltéve, hogy a hozzájuk tartozó `tfm` file-okat használjuk a TEX program futásakor is.[5] és rendelkezésünkre állnak a meghajtó által igényelt (vele együtt forgalmazott) pixelfile-ok.

Az egyes meghajtók korlátozott mértékben egy másik pixelfile-jaival is futhatnak. Ennek az a feltétele, hogy a pixelfile-ok azonos formátumúak vagy egymásba konvertálhatók legyenek. Sajnos az utóbbi időben elég sok különböző pixelfile formátum terjedt el, annak ellenére, hogy eleinte csak egy, általánosnak tekintett ún. PXL formátum létezett. Sőt, napjainkra már ennek a formátumnak is bevezették egy újabb változatát, tehát még a PXL file-ok sem biztos, hogy azonos felépítésűek. Néhány cég — de nem mindegyik — kiegészíti a meghajtóit tartalmazó programcsomagot olyan programmokkal, amelyek PXL formátumba és vissza konvertálják az általuk forgalmazott pixel formátumokat. A bemutatandó példákat azért választottuk, mert egyúttal néhány különböző formátumot is megismerhetünk általuk.

Mielőtt rátérnénk a mátrixnyomtatókat meghajtó programok konkrét ismertetésére, ezek néhány közös sajátosságát írjuk le. A programok az oldal nyomtatása közben maguk irányítják a nyomtatót (tehát saját betűkészletet használnak, maguk osztják be a sortávolságot, stb.), de az oldalemelést a nyomtatóra bízzák. Ezért a programtól függetlenül kell beállítanunk, hogy a nyomtató a felhasznált papír méretének megfelelően emeljen lapot, és hogy tudja, hol van a lap teteje. Ez utóbbit úgy a legegyszerűbb megtenni, hogy a nyomtatót akkor kapcsoljuk be, amikor az írófej egy fizikai lap tetején helyezkedik el.

---

[5] Amennyiben egy meghajtót olyan `dvi` file-on használunk, ami nem a hozzá tartozó `tfm` file-okkal készült, az output nem lesz tökéletes, és a meghajtó figyelmeztető üzeneteket ír a képernyőre. Ezeket az üzeneteket azonban csak a gyakorlott felhasználók értik meg.

A.2.1. A PCTEX-hez árult meghajtók

Ebben a pontban a PCTEX-hel, együttesen PCDOT márkanév alatt forgalmazott, EPSON–FX, RX és LQ mátrixnyomtatókra írt meghajtókat ismertetjük. Ezek elég nagy mértékben igénylik, hogy valóban EPSON típusú nyomtatóval használjuk őket, de akinek olyan nyomtatója van, amely elméletileg valamely EPSON típust — esetleg nem tökéletesen — emulál, mindenképp kipróbálhatja őket. Az EPSON–FX és RX típusokra írt meghajtó neve DVIEPS, míg az LQ típusra írté DVIELQ. A meghajtókat és a pixelfile-okat tartalmazó csomag általában tartalmaz valamely, a nevéből felismerhető installáló file-t, de ha mégsem, akkor az itt leírtak alapján a felhasználó maga is elvégezheti ezt a műveletet.

Maguk a meghajtó programok általában a `PCTEX` alkönyvtárba kerülnek, és a `PCTEX\TEXTFMS` alkönyvtárban keresik a `tfm` file-okat. Ha tehát PCTEX-hel használjuk őket, semmi gondunk ezzel, egyébként fel kell építenünk a megfelelő alkönyvtár-rendszert, amelyet a továbbiak szerint még jobban ki kell majd építeni. A fontok képét tartalmazó file-ok mindig a `PCTEX\PIXEL` alkönyvtárban vannak, azon belül is minden egyes fokozat más és más alkönyvtárban. Ezen alkönyvtárok elnevezésére az a szokás alakult ki, hogy az alapvető fokozatot tartalmazó alkönyvtár mindig a pixel-kép pont/inch mértékének az ötszörösét jelölő számot tartalmazza valamilyen formában, míg a következő fokozatokat tartalmazó alkönyvtárak ennek a nagyításnak megfelelő szorzatát. Így a DVIEPS meghajtó esetében, mivel az 240 pont/inch legsűrűbb felbontással dolgozik, ezek a számok $5\times240 = 1200$, $1200\times1{,}09543 = 1315$, $1200\times1{,}2 = 1440$, 1728, 2074, 2488, és, amennyiben létezik a `magstep5` fokozat, 2986. Ugyanezek a DVIELQ esetében, amely ellentmondva annak, hogy nagyobb felbontó képességű printert használ, 180 pont/inch sűrűséggel dolgozik, 900, 986, 1080, 1296, 1555, 1866 és 2239. Az egyes betűfokozatok itt olyan alkönyvtárakban találhatók, amelyek neve a PXL betűkből és a fokozathoz tartozó számból áll. Az 1200-as sűrűségű fontok tehát a `PXL1200` nevű alkönyvtárban vannak.

Amikor a lemezen forgalomba hozzák, némelyik file még nem a szabványos PXL formátumban, hanem PP formátumban van. Ezt a programcso-

maggal együtt forgalmazott pxlpak.exe program konvertálja PXL formátumba, ha a következő minta alapján meghívjuk az egyes file-okra:

```
pxlpak amr10.pp amr10.pxl /u
```

Ezt a műveletet az installáló program elvégzi, a felhasználónak csak akkor van rá szüksége, ha valamiért maga installálja a pixelfile-okat.

Az itt leírt DVIEPS, illetve DVIELQ programok újabb változatai már képesek egy (korábban már említett) másik, a PXL formátumnál sokkal kisebb helyet elfoglaló fomátumú pixel file-okat felhasználva is futni. Az ilyen új formátumú pixelfile kiegészítője PK. Azon meghajtókkal, amelyek képesek a PK-file-okkal is futni, forgalmaznak két konvertáló programot, a pxtopk.exe és a pktopx.exe programokat, amelyek a

```
pxtopk amr10.pxl amr10.pk
```

és a

```
pktopx amr10.pk amr10.pxl
```

minták alapján használva a PXL formátumú pixelfile-okat PK formátumúvá alakítják, illetve fordítva[6].

Ezek után már csak annyit kell megtanulnunk, hogyan működtethetők ezek a programok. Meghívásuk a nevük és az alkalmazandó DVI file együttes beírásával történik, a

```
dvieps input
```

mintájára, ahol az input.dvi egy a TEX-hel már előállított file. Ezek után a program fajtától függően néhány kérdésre vár választ. Első kérdése arra vonatkozik, hogy melyik nyomtatót használja. Mivel a felhasználó többségénél csak egyetlen nyomtató van egy számítógépre kapcsolva, a szokásos válasz egy egyszerű (return). Akinek pedig egyszerre több nyomtató áll rendelkezésére, annak úgyis tudnia kell, hogyan különbözteti meg őket egymástól. A második kérdés arra vonatkozik, hogy (amennyiben a program választási lehetőséget ad) milyen minőségben kérjük a nyomtatást. Ez a kérdés vagy teljesen elmarad (a DVIELQ csak egyetlen, nem túl igényes minőségben

[6] A B függelékben leírtak szerint az ilyen meghajtók használatakor fontos szerepe lesz a konvertáló programoknak a magyar ékezetes betűk kialakításában.

képes nyomtatni), vagy pedig ún. „draft", azaz gyengébb minőséget kérhetünk a d billentyű lenyomásával és „final", azaz végleges, jobb minőséget az f billentyűvel. A harmadik kérdés azt tudakolja, melyik oldaltól kívánjuk a DVI file-t kinyomtatni, míg az utolsó azt, hogy *hány oldalt* kérünk (tehát ide nem az utolsó kinyomtatandó oldal számát adjuk meg). A számára szükséges paraméterek megadása után a program elkezdi a nyomtatást, és a minőségtől függően néhány perc alatt nyomtat ki egy-egy oldalt.

Egyszerűsége folytán nem sok hibaüzenet érkezik a programtól. A legnagyobb baj, ha nem találja a megadott DVI file-t vagy pedig egy ahhoz szükséges pixelfile-t. Az előző esetben leáll, míg a másodikban egy olyan file nevét kéri, amellyel helyettesítheti a hiányzót. Ilyenkor vagy a kérttel azonos, csak más fokozatú fontot tartalmazó file-t adjunk meg, vagy pedig azonos fokozatú, de eltérő típusú betűt, ugyanabban a formában, ahogy a program a hiányzó fontot kiírta (azaz csak a PXL alkönyvtárat és a font nevét kell megadni, a `pxl2074\hamr10` mintájára. Természetesen egyik megoldás sem az igazi, az így készült nyomtatvány nem elégíti ki a nyomdai igényeket, mindössze a szöveg ellenőrzésére szolgálhat, és annak eldöntésére, hogy milyen fontot használjunk a hiányzó helyet. A végleges változatban csak olyan DVI file-t szabad kinyomtatni, amelyhez megvannak a szükséges fontok.

A program futása közben mindig kiírja (habár elég vázlatosan és néha furcsa módon) annak az oldalnak a képét, amin éppen dolgozik, még mielőtt elkezdené a nyomtatást. Eközben esetleg figyelmeztető üzenetek is megjelenhetnek a képernyőn. A leggyakoribbak úgy kezdődnek, hogy `|h|` és utána egy igen nagy számot tartalmaznak. Ezek a túl hosszú sorokra figyelmeztetnek. Egy másik, remélhetőleg ritkább üzenet, amely valamilyen módon a `checksum`-ról szól. Ennek a legvalószínűbb oka, hogy a DVI file nem ugyanazokkal a TFM file-okkal készült, mint amiket a meghajtó használ.

A.2.2. A MicroTEX-hel árult meghajtók

Most is feltételezzük, hogy az Olvasó már rendelkezik az előző pontban leírt ismeretekkel, még akkor is, ha neki MicroTEX programcsomagja van. Ezzel a programmal DVI-EPS név alatt forgalmaznak EPSON–FX, RX és MX nyomtatókhoz meghajtót. A program installálása vagy már a TEX programmal együtt megtörténik, vagy pedig egy olyan program végzi, amely

nevében fellelhető az INSTALL és EPSON szavak egy-egy részlete. Ez a DVI-EPS programot a TEX alkönyvtárba helyezi, és a pixelfile-ok számára létrehozza annak TEXFONTS alkönyvtárát. Az egyes betűfokozatokat tartalmazó még alacsonyabb rendű könyvtárak neve ebben az esetben csak az előző pontban már ismertetett számból áll (tehát például 1200, vagy 1728). Az egyetlen különbség, hogy itt fele akkora felbontású betűképek is szerepelnek, a 600, 657, 720, 864, 1037 és 1244 nevű alkönyvtárakban. Az ezekben szereplő fontok felbontása 120 pont/inch, és a program a gyengébb minőségű nyomtatás előállításakor használja ezeket.

Ez a meghajtó nem az általános PXL formátumú pixelfile-okat használja, hanem egy EPF nevűt, amelyet természetesen az EPF kiegészítőjű file-ok tartalmaznak. A forgalmazott lemezeken a pixelfile-ok egy része még nem ilyen alakú, hanem CMP formátumban szerepel. Ezeket a programcsomag CMP-EPF programja alakítja át a megfelelő alakúra, ha a

```
cmp-epf amr10.cmp
```

mintájára meghívjuk. Természetesen ezt a lépést elvégzi helyettünk az installáló program. A fenti programhoz hasonló a PXL-EPF program, amely a már létező, PXL formátumú file-okat alakítja át EPF formátumúvá, a

```
pxl-epf amr10.pxl
pxl-epf -d amr10.pxl
```

minták alapján. Az első esetben egy 240 pont/inch felbontású EPF file keletkezik, míg a másodikban egy 120 pont/inch felbontású[7], mindkét esetben a megfelelő felbontású PXL file-t alkalmazva. A második lehetőség nem nagyon elterjedt, mivel 120 pont/inch felbontású PXL formátumú pixelfile-ok nemigen vannak forgalomban.

A nyomtató program meghívása most is a

```
dvi-eps input
```

példa szerint történik, ahol az input.dvi egy, az éppen érvényes alkönyvtárban levő file. Ez a meghajtó azonban sokkal többet tud a PCTEX-hez tartozó változatnál, így használata is bonyolultabb, meghívásakor sok lehetőséget lehet beállítani. Először is, még a használata előtt meg lehet hívni

[7] A 'draft' minőséget jelzi a -d jel.

az ugyancsak a programmal együtt árult `config.exe` file-t, amely a `config.eps`-ben tárolja azokat a beállításokat, amelyek a `dvi-eps` működését befolyásolják. Ebben sok beállítást elvégezhet a hozzáértő. Ezek nagyobb része a `dvi-eps` futtatásakor is beállítható, így itt nem írjuk le. Mindössze az 1. és 13. pont beállítása fontos, amely a számítógép és a nyomtató típusára vonatkozik. Az első pontban adjunk meg FX-et, ha EPSON-FX vagy RX nyomtatónk van, illetve MX-et, ha EPSON–MX (vagy ahhoz hasonló, például IBM Matrix) nyomtatónk. A 13. pontban adjunk IBM választ, ha PC DOS alatt fut a számítógép, és MS DOS választ, ha MS DOS alatt. Miután ez megtörtént, az „s" billentyű lenyomására az új beállítás lép érvénybe.

