KFKI-1981-84

DUBLA L.
KULCSÁR K.

MORTRAN és RATFOR
A FORTRAN NYELV KÉT BŐVÍTÉSE

Hungarian Academy of Sciences

CENTRAL
RESEARCH
INSTITUTE FOR
PHYSICS

BUDAPEST

KFKI-1981-84

M O R T R A N és R A T F O R
A FORTRAN NYELV KÉT BŐVÍTÉSE

Dubla László, Kulcsár Katalin

Központi Fizikai Kutató Intézet
1525 Budapest 114, Pf. 49

HU ISSN 0368 5330
ISBN 963 371 866 X

Tartalomjegyzék

	oldal
Bevezetés	3
1. MORTRAN	6
1.1 Kódolási szabályok	6
1.2 A nyelv szerkezete	7
1.2.1 Utasitás	7
1.2.2 Blokk	7
1.3 Feltételes utasitások	7
1.3.1 IF utasitás	7
1.3.2 IF ... ELSE utasitás	8
1.3.3 ELSEIF utasitás	8
1.4 Ismétlődő utasitások	9
1.4.1 A WHILE utasitás	9
1.4.2 LOOP utasitás	9
1.4.3 UNTIL utasitás	10
1.4.4 FOR utasitás	10
1.4.5 DO utasitás	10
1.4.6 Végtelen ciklus	11
1.5 Többszörös értékadás	12
1.6 Makrók használata	13
1.7 Vezérlő utasitások	15
1.8 A MORTRAN Job Control eljárások ismertetése	16
1.9 Korlátozások, figyelmeztetések	17
1.10 Egy MORTRAN program	18
2. RATFOR	19
2.1 Kódolási szabályok	19
2.2 A nyelv szerkezete	20
2.3 Feltételes utasitások	21
2.3.1 Az IF utasitás	21
2.4 Ismétlődő utasitások	22
2.4.1 WHILE utasitás	22
2.4.2 A FOR utasitás	22
2.4.3 A REPEAT-UNTIL utasitás	23

BEVEZETÉS

Az intézetünkben használt software termékek legnagyobb része tudományos célu program, melyeket intézetünk kutatói dolgoztak ki, illetve amelyeket más hazai vagy külföldi intézetektől vettünk át. Ezeknek a programoknak jelentős része FORTRAN-ban van irva, annak ellenére, hogy ismerjük a FORTRAN nyelv hiányosságait. Más, ma forgalomban lévő magasszintü programozási nyelvek /pl: PASCAL/ ezeket a hiányosságokat nagyrészt kiküszöbölték, használatuk a tudományos jellegü feladatoknál talán célszerübb lenne, mégsem használják őket olyan mértékben, mint a FORTRAN-t. Ennek az okait a következőkben foglalhatjuk össze:

A software termékek cseréjénél fontos szempont, hogy az alkalmazott programozási nyelv eléggé szabványositott legyen ahhoz, hogy különböző gépeken könnyen megvalósitható legyen. Ennek a követelménynek a ma használt magasszintü programozási nyelvek közül a FORTRAN tesz leginkább eleget.

Hasonlóképpen a FORTRAN mellett szól az is, hogy a nyelv jól támogatott. A tudományos szubrutincsomagok és egyéb közhasznu software termékek nagy része iródott FORTRAN-ban, igy ezek FORTRAN programokból elérhetők. A FORTRAN nyelv szivós továbbéléséhez hozzájárul egy bizonyos tehetetlenség is. A tudományos célu programokat nagyrészt maguk a felhasználók irják, akik nem szivesen térnek át az egyszer már megtanult FORTRAN-ról más nyelvre. Ugyancsak fontos számukra az is, hogy az ujonnan kidolgozott programok az eddig már meglevő programokkal kompatibilisak legyenek; a régi programok moduljait be lehessen épiteni uj programokba, a programok eredményeit más programok be tudják olvasni, stb.