Amikor a `dvi-eps` programot meghívjuk, a DVI file után még több dolgot megadhatunk a programnak. Ezek teljes felsorolását mellőzzük, csak a legfontosabb lehetőségek szerepelnek az alábbiakban, egy-egy példával.

1. A `-q` után közvetlenül egy számot megadva a nyomtatás minőségének a fokozatát állíthatjuk be. Ez a szám egytől ötig terjedhet, és minél nagyobb, annál rosszabb lesz a minőség. Az alapértelmezés szerinti értéke 1, azaz a legjobb minőség.

```
dvi-eps input -q3
```

2. Az `-n` után egy számot (egytől háromig) megadva azt határozzuk meg, hogy melyik nyomtatón történjék a kinyomtatás. Alapértelmezése egy, és ennél nagyobb számot természetesen csak akkor érdemes megadni, ha több nyomtató van a géphez csatolva.

```
dvi-eps input -n2
```

3. A `-m` után egy számot megadva a TEX-ben szokásos `magstep` értékek szerint kinagyíthatjuk a nyomtatott szöveget. Alapértelmezése nulla. Ha tehát `-m2`-t adunk meg, akkor a meghajtó a TEX által létrehozott DVI file-t annak 1.44-szeresére nagyítja. Vigyázzunk ezzel a lehetőséggel, ha a TEX input file-ban is használtunk nagyításokat, ugyanis akkor a két nagyítás együttesen olyan nagy fokozatú betűket igényelhet, ami nem áll rendelkezésre.

```
dvi-eps input -m1
```

4. Az `-r` használata után megadhajuk, hogy mettől meddig nyomtassa ki a meghajtó a szöveget. Az egyes helyek megadása tizedes törttel történik,

amelyből az egész rész adja meg, hogy melyik oldalon akarjuk kezdeni, illetve befejezni a nyomtatást, a tizedes rész pedig, hogy annak melyik részén. Az 1.5 például az 1. oldal felét jelenti. A kezdő és befejező értéket kettősponttal választjuk el. A `-r1.33:2.5` megadása például azt jelenti, hogy az első oldal első harmadától a második oldal feléig nyomtassa ki a program a szöveget. A szöveg elejét illetve végét ekkor a `first` és `last` szavakkal adhatjuk meg. Az alapértelmezés szerint a teljes szöveg kinyomtatódik.

```
dvi-eps input -r2.66:last
```

5. A `-c` után egy számot megadva azt határozzuk meg, hogy hány példányban kérjük a nyomtatást. Alapértelmezés természetesen egy.

```
dvi-eps input -c25
```

Végül már csak azt említjük meg, hogy noha a megnövekedett bonyolultság miatt a hibaüzenetek száma itt jóval nagyobb, azok közül, amelyek nem valamely helytelen beállításra vonatkoznak (például amikor azt kérjük a programtól, hogy a 4. oldaltól a 2. oldalig nyomtasson ki egy szövegrészletet), a leggyakoribbak lényegében megegyeznek az előző pontban megadottakkal.

A.2.3. A Preview program

Már több olyan program is létezik, amely a DVI file-t képes a képernyőn megjelentetni, de ezek közül még a Preview a leggyakoribb, így csak ezt ismertetjük. Használatához valamilyen grafikus kártyára is szükség van. Ennek fajtája attól is függ, hogy a Preview melyik változatát használjuk. A leggyakoribb Hercules illetve EGA kártyákat bármely változat képes használni.

Megint csak az előző pontokban már leírtakra utalunk, feltételezzük azok ismeretét. A Preview program installálását nem egy installáló program végzi, hanem maga a felhasználó a DOS `restore` utasítása segítségével. Az eredeti lemezek három csoportba vannak osztva. Az első felirata `Installation Diskette`, amit a számítógép A meghajtójába helyezve, és a

```
restore a: c:*.* /s
```

parancsot kiadva vihetünk fel a merevlemezre. A második csoportba tartoznak a 80/118dpi PXL Diskette feiratú lemezek, végül a harmadikba a 300dpi PXL Diskette feliratúak. Ezeket ugyancsak a fenti módszer szerint másolhatjuk a merevlemezre, egymástól függetelenül. Az egyes csoportok átmásolásakor a gép mindig utasítja a felhasználót amikor lemezt kell cserélni. A lemezek feliratairól már kitalálhattuk, hogy a program 3 különféle betűfokozatot használ.

A 80, 188 és 300 pont/inch-es pixelfile-ok természetesen nem a felbontás sűrűségében térnek el egymástól (hiszen a képernyőnek csak egyetlen felbontása van), hanem a nagyításban. A 80 pont/inch felbontású file-okat használhatjuk majd arra, hogy az egyes oldalak általános külalakját megvizsgáljuk (de ilyenkor elég nehéz kivenni az egyes betűk képét). A 118 pont/inch-es felbontást már a sorok, betűk közelebbi megtekintésére is felhasználhatjuk, ezekkel a betűkkel már jól láthatók és megkülönböztethetők az egyes jelek. Normális méretű (tehát 10 vagy nagyobb pontos) betűk és jelek esetén ez a nagyítás már teljességgel elegendő a szöveg megvizsgálására és nemcsak a betű-, de a tipográfiai hibák észrevételére is. A 300 pont/inch-es felbontás már csak a betűképek közelebbi vizsgálatára, a nagyon kicsi fokozatú betűk megkülönböztetésére és a matematikai képletek esetében szükséges. Aki ilyeneket nem használ, vagy akinek bármely egyéb okból nem szükségesek a 300 pont/inch-es fokozatok, az ne is vigye fel ezeket a fontokat a merevlemezre, mivel igen nagy helyet foglalnak. Amennyiben valaki mátrixnyomtatóval nyomtatja ki a TEX által előállított szöveget, akkor az ahhoz szükséges fontok, és a Preview összes fontja együttesen közel 10 kilobyte helyet foglalnak el. Más a helyzet lézernyomtató esetén, ugyanis akkor a lézernyomtató pixelfile-jai is 300 pont/inch-es felbontásúak lévén, azok használhatók a Preview legnagyobb fokozataként.

Miután a lemezek felmásolása megtörtént, a TEXTSET alkönyvtárban helyezkednek el a Preview-hoz tartozó programok. A TEXTSET-en belül a minket érdeklő alkönyvtárak a PREVIEW, amely a programokat tartalmazza, illetve a PXLFILES, amely a PCTEX PIXEL illetve a MicroTEX TEXTFONTS alkönyvtárainak felel meg. Benne PXL400, PXL438, PXL480 és PXL560 nevű alkönyvtárak tartalmazzák a 80 pont/inch felbontású fontokat és PXL590, PXL646, PXL708, PXL850, PXL1020 és PXL1200 nevűek a 118 pont/inch felbontásúakat. Ha valaki a 300 pont/inch felbontású fontokat is felviszi

a merevlemezre, azoknak a különböző fokozatait a PXL1500, PXL1643, PXL1800, PXL2160, és PXL2592 alkönyvtárakban találja. A PREVIEW alkönyvtár tartalmazza a preview.exe file-t (amelyet egyébként célszerű egy olyan alkönyvtárba is átmásolni, amit megadunk a DOS path utasításában), a később ismertetett preview.cf és preview.fnt file-okat, és a segítő üzeneteket tartalmazó preview.hlp file-t.

A Preview nem használ saját TFM file-okat, mindig valamely más TEX program file-jait veszi igénybe. Ezt azért teheti meg, mert a felbontása valamennyi megjelenítő program között a legrosszabb, valamint az outputja nem kerül végső felhasználásra, így a betűk képe lehet kicsit életlen. Tehát TFM file-okat tartalmazó alkönyvtár nem keletkezik a merevlemezre történő másolás során.

A program felkészítése a futásra a fentiekkel még nem fejeződött be. Be kell állítani a preview.cf és preview.fnt file-okat az adott helyzetnek megfelelően. A Preview csak azokat a pixelfile-okat keresi, amelyek meg vannak adva a preview.fnt file-ban. Egy tetszőleges szövegszerkesztővel vizsgáljuk tehát meg ezt a file-t, és javítsuk ki a hiányosságokat. Abból baj nem lehet, ha egy olyan font is szerepel ennek felsorolásában, amely valóságban az adott konfigurációban nem létezik. Ha viszont például a hamr10.pxl file-unk létezik, azt meg kell adnunk a programnak a preview.fnt file-ban is, különben soha nem, fogja megtalálni. Tehát azokat a pixelfile-okat, amelyek a rendelkezésünkre állnak, de nem szerepelnek a preview.fnt-ben, megadjuk az alábbi minta szerint (minden új betűtípust egy külön sorban):

```
hamr10 400 438 480 576 590 646 708 850 1020
```

Persze a hamr file-okat előbb még létre kell hozni, és noha ezek PXL formátumúnak mondottak, kicsit különbőznek a PCTEX PXL formátumú pixelfile-jaitól, mert azoknál jobban tömörítettek. A Textset társaság a Preview programcsomagjait ezért két konvertáló file-lal árulja, amelyek közül az unpkpxl a Preview PXL formátumát a PCTEX PXL formátumává alakítja, a packpxl pedig fordítva. A MATEX-hel forgalmazott, annak az installálását elvégző file-ok a betűképek átalakítást (a magyar ékezetes betűk létrehozatalát) automatikusan elvégzik, de szükségük van ezekre a kisegítő file-okra. Ha azonban valaki maga szeretné a hamr pixelfile-okat létrehozni, akkor a mapix használata előtt és után a file-okat megfelelően konvertálnia kell. Ezeket a lépéseket teljes részletességgel a következő fejezet tartalmazza.

A Preview helyes futásához be kell állítanunk a `preview.cf` file-t is. Nem mondjuk el a benne levő összes paraméter jelentését, csak azokét, amelyeket mindenképp be kell állítanunk.

- a `magnification` paraméter azt mondja meg a programnak, hogy milyen nagyítású fontok állnak rendelkezésére. Alapbeállítása 400, 590 és 1500, így az a felhasználó, aki a legnagyobb fokozatú fontokat nem használja, törölje ki az 1500-at.
  Például: `magnification 400 590`
- a `magdefault` paraméter azt mondja meg, hogy az egyes oldalak megjelenésekor mekkora legyen a nagyításuk. A legjobb, ha ezt az értéket 590-re állítjuk.
  Például: `magdefault 590`
- a `graphicscard` paraméter azt mondja meg a programnak, hogy milyen grafikus kártyát használunk. A Magyarországon elterjedt kártyák közül a HERCULES-t `hercules`, vagy `hercules-half`-ként kell megadni, attól függően, hogy milyen módban használajuk. Az EGA-t `ega-mono`, vagy (a szineset), `ega-color`-ként, a TECMAR és OLIVETTI gépek grafikus kártyáit pedig `tecmar` illetve `olivetti`-ként.
  Például: `graphicscard hercules`
- a `tfmdirectory` paraméter változó mondja meg a PREVIEW-nak, hogy melyik alkönyvtárban találja a TFM file-okat. Itt tehát azt az alkönyvtárat kell megadni, ahol a használatban levő TEX változat a TFM file-jait tarja.
  Például: `tfmdirectory \pctex\textfms`
- a `memsize` paraméter tartalmazza a számítógép memóriájának a méretét. A Preview program már 512 kilobyte memóriával is képes futni, így ez az érték 512 vagy 640 lehet.
  Például: `memsize 512`

Már csak azt kell megtanulnunk, hogyan működtethetjük a programot. Erről sem írunk le mindent, csak a legfontosabb utasításokat. Meghívni a `preview input` paranccsal kell, ahol az `input.dvi` egy létező DVI file. Miután az első oldal megjelenik (a program elég lassú, egy XT gépen ez sokáig tarthat), a billentyűzetről irányíthatjuk a programot. A néhány legfontosabb billentyű hatását az alábbiakban írjuk el.

`"f"` Hatására a következő lap jelenik meg a képernyőn.

"b" Hatására az előző lap jelenik meg a képernyőn.

"l" Hatására a képernyő balra mozdul el a szöveg fölött a képernyő szélességének felével. Annak ellenére tehát, hogy az l betű az angol left (balra) szó kezdőbetűje, a kép jobbra fog elmozdulni a képernyőn. A control-l hatására az elmozdulás mértéke csak a képernyő szélességének egy tizede, míg az L billentyűre a képernyő egészen a szöveget tartalmazó elméleti lap bal széléig mozdul el.

"r" Hatására a képernyő az oldal fölött jobbra mozdul el. Egyebekben teljesen ugyanúgy viselkedik, mint a l billentyű.

"u" Hatására a képernyő az oldal fölött felfelé mozdul el a képernyő magasságának felével. Egyebekben az l mintájára viselkedik. Látszatra tehát, ebben az esetben az oldal lefelé mozdul el.

"d" Hatására a képernyő lefelé mozdul el, egyebekben minden úgy igaz rá, mint a u-ra.

"p" Hatására az előtte megadott számu oldal jelenik meg a képernyőn. Ezt két különböző módon adhatjuk meg. A 12p hatására a szöveg 12. oldala jelenik meg, míg a 12.p hatására a 12-nek számozott oldal (ami fizikai értelemben akár az első is lehet.)

"e" Hatására a szöveg utolsó oldala jelenik meg a képernyőn.

"m" Hatására a képernyőn levő oldal eggyel nagyobb nagyításban jelenik meg, persze ha megvannak a megfelelő fokozatú betűk. Így ha a szövegoldal az eredeti 590-es nagyításban volt a képernyőn, akkor most az 1500-as nagyításban jelenik meg. Ez meglehetősen sok időt vesz azonban igénybe, ugyanis az új oldalkép összeállítása olyan hatású, mintha egy új oldalra ugranánk.

"s" Hatására a képernyőn levő oldal eggyel kisebb nagyításban jelenik meg. Egyébként minden úgy igaz rá, mint az m-re.

"q" Ezzel lépünk ki a programból.

A képernyőn a szöveggel együtt egy kurzor is megjelenik. Ezt a a billentyűzeten levő, nyilakat ábrázoló gombokkal a szokásos módon mozgathatjuk a szövegen belül. Egy billentyű-lenyomásra a billentyű szerinti irányba 20 pixelt mozog a kurzor. Ha valamikor előtte lenyomjuk az 1 billentyűt, akkor onnan kezdve egyszerre csak 1 pixelnyit mozdul el, a 3 billentyű hatására pedig 30-at. Az F1 billentyű hatására megjelenik a képernyőn egy függőleges vonalzó a kurzoron keresztül, az F2 billentyű hatására pedig egy vízszintes vonalzó. Ezek a billentyűk ismételt lenyomására eltünnek, így tetszőleges

helyen hívhatók elő. A vonalzók a pica (a TEX pc mértékegysége) szerint vannak beosztva, így -„TEXszerűen" olvashatjuk le róluk az esetleg szükséges távolságokat.

B. függelék

A MATEX installálása

B.1. A MATEX lemez tartalma

A könyvhöz tartozó MATEX lemez a következő file-okat tartalmazza:

- MAEK.COM és
- MAEKE.COM billentyűzet átdefiniáló program
- MAPIX.EXE konvertáló program, amely a .TFM és a .PXL file-okat magyarítja.
- MACONV.BAT batch program, amelyik az automatikus konverziót végzi.
- MATFM.BAT a .TFM file-okat automatikusan konvertáló program.
- MAPCTEX.BAT a PCTEX-hez tartozó pk fileokat konvertálja.
- MAMICTEX.BAT a MicroTEX-hez tartozó .epf file-okat konvertálja.
- MAPREW.BAT a Preview-hoz tartozó .PXL file-okat konvertálja.
- MAREND.EXE rendező, tárgymutató készítő program.
- MPLAIN.TEX a MATEX használatához szükséges alap formátumfile, forrás (TEX) formában.
- MATEX.TEX a MATEX makrócsomagot tartalmazó file
- ELVAL.TEX a magyar szavak elválasztásához szükséges „szabályok" gyűjteménye.
- BORITO.TEX a könyv hátsó borítójának forrásszövege.
- MATEX.DOK a programcsomag kiegészítő dokumentációja.

Javasoljuk a felhasználónak, hogy az installálás megkezdése előtt a MATEX.DOK file-t olvassa el, mert ez tartalmazza a program működtetésére vonatkozó legfrissebb információkat. A könyvet ugyanis a nyomdai átfutási idő miatt hónapokkal a program végső változatának elkészítése előtt kellett leadnunk. A MATEX.DOK file tartalmazza azokat az egyszerűsítésekből, javításokból eredő változásokat, amelyek a könyv leadásakor még voltak készen. Ezek a változások nem jelentik azt, hogy a könyvben leírtak bármilyen kis mértékben is érvényüket vesztették volna, (hiszen a könyvet is MATEX programcsomaggal állítottuk elő) hanem csak arra vonatkoznak, hogy néhány további lehetőséget biztosítsanak a (legális) felhasználóknak.

B.2. Az installáláshoz szükséges programok

A felsorolt file-ok közül a következőkre van szükség a MATEX installálásához:

- MAEK.COM vagy MAEKE.COM
- MAPIX.EXE konvertáló
- MACONV.BAT batch program
- MATFM.BAT automatikus konvertáló program
- MAPCTEX.BAT a PCTEX-hez,
- MAMICTEX.BAT a MicroTEX-hez,
- MAPREW.BAT a Preview-hoz,

A lemezen található file-okon kívül a következő programok jelenlétére van szükség:

- FIND.EXE DOS rendszerprogram.
- PKTOPX.EXE és PXTOPK.EXE konvertáló programok, ha PCTEX-hel akarjuk a MATEX-et használni.
- EPF-PXL.EXE és PXL-EPF.EXE konvertáló programok, ha MicroTEX-hel akarjuk a MATEX-et használni.
- PXLPACK.EXE és UNPKPXL.EXE konvertáló programok, ha Preview-val akarjuk a MATEX-et használni.

Ezeknek a programoknak a beszerzésére nem kell külön gondot fordítani, mivel a `FIND.EXE` file-t minden MS DOS operációs rendedszerhez adják, a PXL file-okat konvertáló programokat pedig a megfelelő TEX nyomtató programokkal együtt árulják.

B.3. A billentyűzet átdefiniálása

Az első lépésben installáljuk a billentyűzet átdefiniáló programot. Ez egy memória-rezidens (memóriában megmaradó) program, amely a

Prt Sc

*

billentyűvel kapcsolható ki, ill. be. Elször másoljuk a programot a DOS redszerfile-okat tartalmazó alkönyvtárba, majd indítsuk el:

```
>copy a:maek.com c:\system
```

(Esetünkben a redszerfile-ok a `system` alkönyvtárban vannak.)

Feltételezzük, hogy a rendszert tartalmazó alkönyvtár `path` segítségével elérhető, ha ez nem volna így, adjuk ki a következő parancsot:

```
>path c:\system
```

Ezután indítsuk el a programot:

```
>maek
```

A program az elindítása után üzenetet ír ki a képernyőre:

```
>   maTeX ékezet
```

A billentyűzet ettől fogva át van definiálva, probáljuk ki például a \ billentyűt leütni, a kis hosszú í betűnek kell a képernyőn megjelenni. Ha vissza akarunk váltani az eredeti billentyűzetre, nyomjuk le a

Prt Sc

*

billentyűt. Ha megint magyar betűkre van szükségünk, ismét nyomjuk le a

Prt Sc

*

billentyűt és ezzel átváltunk „magyar" üzemmódra.

A MAEK.COM és a MAEKE.COM programok között csak annyi a különbség, hogy a MAEK.COM olyan billentyűzetet hoz létre, amelyen az Y és a Z betű az angol (QWERTY) billentyűzethez képest fel van cserélve; a magyar írógépeken ugyanis így (QWERTZ) van. A MAEKE.COM program viszont meghagyja ezeket eredeti (angol billentyűzet szerinti) helyükön.

Bekapcsoláskor, ill. új rendszer betöltésekor (boot-oláskor) újra be kell tölteni a programot. Ha az AUTOEXEC.BAT file utolsó sorába beleírjuk:

.

.

.

maek

akkor még ezzel sem kell törődnünk, a magyar betűk mindig elérhetőek lesznek.

Fontos figyelmeztetés: A program körülbelül 9 kbyte-ot használ el a memóriából, így — különösen, ha 512 kbyte-os gépünk van — lehetséges, hogy a korábban probléma nélkül futó nagyobb programok (köztük a tex is) már nem férnek el a memóriában. Ekkor újra be kell töltenünk a rendszert most már maek.com nélkül, és így kell futtatnunk a programot. A maek.com programot csak úgy lehet kitörölni a memóriából, hogy újratöltjük a rendszert.

A

Prt Sc

*

billentyű funkciója megváltozik: a továbbiakban a '*' karaktert csak az IBM üzemmódban, a SHIFT 8 billentyűk együttes lenyomásával érhetjük el. A

Prt Sc

*

billentyű képernyő kinyomtató funkcióját az átdefiniálás nem érinti, ez továbbra is (mindkét üzemmódban) használható. Azokon a billentyűzeteken, ahol a '*' és a 'Prt Sc' funkciókra külön billentyű szolgál a '*'-ot kell használni.

Az ékezetes betűk a képernyőn több-kevesebb szépséghibával jelennek meg. Ahhoz, hogy forrásfile elkészítésekor a képernyőn is szép magyar betűket lássunk a karaktergenerátort kell kicserélni.

A maTeX magyar ékezetes karaktereinek kódjai:

Kisbetű	Hex	Decimális	Nagybetű	Hex	Decimális
á	84	134	Á	8F	143
é	82	130	É	90	144
í	A1	161	Í	8B	139
ó	95	149	Ó	9B	155
ö	94	148	Ö	99	153
ő	93	147	Ő	9D	157
ú	97	151	Ú	80	128
ü	81	129	Ü	9A	154
ű	96	150	Ű	98	152

A MAEK és a MAEKE programok teljes leírását a MATEX lemezen levő MATEX.DOK file tartalmazza. Ebben tüntetjük fel azt is, hogy mely ékezetes karakterek kódjai térnek el a magyar szabványtól, és teljes billentyűzetképeket adunk a MAEK (MAEKE) programok használatához. Most csak az ékezetes betűkre szorítkozva megadjuk, hogy az angol billentyűzet mely billentyűjét lenyomva jelennek meg az ékezetes betűk:

kisbetű	billentyű	nagybetű	billentyű
á	'	Á	shift + '
é	;	É	shift + ;
í	\	Í	shift + \
ó	=	Ó	shift + =
ö	0	Ö	shift + 0
ő	[	Ő	shift + [
ú	]	Ú	shift +]
ü	-	Ü	shift + -
ű	`	Ű	shift + `

B.4. A TFM file-ok konvertálása

A következő lépés a TFM file-ok konvertálása. Ehhez a MAPIX.EXE és a MATFM.BAT programokra van szükségünk. Másoljuk át a lemezről a MAPIX.EXE és a MATFM.BAT file-okat a redszert tartalmazó alkönyvtárba (itt kell lennie a FIND.EXE programnak is):

```
>copy a:mapix.exe c:\system
>copy a:matfm.bat c:\system
```

Ezek után indítsuk el a MATFM batch programot a következőképpen

```
>matfm név
```

ahol a 'név' helyén megadjuk azt az alkönyvtárat, amely a TFM file-okat tartalmazza. Például a PCTEX esetén, ha a szokványos alkönyvtárszerkezetet követjük, és nem írtuk felül a PCTEX installálásakor létrehozott szerkezetet, ez a következő:

```
>matfm c:\pctex\textfms
```

a MicroTEX esetén pedig:

```
>matfm c:\tex\fonts
```

A program az elindítása után átnevezi file-okat, ekkor lehetséges, hogy a következő hibaüzeneteket kapjuk:

```
Duplicate filename, or file not found
```

Ez azonban csak annyit jelent, hogy nem rendelkezünk minden TFM filelal hanem csak egy részével annak, amelyre a MATFM.BAT program felkészült. Ha minden TFM file-ra ezt kapjuk, akkor valószínű, hogy rosszul adtuk meg a TFM-eket tartalmazó alkönyvtárat.

A TFM file-ok átnevezése után a MAPIX.EXE program végrehajtja azokat az átalakításokat, amelyek az ékezetes magyar betűk létrehozásához szükségesek.

A program az elindulásakor a következőket írja ki:

```
maPIX  program V2.0 (c) Magiszter
```

```
(nyomj e-t, hogy egyenkent konvertaljon)
```

Azt az üzenetet, hogy '`(nyomj e-t, hogy egyenkent konvertaljon)`' most nem kell komolyan vennünk, ennek csak akkor van jelentősége, ha a file-okat egyenként konvertáljuk (erről majd később részletesen is írunk). A program ezután elkezdi konvertálni a TFM file-okat, minden egyes konvertált TFM file-nál (mint például most az AMBX10.TFM file-nál) kiírja a következőt:

```
ambx10 .tfm
------------------------- a beolvasas megvan!
ambx10 .tfm?
------------------------- a kiiras megvan !
```

A konvertálás utolsó részében visszanevezi a file-okat, és ezzel elkészült a TFM konverzió. A konverzió néhány percig tart.

A konverzió nem konvertál minden egyes TFM file-t a megadott alkönyvtárban, hanem csak a „betű" fontokat, amelyek a magyarosítás szempontjából érdekesek. **A betű fontokat az AMR5, AMR6, ...,AMR10 file-ok kivételével felülírja.** Az AMR ékezet nélküli fontfile-ok változás nélkül megmaradnak, a font magyar változata (amelyet az \ant paranccsal érhetünk el) az újonnan elkészített HAMR5, HAMR6, ..., HAMR10 file-okba kerül.

Nagyon fontos, hogy a file-okat **konvertálás előtt mentsük ki lemezekre**, mert ha a konvertálás nem sikerül, az eredeti TFM file-ok elvesznek. (Az eredeti TFM file-okra sikeres konvertálás után valószínűleg már nem lesz szükségünk.)

Ha olyan TFM file-okat akarunk konvertálni, amelyekre a MATFM nincs felkészülve, a MATFM.BAT file-ban, a fontok felsorolásánál ezt a fontot is meg kell adnunk.

A TFM file-okat minden forgalmazott TEX programnál TFM kiterjesztésű file-ok formájában árulják. A bevezetésben említett pl alakú file-ok — bár ugyanazokat az információkat tartalmazzák, mint a TFM file-ok — nagyobbak, mert nem olyan tömör, hanem emberi szem számára is olvasható formában tárolják a betűk adatait. A MATFM és a MAPIX programok nem képesek kezelni a pl file-okat, ezért ezeket magyarítás előtt és után a megfelelő PLTOTF ill. TFTOPL programokkal konvertálnunk kell. Ha PL alakú file-okat kapunk bizonyos, hogy a megfelelő konvertáló programokat is megkapjuk.

B.5. Az MPLAIN formátum installálása

Az installálás következő lépése az MPLAIN.FMT formátumfile létrehozása. Ehhez a megfelelő (PCTEX vagy TEX) alkönyvtárban kell lennünk és ott kiadni az inicializáló utasítást. Mielőtt ezt megtennénk, másoljuk át az MPLAIN.TEX, a MATEX.TEX és ELVAL.TEX file-okat a lemezről a megfelelő TEX input file-ba. A PCTEX esetén a következő utasítást kell kiadnunk:

```
>copy a:*.tex c:\pctex\texinput
```

A MicroTEX esetében pedig:

```
>copy a:*.tex c:\tex\inputs
```

Az inicializálást a PCTEX esetében így hajtjuk végre:

```
>tex mplain/i
```

A TEX.EXE program megkeresi és inicializálja az MPLAIN.TEX és a MATEX.TEX makrócsomagot, majd az inicializálás után egy '*'-ot ír ki. Ekkor a '*' jel után be kell gépelnünk a \dump utasítást, ennek hatására létrejön az MPLAIN.FMT formátum file, amelyet be kell másolnunk a formátumfile-ok alkönyvtárába:

```
*\dump
>copy mplain.fmt texfmts
```

A MicroTEX esetében más a helyzet, mert itt külön programmal történik az inicializálás, nevezetesen az INITEX.EXE programmal. A MicroTEX által forgalmazott lemezeken a tex programban már benne van a PLAIN makrócsomag. Ezért először egy VIRTEX-et kell generálnunk, és ennek futtatása során kell betöltenünk a MPLAIN.FMT formátum file-t. A VIRTEX elkészítését az A. függelékben ismertettük. Az MPLAIN.FMT elkészítése:

```
>initex mplain
```

Az inicializálás megtörténtekor most is, mint a PCTEX-nél, kapunk egy '*'-ot, ekkor ugyanúgy a \dump utasítással kiiratjuk a formátum file-t, majd bemásoljuk a megfelelő formátumokat tartalmazó alkönyvtárba:

```
*\dump
```

.

.

```
>copy mplain.fmt formats
```

Ezután az MPLAIN.FMT ill. MPLAIN.LOG file-okat kitörölhetjük abból az alkönyvtárból, ahol voltunk, és ezzel kész az installálás egyik fele. A másik fele már csak a nyomtató programokat érinti, így tehetünk egy próbát (bár az eredményt nem tudjuk még kinyomtatni).

Ehhez másoljuk át a BORITO.TEX file-t a lemezről, és mint TEX forrásszöveget szedessük ki a TEX-hel. Például PCTEX-hel:

```
>copy a:borito.tex c:\pctex
>tex &mplain borito
```

Ha a program hiba nélkül lefut, és elkészíti az BORITO.DVI file-t akkor valószínüleg sikerült az installáció.

B.6. A PXL file-ok konvertálása

Az installálás negyedik lépése, hogy a nyomtatóprogramok által használt, a betűképeket tartalmazó file-okat átírjuk. Az átírás során az egyes fontokban a betűképek valamelyest megváltoznak. Ennek a változásnak összhangban kell lennie azzal az átírással, amelyet a TFM file-ban végeztünk. Ha a TFM és a PXL file-ok közül csak az egyiket konvertálnánk, a TEX nem a nyomtatásnál használt betűképek valódi magassága, szélessége, stb. alapján tördelné a szöveget, ezért csúnya, nyomtatási képet kapnánk.

A PXL file-ok — a TFM file-okkal ellentétben, amelyeket szinte mindig TFM formában kapunk meg, többféle file-formátumúak lehetnek. Ezek bármelyike használható a MATEX-hel, ha rendelkezünk a megfelelő PXL-be és vissza konvertáló programokkal. A konvertálást megkönnyítő bat programokat, csak a PK, EPF és az ID1002-es PXL formátumú fileok-hoz készítettük el. Azért ezekhez, mert a leggyakrabban ezekkel találkozik a felhasználó:

- a PCTEX-hez adott DVIDOT és PCDOT nyomtató programok, valamint a DVIJLA lézernyomtató program PK formában használja a PXL file-okat.

- a MicroTEX-hez adott nyomtatóprogram EPF formában használja a PXL file-okat.
- a TEXTSET Inc. cég által készített Preview, DVILaser HP nyomtató programok ID1002 alakú, PXL kiterjesztésű formátumot használnak.

Ennek megfelelően a MATEX lemezen a következő batch programok találhatók:

- MAPCTEX.BAT a PK formátumok magyarítására,
- MAMICTEX.BAT az EPF formátumok magyarítására,
- MAPREW.BAT az ID1002-es PXL formátumok magyarítására.

Az ezektől eltérő alakban levő PXL file-okat is lehet konvertálni a MAPIX programmal, de ehhez a felhasználónak át kell írnia valamelyik BAT programot, vagy egyenként kell konvertálnia a fontokat (ezt a következő szakaszban részletesen leírjuk).

Még mielőtt a konvertáláshoz hozzákezdenénk, egy fontos figyelmeztetés:

A konvertálás során az egyes fontfile-ok felülíródnak, ezért szükséges, hogy másolattal rendelkezzünk! Ha nincs másolatunk, akkor a konvertálás előtt mentsük ki a fontfile-okat lemezre!

A konvertáláshoz a MAPCTEX.BAT, MAMICTEX.BAT vagy a MAPREW.BAT; a MACONV.BAT, MAPIX.EXE és a FIND.EXE programokat tartalmazó file-okra van szükségünk.

Ezeket a programokat másoljuk át a rendszert tartalmazó alkönyvtárba. Az átmásolás után — ha a megfelelő nyomtatóprogram már installálva van — elindíthatjuk a konvertálást. A magyarítás nem egyszerre, hanem az egyes nagyításoknak megfelelő alkönyvtárak szerint történik. A PCTEX esetében a 0-ás nagyításhoz tartozó (240 dpi-s) fontokat így konvertáljuk:

```
>mapctex 1200
```

Majd ha ez lefutott, a többit is:

```
>mapctex 1315
>mapctex 1440
>mapctex 1728
>mapctex 2074
>mapctex 2488
```

```
>mapctex 2986
```

A képernyőn megjelennek konvertáló program üzenetei, majd bejelentkezik a MAPIX program, és a magyarított fontokat, néhány technikai adat kíséretében kiírja, majd ismét a konvertáló program üzenetei láthatók.

A magyarítás során az egyes fontok PXL alakúra konvertálódnak majd a MAPIX elvégzi a magyarítást, és eredeti alakjukra visszakonvertálódnak. Ha más felbontású fontokkal rendelkezünk, akkor az azokanak megfelelő számokat kell a programok meghívásánál beírnunk!

A Microtex esetében hasonlóan kell eljárnunk:

```
>mamictex 1315
>mamictex 1440
>mamictex 1728
>mamictex 2074
>mamictex 2488
>mamictex 2986
```

A Preview esetében hasonlóan:

```
>maprew 400
>maprew 438
>maprew 480
```

.

A konvertálás hasonlóan elvégezhető a DVIJLA, DVILaser HP, stb. programok esetében is.

A magyarítás a file-ok kétszeres (PXL-be és vissza) konvertálása miatt **több óráig is eltarthat**, ezért ha sikeresen lefutott érdemes a fileokat kimenteni, hogy ha felülíródnak, ne kelljen újra lefuttatni a programokat. A konvertálás befejezése után kitörölhetjuk a konvertáló programokat.

Ellenőrzésképpen nyomtassuk ki, az előző szakaszban elkészített BORITO.DVI file-t (pl. a PCTEX PCDOT programjával):

```
>dvieps borito
```

B.7. A MAPIX program

A MAPIX program két üzemmódban használható:

- az egyiket a fejezet előző részében működés közben láthattuk, ez a batch mód,
- a másik az egyedi mód, amikor a fontfile-okat egyenként konvertáljuk.

Ha a program bejelentkezését követően egy e-t nyomunk, a TFM és PXL file-okat egyenként konvertálhatjuk:

```
maPIX program V2.0 (c) Magiszter
(nyomj e-t, hogy egyenkent konvertaljon)
>e
a font neve: amtt10
TFM vagy PXL (t/p):p
milyen tip. (1-sima; 2-tt; 3-it/sl; 4-tt&sl):2
merete (byte-ban):1120
```

A program rákérdez a font nevére (kiterjesztés nélküli nevet kell megadnunk; majd arra, hogy TFM vagy PXL file-ról van-e szó; majd a típusára. Ez az információ azért kell a programnak, hogy az ékezeteket a megfelelő helyre tegye. (Egy dőlt betűnél pl. kicsit jobbra kell eltolni ahhoz, hogy szép legyen.) Végül a programnak meg kell adnunk a file méretét is. (Ha ezt nem tudjuk a DOS `dir` paranccsal kiírathatjuk a file-ok méretét a képernyőre.)

Négy típust különböztet meg:

- az egyenes, nem írógép típust, ekkor 1-et kell a programnak megadnunk;
- az egyenes, írógép típust, ekkor 2-est kell megadnunk;
- a dőlt, nem írógép típust, ekkor 3-ast kell megadnunk;
- a dőlt, írógép típust, ekkor 4-est kell megadnunk.

A fontok neveiből általában láthatjuk, hogy melyik típusba esnek:

- ha a névben TT szerepel, akkor ez írógép típusú, pl. AMTT10, 2-est kell megadnunk;

- ha a névben TI vagy SL szerepel, akkor ez dőlt típusú, pl. AMTI10, AMSL10, ekkor 3-ast kell megadnunk;
- ha a névben TT és TI (vagy SL) szerepel, akkor ez dőlt és írógép típusú, pl. AMSLTT10, ekkor 4-ast kell megadnunk;

Ne feledjük, hogy egy adott fontra mind a TFM file-t, mind a és PXL file-t konvertálnunk kell.

A különböző típusú fontokat a MAPIX pprogram nem egyformán konvertálja. Az írógéptípusú fontokból a magyarítás után is hiányoznak az Í és az Ú betűk. (A kisbetűk í, ú benne vannak.) Ennek oka az, hogy ezek ritkán fordulnak elő a szövegekben, másrészt a nekik megfelelő helyeken a < és a > jelek állnak, amelyek éppen ebben a betűtípusban gyakran használatosak. Ez természetesen nem jelenti azt, hogy nem tudunk írógéptípusban Í és Ú betűt írni, íme:

Í Ú

Ehhez azonban a kényelmetlen \'I és \'U parancsokat kell használnunk. Ha a TEX írógéptípusú betűk szedése közben < és > betűket talál, < és > jelet szed ki a helyükön.

A program csak a betűfontokat konvertálja, ha szimbólumfontokat konvertálunk vele, hibaüzenetet ad.

A program csak 65000 byte-nál kisebb PXL file-ok konvertálását tudja elvégezni, ez általában (300 dpi felbontásig) elég.

Ha ennél nagyobb file-ra alkalmazzuk hibaüzenetet ad. Szintén hibaüzenetet ad, ha nem (az általánosan használt) ID1001-es PXL file-ra alkalmazzuk. Ha a font ID1002-es PXL formátunú, a megfelelő konvertálást előbb el kell végeznünk, ahogy azt a fentiekban leírtuk.

Hibaüzenetet kapunk akkor is, ha az input/output műveletek nem sikerültek.

C. függelék
Fonttáblázatok

A következő néhány oldalon bemutatjuk a TEX legfontosabb betűtípusait, illetve az azokat tartalmazó fontokat. Minden fontnak az eredeti angol nevét adtuk meg, kivéve a magyar antikva betűtípust. Ennenk mi adtunk nevet, magyar voltát az angol elnevezésben szereplő „Hungarian" szó adja.

A függelék minden további oldalán egy-egy font táblázata szerepel. Mivel az egyes fontok 128 jelet tartalmaznak, a táblázatokat 16 × 8-as elrendezésben adjuk meg. Egy jel TEX kódját nyolcas számrendszerben a jel sorában és oszlopában található oktális számok összege adja meg. Tehát például a roman típusban a Æ kódja '035. A táblázatok alatt megadjuk a nyomdai szokások szerint az ábécék méreteit, és néhány szóban jellemezzük a betűtípusokat.

A matematikai fontok nem szövegek előállítására szolgálnak, így nem adjuk meg a bennük esetleg létező ábécé méretét sem.

Az 'Almost Modern Roman'
11 pontos font

	0	*1*	*2*	*3*	*4*	*5*	*6*	*7*
'000	Γ	Δ	Θ	Λ	Ξ	Π	Σ	Υ
'010	Φ	Ψ	Ω	ff	fi	fl	ffi	ffl
'020	ı	ȷ	`	´	ˇ	˘	¯	°
'030	¸	ß	æ	œ	ø	Æ	Œ	Ø
'040	‘	!	”	#	$	%	&	’
'050	(	)	*	+	,	-	.	/
'060	0	1	2	3	4	5	6	7
'070	8	9	:	;	¡	=	¿	?
'100	@	A	B	C	D	E	F	G
'110	H	I	J	K	L	M	N	O
'120	P	Q	R	S	T	U	V	W
'130	X	Y	Z	[	“	]	^	˙
'140	‘	a	b	c	d	e	f	g
'150	h	i	j	k	l	m	n	o
'160	p	q	r	s	t	u	v	w
'170	x	y	z	–	—	”	~	¨

abcdefghijklmnopqrstuvwxyz
Az ábécé hossza 139.74153pt.
A kisbetűk magassága 7.60416pt; mélysége 2.12917pt

ABCDEFGHIJKLMNOPQRSTUVWXYZ
Az ábécé hossza 203.86748pt; magassága 7.4825pt

Ez az ekezet nelkuli betutipus a legtobb TEX kiadvany alapja. Csak akkor van ra szuksegunk, ha az ekezetes betuk helyen allo gorog vagy kulonleges betuket hasznaljuk.

A 'Hungarian Almost Modern Roman'
11 pontos font

	0	*1*	*2*	*3*	*4*	*5*	*6*	*7*
'000	„	Á	á	é	É	í	ó	Ó
'010	ö	Ö	ő	ff	fi	fl	ffi	ffl
'020	ı	ȷ	`	´	ˇ	˘	¯	°
'030	¸	ß	Ő	ú	ü	Ü	ű	Ű
'040	-	!	”	#	$	%	&	’
'050	(	)	*	+	,	-	.	/
'060	0	1	2	3	4	5	6	7
'070	8	9	:	;	Í	=	Ú	?
'100	@	A	B	C	D	E	F	G
'110	H	I	J	K	L	M	N	O
'120	P	Q	R	S	T	U	V	W
'130	X	Y	Z	[	“	]	^	˙
'140	‘	a	b	c	d	e	f	g
'150	h	i	j	k	l	m	n	o
'160	p	q	r	s	t	u	v	w
'170	x	y	z	–	—	˝	~	¨

aábcdeéfghiíjklmnoóöőpqrstuúüűvwxyz
Az ábécé hossza 188.10394pt.
A kisbetűk magassága 7.60416pt; mélysége 2.12917pt

AÁBCDEÉFGHIÍJKLMNOÓÖŐPQRSTUÚÜŰVWXYZ
Az ábécé hossza 273.67366pt; magassága 10.37209pt

Ez a magyar antikva betűtípus. Az előző oldalon szereplő 'Almost Modern Roman' típustól a magyar ékezetes betűkben tér el. Ezek a görög nagybetűk és a skandináv ligatúrák helyén szerepelnek. A magyar nyelvű szövegek alapvető betűtípusa. Figyeljük még meg a *'000* helyén álló magyar kezdő idézőjelet.

Az 'Almost Modern Text Italic'
11 pontos font

	0	1	2	3	4	5	6	7
'000	„	Á	á	é	É	í	ó	Ó
'010	ő	Ö	õ	ff	fi	fl	ffi	ffl
'020	ı	ȷ	`	´	ˇ	˘	¯	°
'030	¸	ß	Ő	ú	ü	Ü	ű	Ű
'040	-	!	”	#	£	%	&	’
'050	(	)	*	+	,	-	.	/
'060	0	1	2	3	4	5	6	7
'070	8	9	:	;	Í	=	Ú	?
'100	@	A	B	C	D	E	F	G
'110	H	I	J	K	L	M	N	O
'120	P	Q	R	S	T	U	V	W
'130	X	Y	Z	[	“	]	^	˙
'140	‘	a	b	c	d	e	f	g
'150	h	i	j	k	l	m	n	o
'160	p	q	r	s	t	u	v	w
'170	x	y	z	–	—	˝	~	¨

aábcdeéfghiíjklmnoóöőpqrstuúüűvwxyz
Az ábécé hossza 177.69382pt.
A kisbetűk magassága 7.60416pt; mélysége 2.12917pt

AÁBCDEÉFGHIÍJKLMNOÓÖŐPQRSTUÚÜŰVWXYZ
Az ábécé hossza 270.80838pt; magassága 10.37209pt

Ez a kurzív betűtípus. Gyakran használják kiemelésekre, de esetleg egyes nagyobb egységeket is szedhetünk vele. Az antikva betűtípussal megegyező helyen tartalmazza a magyar ékezetes betűket, néhány más jelben azonban eltér attól. Figyeljük meg például, hogy itt £ áll azon a helyen, ahol az antikva fontban $

Az 'Almost Modern Bold Extended'
11 pontos font

	0	1	2	3	4	5	6	7
'000	„	Á	á	é	É	í	ó	Ó
'010	ö	Ö	ő	ff	fi	fl	ffi	ffl
'020	ı	j	`	´	ˇ	˘	¯	°
'030	¸	ß	Ő	ú	ü	Ü	ű	Ű
'040	‘	!	”	#	$	%	&	’
'050	(	)	*	+	,	-	.	/
'060	0	1	2	3	4	5	6	7
'070	8	9	:	;	Í	=	Ú	?
'100	@	A	B	C	D	E	F	G
'110	H	I	J	K	L	M	N	O
'120	P	Q	R	S	T	U	V	W
'130	X	Y	Z	[	“	]	^	·
'140	‘	a	b	c	d	e	f	g
'150	h	i	j	k	l	m	n	o
'160	p	q	r	s	t	u	v	w
'170	x	y	z	–	—	˝	~	¨

aábcdeéfghiíjklmnoóöőpqrstuúüűvwxyz
Az ábécé hossza 216.91765pt.
A kisbetűk magassága 7.60416pt; mélysége 2.12917pt

AÁBCDEÉFGHIÍJKLMNOÓÖŐPQRSTUÚÜŰVWXYZ
Az ábécé hossza 315.08134pt; magassága 10.25041pt

Ez a félkövér betűtípus. Az egyes karakterek ugyanott helyezkednek el benne, mint az antikva típusban. Leginkább címek szedésére használják. Itt is szerepel a magyar alsó idézőjel.

Az 'Almost Modern Typewriter'
11 pontos font

	0	1	2	3	4	5	6	7
'000	„	Á	á	é	É	1	ó	Ő
'010	ö	Ö	ő	↑	↓	'	¡	¿
'020	ı	j	`	´	˘	ˇ	¯	˙
'030	¸	ß	Ő	ú	ü	Ü	ű	Ű
'040	␣	!	"	#	$	%	&	'
'050	(	)	*	+	,	-	.	/
'060	0	1	2	3	4	5	6	7
'070	8	9	:	;	<	=	>	?
'100	@	A	B	C	D	E	F	G
'110	H	I	J	K	L	M	N	O
'120	P	Q	R	S	T	U	V	W
'130	X	Y	Z	[	\	]	^	_
'140	‘	a	b	c	d	e	f	g
'150	h	i	j	k	l	m	n	o
'160	p	q	r	s	t	u	v	w
'170	x	y	z	{	|	}	~	¨

aábcdeéfghiíjklmnoóöőpqrstuúüűvwxyz
Az ábécé hossza 201.20567pt.
A kisbetük magassága 6.69165pt; mélysége 2.43332pt

AÁBCDEÉFGHIÍJKLMNOÓÖŐPQRSTUÚÜŰVWXYZ
Az ábécé hossza 201.20567pt; magassága 8.66873pt

Ez az ún. írógép betűtípus, amelyet nevének megfelelően írógép vagy egyéb gépi jellegű betűk megjelentetésére használunk. Tartalmazza a legtöbb magyar betűt, de az Í és Ú hiányzik belőle.

Az 'Almost Modern Sans Serif'
11 pontos font

	0	1	2	3	4	5	6	7
'000	„	Á	á	é	É	í	ó	Ó
'010	ö	Ö	ő	ff	fi	fl	ffi	ffl
'020	ı	j	`	´	ˇ	˘	¯	°
'030	¸	ß	Ő	ú	ü	Ü	ű	Ű
'040	-	!	"	#	$	%	&	'
'050	(	)	*	+	,	-	.	/
'060	0	1	2	3	4	5	6	7
'070	8	9	:	;	Í	=	Ú	?
'100	@	A	B	C	D	E	F	G
'110	H	I	J	K	L	M	N	O
'120	P	Q	R	S	T	U	V	W
'130	X	Y	Z	[	"	]	^	˙
'140	`	a	b	c	d	e	f	g
'150	h	i	j	k	l	m	n	o
'160	p	q	r	s	t	u	v	w
'170	x	y	z	–	—	˝	~	¨

aábcdeéfghiíjklmnoóöőpqrstuúüűvwxyz
Az ábécé hossza 179.21454pt.
A kisbetűk magassága 7.60416pt; mélysége 2.12917pt

AÁBCDEÉFGHIÍJKLMNOÓÖŐPQRSTUÚÜŰVWXYZ
Az ábécé hossza 251.08922pt; magassága 10.18958pt

A sans serif betűtípus a magyarul groteszk vágásúnak nevezett betűcsalád képviselője a TEX fontjai között. Figyeljük meg, hogy az antikva családra jellemző, az egyes betűket alul és felül lezáró vízszintes szakaszok hiányoznak a groteszk betűknél.

Az 'Almost Modern Math Italic'
11 pontos font

	0	1	2	3	4	5	6	7
'000	Γ	Δ	Θ	Λ	Ξ	Π	Σ	Υ
'010	Φ	Ψ	Ω	α	β	γ	δ	ϵ
'020	ζ	η	θ	ι	κ	λ	μ	ν
'030	ξ	π	ρ	σ	τ	υ	ϕ	χ
'040	ψ	ω	ε	ϑ	ϖ	ϱ	ς	φ
'050	↼	↽	⇀	⇁	‹	›	▷	◁
'060	0	1	2	3	4	5	6	7
'070	8	9	.	,	<	/	>	⋆
'100	∂	A	B	C	D	E	F	G
'110	H	I	J	K	L	M	N	O
'120	P	Q	R	S	T	U	V	W
'130	X	Y	Z	♭	♮	♯	⌣	⌢
'140	ℓ	a	b	c	d	e	f	g
'150	h	i	j	k	l	m	n	o
'160	p	q	r	s	t	u	v	w
'170	x	y	z	ı	ȷ	℘	⇀	⇁

Ez az elsődleges matematikai jelkészlet. A benne található betűket nem szöveg szedésére, hanem matematikai jelekként használjuk. Így nem okoz problémát, hogy a magyar ékezetes betűk nem találhatók meg benne.

Az 'Almost Modern Math Symbols'
11 pontos font

	0	1	2	3	4	5	6	7
'000	−	·	×	*	÷	◇	±	∓
'010	⊕	⊖	⊗	⊘	⊙	◯	∘	•
'020	≍	≡	⊆	⊇	≤	≥	≼	≽
'030	∼	≈	⊂	⊃	≪	≫	≺	≻
'040	←	→	↑	↓	↔	↗	↘	≃
'050	⇐	⇒	⇑	⇓	⇔	↖	↙	∝
'060	′	∞	∈	∋	△	▽	/	∖
'070	∀	∃	¬	∅	ℜ	ℑ	⊤	⊥
'100	ℵ	$\mathcal{A}$	$\mathcal{B}$	$\mathcal{C}$	$\mathcal{D}$	$\mathcal{E}$	$\mathcal{F}$	$\mathcal{G}$
'110	$\mathcal{H}$	$\mathcal{I}$	$\mathcal{J}$	$\mathcal{K}$	$\mathcal{L}$	$\mathcal{M}$	$\mathcal{N}$	$\mathcal{O}$
'120	$\mathcal{P}$	$\mathcal{Q}$	$\mathcal{R}$	$\mathcal{S}$	$\mathcal{T}$	$\mathcal{U}$	$\mathcal{V}$	$\mathcal{W}$
'130	$\mathcal{X}$	$\mathcal{Y}$	$\mathcal{Z}$	∪	∩	⊎	∧	∨
'140	⊢	⊣	⌊	⌋	⌈	⌉	{	}
'150	⟨	⟩	|	‖	↕	⇕	\	≀
'160	√	∐	∇	∫	⊔	⊓	⊑	⊒
'170	§	†	‡	¶	♣	♢	♡	♠

Ez a matematikai jelek egy másik készlete. Tartalmazza például az ún. kaligrafikus betűket és a legtöbb matematikai relációjelet. Ezen, és az előzőn kívül is létezik még a TEX-ben matematikai jelkészlet, de ez a kettő a legalapvetőbb. Láthatjuk, hogy sok olyan jel előfordul itt, amit az 5. fejezetben megismertünk.

D. függelék

TEX parancsok listája

Ebben a függelékben felsoroljuk a könyvben ismertetett TEX és MATEX parancsokat, megadjuk azok rövid definicióját, és egy javasolt magyar elnevezést azok számára, akik jobban meg tudják jegyezni a magyar rövidítéseket. A listában nem szerepel az összes TEX parancs. Még azok sem mind szerepelnek, amelyek a könyvben meg vannak említve. Nem lenne sok értelme például a görög betűk előállítására használt parancsneveket felsorolni.

A magyar elnevezés csak azoknál a neveknél szerepel, ahol értelme van, azaz egy a magyar nyelvből vett rövidítés jobb az eredeti parancsnévnél. Mindenki létrehozhat magának egy egyszerű makrócsomagot, amelyben a `\let` parancs segítségével az angol neveknek a javasolt (vagy a saját maga által kitalált) magyar elnevezéseket feleteti meg.

A táblázatról annyit jegyzünk meg előljáróban, hogy az ékezetet kijelölő parancsokat mindig az *a* betűn szemléltetjük, anélkül, hogy a hatását körülírnánk. A matematikai módot mat. módnak rövidítjük. Azon parancsok esetében, amelyek egy-egy jelet hoznak létre, a megfelelő jel szerepel magyarázatként. A matematikai parancsoknak csak egy igen kis része szerepel ebben a listában.

`\alalegys`	másodrendű indexalegység	
`\alegys`	indexalegység	
`\alpont`	alpont megadása	
`\ant`	magyar antikva betűtípus	
`\accent`	ékezet	`\ekezet`
`\acute`	*á* ékezet mat. módban	`\vvesz`
`\advance`	emeli az értékeket	`\emel`

\advancepageno	eggyel növeli az oldalszámot	\lapszemel
\allowbreak	képletben megengedett törési hely	\soremel
\ast	*	\csillag
\atop	mat. módban egymás fölé helyezés	\fl
\backslash	\, forditott törtvonal	\bsl
\b	a̲ ékezet	\al
\bar	ā ékezet mat. módban	\von
\baselineskip	alapvonalak távolsága	\sortav
\bf	félkövér betűtípus	\fk
\bigbreak	jó hely az oldalemelésre	\nagyvlts
\bigskip	nagy függőleges beosztás	\nagybeo
\box	doboz	\doboz
\break	sor- vagy oldalemelés	\vlts
\breve	ă ékezet mat. módban	
\bullet	•	\korong
\bye	a program vége	\vege
\c	a̧ ékezet (cedilla)	
\cdot	· szorzópont mat. módban	\kpont
\cdots	··· mat. módban	\kpontok
\centerline	középre zárt sort készit	\ksor
\check	ǎ hacsek mat. ékezet	\hacsek
\cleartab	kitörli a tabulátorokat	\trltab
\columns	oszlopok	\oszlop
\copy	egy doboz másolása	\masol
\cr	sor vége (kocsi vissza)	\kv
\d	a̤ ékezet	
\dag	† tőrjel	\kereszt
\day	a dátum napja	\nap
\ddag	‡ dupla tőrjel	\dkereszt
\ddot	ä ékezet mat. módban	\ppont
\ddots	⋱ átlósan elhelyezett pontok	\xpontok
\def	a definíció utasítása	
\discretionary	kivételes elválasztások megadása	\kiveteles
\dot	ȧ ékezet mat. módban	\pnt
\dotfill	pontkitöltés	\pontktl
\dots	... nem mat. módban	\pontok

\eject	kötelező lapváltás helye	\lapvlt
\egys	indexegység	
\else	vagy	\vagy
\end	program vége másik módon	\end
\endinsert	ábra vége	\abrveg
\enskip	fél "em"-nyi vízszintes beosztás	\enugr
\fejezet	szövegegység	
\fi	a \if-et (ha) lezáró parancs	\ah
\folio	kiszedi az oldalszámot	\pagina
\font	betűkészlet	
\footline	lábsor	\lbsor
\footnote	lábjegyzet	\lbjegyzet
\gdef	globális definíció	
\global	globális értelemezés	
\goodbreak	oldal vagy soremelési hely	\emels
\grave	à ékezet mat. módban	
\H	ő „magyar" ékezet	
\halign	táblázatkészítő parancs	\tablaz
\hat	â ékezet mat. módban	\kalap
\hbox	„vízszintes" doboz	\vdoboz
\headline	élőfej	\fejsor
\hfil, \hfill	vizszintes kitöltés	\vkitl, \vkitll
\hidewidth	„eltitkolja" egy szó szélességét	\rejt
\hoffset	baloldali margó	\bmarg
\hrule	vízszintes vonal vagy téglalap	\vvonal
\hrulefill	kitöltés vízszintes vonallal	\vvonalkitl
\hsize	szedéstükör szélessége	\szlessg
\hskip	vízszintes beosztás	\vbeoszt
\if	ha	\ha
\ifcase	esetek	\esetek
\ifhmode	ha vízszintes mód	\haviz
\ifmmode	ha matematikai mód	\hamat
\ifnum	két szám összehasonlítása	\haszam
\ifodd	ha páratlan	\hapar
\ifvmode	ha függőleges mód	\hafug
\ind	index eleme	

\indexkesz	gyűjtse ki a szavakat az indexbe	
\indent	bekezdés mértéke	\beljebb
\input	input file megadása	
\it	italic, azaz kurziv betűtípus	\kr
\kelt	a dátum megadása	
\kern	adott vízszintes mozgatás	
\ketoszlop		
\ketoszlopvege		
\ldots	... mat. módban	\apontok
\leftline	balra zárt sor	\balsor
\let	legyen (definíció)	\legyen
\line	sor	\sor
\lower	doboz lejjebb helyezésére	\lejjebb
\magnification	a nagyítás mértéke	\nagyit
\magstep	a nagyítás mértéke ugrásokban	\nagyfok
\matrix	mátrix	\matrix
\medbreak	alkalmas hely az oldalemelésre	\kzepesvlts
\medskip	közepes függőleges beosztás	\kzepestv
\midinsert	ábra	\abr
\mit	mat. italic (kurzív) típus	\matkr
\month	a dátum hónapja	\honap
\moveleft	doboz balra mozdítása	\balra
\moveright	doboz jobbra mozdítása	\jobbra
\multispan	több táblázatoszlop helyettesítése	\fesz
\newbox	új doboz definiálása	\ujdob
\newcount	új számláló definiálása	\ujszam
\newdimen	új mérték definiálása	\ujmer
\nobreak	tiltja az oldal vagy soremelést	\nincsvlts
\noindent	a sort nem kezdi beljebb	\nembeljebb
\nopagenumbers	letiltja az oldalszámozást	\nincspagina
\number	kiszedi a számláló tartalmát	\szam
\obeylines	megtartja a soremelést	\sorokszerint
\or	vagy	\vagy
\output	az outputot létrehozó parancs	
\over	mat. módban tört	\trt
\pageinsert	egész oldalas ábra	\oldabr

\pageno	az oldalszámot tartalmazó számláló	\oldalsz
\par	uj bekezdést kezd	\ujbekezd
\parskip	a bekezdések közötti beoztás	\bekezdtav
\pont	szövegegység	
\proclaim	tétel, kijelentés	\kijelent
\quad	egy „em"-nyi vízszintes beosztás	
\qquad	két „em"-nyi vízszintes beosztás	
\raggedright	szabadsorosan	\szabs
\raise	doboz feljebb helyezésére	\feljebb
\rm	roman, azaz antikva betűtípus	
\setbox	létrehozza egy doboz tartalmát	\dobtar
\settabs	a tabulátorok beállítása	\tabul
\show	megmutatja a parancs jelentését	\mutasd
\sl	dőlt betűtípus	\dlt
\smallbreak	kicsit alkalmas hely oldalemelésre	\kisvlts
\smallskip	kis függőleges beosztás	\kistv
\szakasz	szövegegység	
\star	*	\vcsillag
\strut	„láthatatlan" függőleges pálca	\palc
\t	a͡a ékezet	
\tabskip	táblázatelemek közti távolság	\tbltav
\tartjegyz	kiszedi a tartalomjegyzéket	
\tartkesz	kigyűjti a tartalomjegyzék adatait	
\tilde	ã ékezet mat. módban	
\tmut	kiszedi a tárgymutatót	
\topinsert	az oldal tetején elhelyezendő ábra	\felabr
\tt	az írógépet utánzó betűtípus	
\underbar	aláhuzás	\alhuz
\underline	aláhuzás mat. módban	\alh
\vbox	„függőleges" doboz	\fdoboz
\vfil, \vfill	függőleges kitöltés	\fkitl, \fkitll
\voffset	felső margó	\fmargo
\vrule	függőleges vonal vagy téglalap	\fvonal
\vsize	az oldal magassága\magassg	
\vskip	függőleges beosztás	\fbeoszt
\year	a dátum éve	\ev

Bibliográfia

- *Donald E.Knuth:* Computers and Typesetting Published by Addison-Wesley Publishing Co.,Inc., Reading, Mass., 1986.
- *Donald E.Knuth:* The TEXbook Published by Addison-Wesley Publishing Co., Inc., Reading, Mass., and American Mathematical Society, Providence,R.I., 1986 edition.
- *Donald E.Knuth:* TEX: The Program Published by Addison-Wesley Publishing Co.,Inc., Reading, Mass., 1986.
- *Donald E.Knuth:* The METAFONTbook Published by Addison-Wesley Publishing Co.,Inh., Reading, Mass., and American Mathematical Society, Providence, R.I., 1986.
- *Donald E.Knuth:* METAFONT: The Program Published by Addison-Wesley Publishing Co.,Inc., Reading, Mass., 1986.
- *Donald E.Knuth:* Computer Modern Typefaces Published by Addison-Wesley Publishing Co.,Inc., Reading, Mass., 1986.
- *Arthur L.Samuel:* First Grade TEX: A Beginner's TEX Manual Originally published as Stanford Computer Science Department Report No. STAN-CS-83-985.
- *Michael Spivak:* The Joy of TEX Published by the American Mathematical Society, Providence, R.I., 1986.
- Leslie Lamport: LATEX: A Document Preparation System Published by Addison-Wesley Publishing Co.,Inc., Reading, Mass., 1985.
- *Michael Urban:* An Introduction to LATEX Originally prepared for the TRW Software Productivity Project, 1986.
- *L. Botway és C. Biemesderfer:* LATEX Command Summary Originally prepared for internal use at STSch.
- VAX Language-Sensitive Editor (LSEDIT) Quick Reference Guide for use with the LATEX Enviroment Originally prepared for internal use at Lear Siegler, Inc., Instrument Division, Grand Rapids, Michigan.

- Proceedings of the First European Conference on TEX for Scientific Documentation, Como, Italy, May 16-17, 1985. Published by Addison-Wesley Publishing Co.,Inc., Reading, Mass., 1985.
- *Ellen Swanson:* Mathematics into Type Published by the American Society, Providence, R.I., 1982.

Tárgymutató

\ 32, 38, 144
" 33
\" 37
32–34, 155, 178–180, 212
\# (#) 37, 212
$ 32–34, 118–119, 199, 200, 210
\$ ($) 36–37, 175
$$ 118, 141, 160
% 32-35
\%(%) 36–37
& 32–34, 70, 137, 139–140, 142, 143, 152–155, 159, 161, 206, 209
\& (&)37, 153, 209, 250
\' 37, 41, 46, 175
'' (")33
(32, 33, 38,113, 131
* 37
** 70
\+ 37, 152–153, 155
+ 33
\, 37, 127–128
,, 33, 34
\- 37, 38, 88, 176
\. 37, 233
/ 32, 33, 113, 122, 131
\; 37, 127–128
; 33, 269
\= 37, 233
\? (!) 33, 38, 112, 175, 222
@ 38, 215, 225
] 33, 38, 214, 269
^ 32, 34, 119–121, 125
\^ 37
_ 32–34, 119–121, 125, 144
_ 37
\` 37
` 33
{ 32–34, 43, 44–46, 113, 137
\{ 37, 113
\| 37, 115
} 32–34, 43, 44, 46,113
\} 37, 113
~ 32, 33, 91–93
\~ 37, 42, 92
£ 42
& 42

\aa (å) 41
\AA (Å) 41
\abstract 237
\active 215
\acute 112

\Acute 235
\address 236
\advance 79–80, 174, 184
\advancepageno 80
\ae (æ) 41, 42
\AE (Æ) 41
\affil 236
\alalegys 229–230
alapvonal 99, 101
\alegys 229–230
\aleph 115
algoritmus 83, 89, 99, 102, 104, 108
\align 235
\alignat 235
állapot (mód) 106
  belső függőleges 106
  belső vízszintes 106
  függőleges 106
  kiemelt matematikai 107
  matematikai 107
  vízszintes 106
\allowbreak 93, 106
\allowmathbreak 234
\alpha 114
\alpont 226
\alpontsz 226
\amalg (II) 115
AMBX10 54
AMEX10 54
AMMI10 54
AMR10 54
AMSL10 54
amsplain 244
amsppt 232, 234
AMSTEX 29
amstex.tex 244, 248
AMSY10 54
AMTT10 54
\and 232
\angle (∠) 115
angol ábécé 175
\ant 39, 40, 42, 54, 171, 271
\approx (≈) 116
\arccos 144
\arcsin 144
\arctan 144
\arg 144
\ast 115
\asymp (≍) 116
at 55
\atop 123–125
\author 236
autoexec.bat 243, 245, 268

\B 233
\b 41
\backslash 115, 131
\bar 112
\Bar 235
\baselineskip 50, 52, 81, 99, 100–102, 172
bat 242
BAT 273– 274
\batchmode 205
bekezdés 94, 98, 100, 102
  új 94
\beta 114
betűnagyság 100
betűtípus 20–21, 53
\bf 39, 40, 42, 54, 178
\Big 132
\big 132–133
\bigbreak 106

\bigcap 115
\bigcirc 115
\bigcup 115
\Bigg 132
\bigg 132
\biggl 132
\Biggl 132
\Biggm 133
\biggm 133
\biggr 132
\Biggr 132
\bigl 131
\Bigl 132
\bigm 133
\Bigm 133
\bigodot 115
\bigoplus 115
\bigotimes 115
\bigr 131
\Bigr 132
\bigskip 51, 62, 64, 101
\bigsqcup 115
\bigtriangledown 115
\bigtriangleup 115
\biguplus 115
\bigvee 115
\binom 235
\biwedge 115
\bmatrix 235
BORITO.TEX 70, 273
\bot 115
\bowtie (⋈) 116
\box 186
\brace 124
\brack 124
\break 92–94, 105
\breve 112
\Breve 235
\buffer 233
\bullet (•) 115
\bye 69, 165

\c 41
\cal 114, 232, 233
\Cal 232
\cap 115
\caption 234
\captionwidth 234
\cases 139–140
\cdot 113, 115
\centerline 62, 97–98, 105, 160, 166 200, 201
\cfrac 235
\check 112
\Check 235
checksum 255
\chi 114
\choose 124
címoldal 187
címsor 105
\circ (∘) 115
\cite 237
\clubsuit (♣) 115
cm 77
\columns 152–153
config.sys 208, 244
\cong (≅)116
\coprod (∐) 115
\copy 186
\copyright (©) 42
\cos 144
\cosh 144
\cot 144

\coth 144
\cr 137, 139–140, 143, 152, 157, 206, 207, 209
\csc 144
\cup (∪)115

\D 233
\d 41
\dag (†) 41, 65
\dagger 115
\dashv 116
\day 189–190
\ddag (‡) 42, 66
\ddagger 115
\ddot 112
\Ddot 235
\ddots 138
\def 63, 171–174, 182–183, 209
\define 234
definiálatlan parancs 197, 200
definíció 183
\deg 144
\delta 114
\Delta 114
depth 151
\det 144
\dfrac 235
\diamond (⋄) 115
\diamondsuit (♢) 115
\dim 144
\dimen\footins 166
\discretionary 89
\displaystyle 135
\div 115
doboz 56–59, 147,184
  mérete 147
\documentstyle 234
\dot 112
\Dot 235
\doteq (≐) 116
\dotfill 151
\dots 42, 151
\downarrow (↓) 116, 131
\Downarrow 116, 131
\dsize 234
\dump 243, 247, 272
Duplicate filename, or file not found 270
DVI file 19, 22, 25, 71, 73, 98, 246, 251, 252
dvi-eps 257

egyenes vonal 150
\egys 227, 229, 230
\eject 105
ékezetes betűk 175
\ell (ℓ) 115
élőfej 77, 79, 80–82, 191
\else 186–187, 189–190
elval.tex 222, 243, 247, 272
elválasztás 84, 87, 90, 170
  kivételes 88
  oldalvégi 102
\emb 169
\emptyset (∅) 115
\end 69, 165
\endcfrac 235
\enddocument 237
\endinsert 164
\endtopmatter 236
\enskip 51–52, 154
\enspace 180–181
EPF file 265, 273, 274
\epsilon 114

EPSON 256
\eqalign 140–143
\eqalignno 142
\equiv (≡) 116
\errorstopmode 206
\eta 114
\exists 115
\exp 144

fattyúsor 102, 103
\fejezet 226
felkiáltójel 67, 102, 112, 233,
feltétel 186
feltételes utasítás 183
\fi 186–189
FIND.EXE 270, 274
\fivebf 54
\fivei 54
\fiverm 54
\fivesy 54
\flat 115
FMT file 170, 207, 219, 221, 241, 243, 245, 247
\folio 53–55, 80–81, 191
FONTS alkönyvtár 246, 247, 250
\footline 80–81, 187–188
\footnote 65, 166, 194–195
\forall 115
FORMATS alkönyvtár 246, 247, 250
formátum 30, 70, 170, 244, 250
formátumfile 30, 70
\frac 235
\frown (⌢) 116
fülszöveg 61, 63

\gamma 114
\Gamma 114
\gcd 144
\gdef 183–184
\ge (≥) 116
\geq 116
\gets 116
\gg 116
\global 183–184, 195
\goodbreak 93–94, 106
görög betűk 113–114
\grave 112
\Grave 235

\H 41, 42
\halign 155–158
HAMR10 54
\hat 112
\Hat 235
\hatant 45
hatvány 119–122
\hbar 115
\hbox 58, 59, 92, 106–108, 147–150, 157
\headline 80–81, 191
\heartsuit (♡) 115
height 151
\hetant 54
\hfil 80–81, 91, 94, 154, 159, 187–188
\hfill 151, 154, 198, 204
\hfuzz 86
hiba 197
  javítás 201
    interaktív 202
hibaüzenet 73, 85, 107, 197
\hidewidht 158
\hoffset 78, 192
\hom 144

\hookleftarrow (↩) 116
\hookrightarrow (↪) 116
hosszú sor 86
\hrule 108, 150–151
\hrulefill 81, 151
\hsize 47–50, 58, 77–78, 184, 194, 198, 199
\hskip 51, 55, 108
\hss 80–81
hyphen.tex 222
\hyphenation 89, 210, 211

\idotsint 233
\if 186–190
\ifcase 189–190
\ifdim188
\ifhmode 189
\ifhmode 189
\ifmmode 189
\ifnum 187–188
\ifodd 188
\ifvmode 189
\iiiint 233
\iiint 233
\iint 233
\Im 115
\imath 115
\impliedby 233
\implies 233
\in (∈) 116
\inaccessible 209
\ind 227–230
\indent 95, 160
\indexkesz 227–229
\inf 144
\infty (∞) 115
INITEX 219, 220, 221, 223, 240, 241, 242, 246, 247, 248, 272
\input 169, 198, 202, 203, 205, 208, 220, 221, 222, 223
INPUTS alkönyvtár 246, 247
INSTALL.BAT 246, 247
\int (∫) 125, 128
Interruption 245
INX file 228–230
\iota 114
írógép stílus 90
\it 39, 42, 43, 54, 164
\item 66–68, 96–97, 237
\itemitem 68

jelval.tex 242
\jmath 115

\kappa 114
\kelt 171, 189, 225
\ker 144
\kern 63, 107–108
\ketoszlop 231
\ketoszlopvege 231
\keywords 236–237
kipontozás 151
KIS file 231
kitevő 119–122
kizárás 97
komment-jel (&) 35
konyv.sty 222
környezet 43–46
kötőjeles szó 87, 88, 92
kurzív sorkiegyenlítés 23, 38

\l (ł) 41
\L (Ł) 41

\labj 195
lábjegyzet 65, 77
\labjszam 195
lábsor 77, 79, 80–82, 187, 191
\lambda 114
\Lambda 114
\land 115
\langle (⟨) 131
lapemelés 102–106
\large 223
latex.tex 223
LaTeX 29
\lbrace ({) 131
\lbrack ([) 131
\lceil (⌈) 131
\ldots 138
\le (≤) 116
\left 133–135, 137, 142
\leftarrow (←)116
\Leftarrow 116
\leftharpoondown (↽) 116
\leftharpoonup (↼) 116
\leftline 97–98
\leftrightarrow (↔) 116
\Leftrightarrow 116
\leftskip 96
\leq 116
\leqalignno 143
\let 183–184
\lfloor 131
\lg 144
ligatúra 23, 32, 45
\lim 144
\liminf 144
\limits 126
\limsup 144
\line 97
\lineskip 99, 101–102, 172
\lineskiplimit 99, 101, 172
\ll 116
\llap 150
\ln 144
\lnot 115
\log 144
LOG file 71, 75202, 203, 204, 213, 216, 246
\long 182–184
\longleftarrow (⟵) 116
\Longleftarrow 116
\longleftrightarrow (⟷) 116
\Longleftrightarrow 116
\longmapsto (⟼) 116
\longrightarrow (⟶) 116
\Longrightarrow 116
\lor (∨) 115
\lower 149
lplain 220, 223, 231, 243

MACONV.BAT 274
MAEK 211, 268, 269
MAEKE 266, 268, 269
\magnification 48, 55–56, 100
\magstep... 48, 55–56, 100
\magstephalf 48, 56
magyar ékezetes betűk 176
makró 29, 93, 169
  file 169
  csomag 29–30, 47
MAMICTEX.BAT 274
MAPCTEX.BAT 274
MAPIX 241, 260, 270, 271, 274, 275
MAPREW.BAT 274
\mapsto (↦) 116

marend 228, 230
margó 77–78, 192
matematikai ékezet 111
matematikai mód 111, 199
matematikai kurzív 111
MATEX.DOK 269
MATEX 31–32, 34, 36, 37, 39, 41, 42, 62, 63, 64, 70, 84, 88
MATEX.TEX 175, 189, 211, 223, 225, 242, 243, 247, 272
MATFM.BAT 270, 271
\mathbreak 234
\matrix 136–137, 140, 235
\max 144
\medbreak 106, 180
\medskip 51, 101–102
mérték 184
mértékegység 47, 73, 74, 198
MicroTEX 19, 28, 71
\mid 116
\midinsert 164–165
\midspace 233, 234
\min 144
minus 67
\mit 233
mm 77, 158
\models (⊨) 116
\month 189–190
\moveleft 149
\moveright 149
\mp 115
mplain (MATEX) 46, 47, 51, 53, 54, 69, 70, 166, 169, 170, 172, 175, 176, 193, 194, 195, 210, 211, 215, 216, 217, 220, 222, 223, 225, 231, 232, 240, 241, 242, 243, 244, 247, 248, 249, 272
MPLAIN.FMT 273
MPLAIN.LOG 273
\mu 114
\multispan 159
\multline 235

\nabla (∇) 115
nagyítás 48–49, 55–56, 62
\narrower 95–96, 173–174, 183–184, 195
\natural 115
\nearrow (↗) 116
\neq (*neq*) 115, 116
\newbox 186
\newcount 185, 192, 194
\newdimen 185
\newif 190
\ni 116
\noalign 157
\nobreak 93, 105, 107–108
\nofrills 237
\noindent 64, 68, 94–96, 227
\nolimits 126
\nomathbreak 234
\nonstopmode 205
\nopagenumbers 53, 75, 79, 81
\normalbaselines 172–173
\normalbottom 104–105
\not 116
\notin (∉) 116
\nu 114
\number 185, 190
\nwarrow (↖) 116
\nyolcant 195

\o (ø) 42
\O (Ø) 42
\obeylines 94
\odot 115
\oe (œ) 41
\OE (Œ) 41
\oint (∮) 125
oldal
  szélessége 77
  magasassága 77
  számozás 52–53, 79, 80, 82, 187
\oldnos 233
\oldstyle 233
\omega 114
\Omega 114
\ominus (⊖) 115
\omit 158–159
\oplus (⊕) 115
\or 189–190
\oslash (⊘) 115
\otant 54
\otimes ((⊗) 115
\outer 183–184
\output 98, 193
\over 122–124
Overfull 104, 231
  \hbox 85, 86, 90, 91, 148
  \vbox 165
\overfullrule 86–87
\overleftarrow 129
\overline 129
\overlong 236
\overrightarrow 129
\overset 235
\owns (∋) 116

\P (¶) 42
packpxl 260
\pagebody 193
\pageinsert 164–165
\pageno 52, 79–80, 185, 187, 191
\par 94–96, 108, 180–182, 199, 200, 210
\parallel 116
paraméter 46–48, 178, 184
parancsjel 38, 83, 84, 87, 89, 176
parancs 36
parancsszó 38–42
\parindent 46, 48, 49, 52, 94–96, 174
\parskip 50, 52, 66, 102
\partial 115
PATH 243, 245, 260, 267
pc 77
PCTEX 28, 71, 242–246, 253272
\perp 116
\phi 114
\Phi 114
\pi 114
\Pi 114
PIXEL alkönyvtár 253, 259
PK 265, 273, 274
pktopx.exe 254
plain 29, 46, 47, 51, 53, 169, 170, 172, 173, 174, 175, 176, 179, 187, 188, 193, 195, 210, 211, 215, 220, 222, 223, 224, 231, 232, 233, 235, 240, 241, 243, 244, 246, 247, 248
\plainoutput 193
PLTOTF 271
\pm 115
\pmatrix 138, 235

\pont 226
\pontsz 226
\Pr 144
\prec 116
\preceq 116
\predefine 234
preload 246–249
PREVIEW 259, 260
preview.cf 260–261
preview.fnt 260
\prime 115
primitív utasítás 29, 169
\proclaim 179–181, 183–184
\prod (∏) 115
\propto (∝) 116
\psi 114
\Psi 114
pt 77, 199
PXL file 25–27, 55, 73, 265, 273, 274, 275, 277
PXLFILES alkönyvtár 259
pxlpak.exe 254
pxtopk.exe 254
PXTOPK.EXE 266

\qquad 128, 141–142, 154, 158
\quad 108, 128, 154

\raggedbottom 104
\raggedright 90–91, 94, 104
\raise 149, 186
\rangle (⟩) 131
\rbrace (}) 131
\rbrack (]) 131
\rceil (⌉) 131
\Re 115
\redefine 234
\Refs 237
restore 258
\rfloor (⌋) 131
\rho 114
\right 133–135, 137, 142
\rightarrow (→) 116
\Rightarrow 116
\rightharpoondown (⇁) 116
\rightharpoonup (⇀) 116
\rightleftharpoons (⇌) 116
\rightline 97–98
\rightskip 96
\rlap 150
\rm 39, 42
római
  oldalszám 79
  szám 79–80, 192
\romannumeral 192
\root...\of 130
\roster 237

\S 41, 42
\sanss 53
\sb 121
\Sb 235
\scriptscriptstyle 136
\scriptstyle 136
\scrollmode 205
\searrow (↘) 116
\sec 144
\setbox 186
\setminus 115
\settabs 152–153
\sevenbf 54
\seveni 54
\sevenrm 54
\sevensy 54

\sharp 115
\show 172, 176, 216
\sigma 114
\Sigma 114
\sim (∼) 116
\simeq (≃) 116
\sin 144
\sinh 144
\sl 39, 42, 153
\smallbreak 106
\smallint 115
\smallmatrix 235
\smallskip 51, 101
\smc 233
\smile (⌣) 116
soremelés 87–93, 102
  befolyásolása 91
  kikényszerítése 92
sorkizárás 83, 90, 93
sortávolság 50, 66, 67, 98–99
sortördelés 95
\Sp 121, 235
\spadesuit (♠) 115
\span 159
speciális sorok 94
\split 235
spread 147–149
\sqcap (⊓) 115
\sqcup (⊔) 115
\sqrt 129
\sqsubseteq (⊑)116
\sqsupseteq (⊒)116
\ss 42
\ssize 234
\sssize 234
\star 115
stílus 221
\strut 136, 161
STY file 221, 224, 232, 234
\subjclass 237
\subset (⊂) 116
\subseteq (⊆) 116
\succ 116
\succeq 116
\sum (∑) 115, 125, 128, 144
\supset (*supset*) 116
\supseteq (⊇) 116
\surd 115
\swarrow (↙) 116
\syntax 234

\szakasz 226
\szakaszm 226
\szaksz 226
számlálók 52
szóköz 40, 81, 83, 92, 112
szövegformázó 16
szövegszerkesztő 16, 74, 201
szövegszedő 16

\t 42
\tabskip 156–157
\tag 234
\tagsOnleft 234
\tagsOnright 234
\tan 144
\tanh 144
TAR file 227
\tartjegyz 227, 229
\tartkesz 227
\tartm 227
\tau 114
téglalap 150

\tenbf 54
\tenex 54
\teni 54
\tenit 54
\tenrm 54
\tenrm 81
\tensl 54
\tensy 54
\tentt 54

tex.exe 207, 242, 243, 245, 246, 247, 248, 249
tex.poo 242, 245, 247
TEXFMTS alkönyvtár 242, 243, 244, 245, 246
TEXFONTS alkönyvtár 256
TEXINPUT alkönyvtár 242, 243, 244, 245, 246
TEXINST.BAT 242
TEXTFMS alkönyvtár 242, 243, 245, 246
TEXTFONTS alkönyvtár 259
TEXTSET 259
\textstyle 135
TFM file 21, 23, 26, 53, 54, 207, 240, 241, 242, 245, 250, 252, 253, 265, 270, 271, 273
TFTOPL 271
\thanks 236
\theta 114
\Theta 114
\tilde 112
\Tilde 235
\times (×) 115
tipográfia 100, 102
\title 236
\tizant 54, 81
tizedespont 51
\tmut 228–229, 231
to 58, 147–149
\to 235
to be read again 199
\top 115
\topinsert 164–166
\topmatter 236
\topspace 233–234
\tracingcommands 216
\tracingmacros 216
\triangle (△) 115
\triangleleft (◁) 115
\triangleright (▷) 115
TRT file227
true 49, 99
\tsize 234
\tt 39
\ttraggedright 90–91

\underbar 43–45, 67, 128–129
Underfull 104, 231
  \hbox 85, 90, 93, 148
  \vbox 105
\underline 128
\underset 235
UNPKPXL.EXE 260, 266
\uparrow 116, 131
\Uparrow 116, 131
\updownarrow 116, 131
\Updownarrow 116, 131
\uplus 115
\upsilon 114
\Upsilon 114
üres sor 68, 69, 199
utasításnevek 174

\vadjust 105
változó 178, 183, 200
VANILLA 29
\varepsilon 114
\varGamma 232
\varphi 114
\varpi 114
\varrho 114
\varsigma 114
\vartheta 114
\vbox 58–59, 104, 107–108, 147–150, 160
\vdash 116
\vdots 138
\vec 112
\Vec 235
védjegy 65
\vee 115
\Vert 115,131
\vert 131
vezérlő karakterek 32–33
virtex.exe 249, 250
\vmatrix 235
\Vmatrix 235
\voffset 78, 81
\vrule 150–151, 160–161
\vsize 47–49, 58, 73, 77–78, 104–105
\vskip 50–51, 55, 93, 101–102, 108, 149, 164, 229
\vtop 149–150

\wedge 115
width 151
\wp 115
\wr (≀) 115

\xi 114
\Xi 114

\year 189–190

\zeta 114

Bibliotheca Academiae Scientiarum Hungaricae
MAGYAR TUDOMÁNYOS AKADÉMIA KÖNYVTÁRA

AKADÉMIAI KIADÓ ÉS NYOMDA VÁLLALAT
Felelős kiadó: Hazai György főigazgató
Felelős szerkesztő: Votisky Zsuzsa
A borítót Szabadi Zoltán tervezte
ISBN 963 05 4869 0
A kézirat lezárva: 1988. február
Készült az OPTIMUM Kisszövetkezet közreműködésével
MGKSZ Nyomdaüzem (88.125)
Felelős vezető: Mázi Ferenc

2902

150

Magiszter Számítástechnikai Szerkesztőség

MATEX programcsomag

A MATEX programcsomag IBM PC/XT/AT kompatibilis számítógépeken futó PCTEX és MicroTEX valamint a Preview programokat teszi alkalmassá arra, hogy magyar nyelvű szövegeket lehessen könnyen, gyorsan, szépen — a TEX szövegszedő program minden lehetőségét felhasználva — előállítani. A programcsomag egyszerűen implementálható; futtatható minden olyan számítógépen, amelyen a PCTEX, a MicroTEX vagy a Preview program fut, tehát semmilyen külön hardware igénye nincs. A MATEX programcsomag az alábbi szolgáltatásokat tartalmazza:

‡ Segítségével új, ékezetes betűket is tartalmazó betűcsaládokat lehet készíteni minden betűtípusban és nagyságban. (**Például ékezetes, nem proporcionális betűtípust is, amely a számítástechnikai könyvekhez szükséges.**)

‡ Használatával elkerülhető az a nehézkes megoldás, amely az ékezetes betűket egy preprocesszor segítségével TEX parancsokká alakította át. A MATEX alkalmazásával a TEX az ékezetes betűket közvetlenül megérti, nem kell hozzá preprocesszor !

‡ Tartalmaz egy billentyűzet átdefiniáló programot is, amely az ékezetes betűk bevitelét kényelmessé teszi.

‡ Megtanítja a TEX-et a magyar szavak helyes elválasztására, így magyar szöveget is teljesen automatikusan tud a TEX sorokra, lapokra tördelni.

‡ Önállóan készíti el és szedi ki a tartalomjegyzéket, a megjelölt szavakból pedig kéthasábos, rendezett tárgymutatót készít.

‡ Más fontos magyar szolgáltatásokat is biztosít, mint például:

- az automatikus keltezés, fejezetcím készítés és számozás négy címfokozatig,
- a magyar szokásoknak megfelelő „idézőjelek", és fejlécek készítése,

és még sok egyéb hasznos dolog.

A MATEX programcsomag **megőrzi a rendszer TEX kompatibilitását.** Így például megőrzi a TEX egyedülálló matematikai szedési lehetőségeit is:

$$\lim_{z\to\frac{1}{n\pi}}\left(z-\frac{1}{n\pi}\right)\operatorname{cosec}\left(\frac{1}{z}\right)=\frac{(-1)^{n+1}}{n^2\pi^2},\qquad S(n,k)=\frac{1}{k}\sum_{j=1}^{k}(-1)^{k-j}\binom{k}{j}j^n$$

Ez a könyv az első magyar nyelvű TEX ismertető. Kis terjedelme ellenére több célt is szolgál: bevezető a TEX-kel ismerkedőknek, hasznos kézikönyv a gyakorlott felhasználóknak és emellett a MATEX programcsomag teljes dokumentációja.

©IBM az International Business Machines Corp. jegyzett védjegye; ©TEX az AMS (Amerikai Matematikai Társulat) védjegye; ©PCTEX a Personal TeX vállalat védjegye; ©MicroTEX az Addison–Wesley Publishing Company védjegye; ©Preview a Textset, Inc. vállalat védjegye; ©MATEX az Akadémiai Kiadó védjegye

1828