A FORTRAN ellen szól viszont az, hogy a mai software termékektől alkalmazási területektől függetlenül elvárjuk azt, hogy a strukturált programozás követelményeinek eleget tegyenek.

A program kódolásának megkönnyitésére, az utóbbi években egyre jobban terjednek a FORTRAN nyelv bővitett változatai. Ezek olyan ciklusképzési és vezérlési utasitások használatát engedik meg, melyek a FORTRAN nyelvben nem állnak rendelkezésre.

Ezek a módszerek lehetővé teszik /kihasználva a FORTRAN nyelv által nyujtott moduláris programozási technikát is/, hogy olyan programokat irjunk, melyek a fenti követelményeket nagyrészt, vagy teljesen kielégitik, ugyanakkor a bővitett FORTRAN nyelvü programok könnyen összekapcsolhatók FORTRAN modulokkal.

A FORTRAN programok használatát az input/output kötöttségek lazitása is megkönnyiti. Erre többféle próbálkozást ismerünk. Egyes FORTRAN gépi reprezentációk tartalmaznak kötetlen formátumu input/output utasitásokat. Ilyen például az IBM FORTRAN "NAMELIST" utasitása. Egy másik lehetőség: FORTRAN nyelvü szubrutinokkal megoldani a kötetlen formátumu adatbeolvasást. A bővitett FORTRAN változatok, sajnos, nem tartalmaznak kötetlen formátumu input/output utasitásokat.

A következőkben a FORTRAN nyelv két bővitett változatát, a MORTRAN [4] /MACRO FORTRAN/ és a RATFOR [3,5,6] /Rational FORTRAN/ programozási nyelveket mutatjuk be részletesebben. Ezeken a nyelveken irt programok a KFKI Számitóközpont R-40-es számitógépén is futtathatók.

A MORTRAN és a RATFOR programok feldolgozása annyiban tér el a FORTRAN programokétól, hogy ezeket előbb egy /FORTRAN nyelven irt/ előfeldolgozó programmal /preprocesszorral/ le kell forditani FORTRAN nyelvre. A feldolgozás további menete azonos a FORTRAN programokéval.

A nyelvek ismertetésében nem törekszünk teljességre, mivel mindkét nyelvről bővebb leirás is rendelkezésünkre áll [4], [5].

- A cimke betük és számok tetszőleges sorozata.
- Cimkét minden utasitás elé irhatunk a programban, és kettőspontok /:/ közé kell tenni, pl: :LABEL:.
- Szóköz karaktereket a cimkék, a karaktersorozatok és a felhasználók által definiált makrók tetszőlegesen tartalmazhatnak.
- A preprocesszor a saját cimkéit a 10000-től 10999-ig terjedő intervallumban generálja; a felhasználó csak ennél kisebb egész számot használhat cimkeként.

1.2 A nyelv szerkezete

1.2.1 Utasitás: A FORTRAN nyelv ugy tekinthető, mint a MORTRAN egy részhalmaza. Igy egy érvényes FORTRAN utasitás egyben érvényes MORTRAN utasitás is, ha az utasitásokat a következőképpen módositjuk:

a/ az utasitást pontosvessző zárja le,
b/ a folytatósort jelző karaktert /ha van/ elhagyjuk,
c/ a cimkét /ha van/ kettőspontok közé tesszük.

1.2.2 Blokk: Egy MORTRAN blokk utasitások sorozata két speciális karakter, a "<" és ">" jelek közé zárva, pl:

S1;S2;S3; ... SN; utasitások sorozata.
<S1;S2;S3...SN> utasitások sorozatát tartalmazó blokk.
A blokkok egymásba skatulyázhatók.
<S1; S2; S3;.....<T1; T2;.....TM;> ....SN;>

1.3 Feltételes utasitások

1.3.1 IF utasitás

Az utasitás alakja:

IF E <...>

ahol E egy logikai kifejezés, <...> pedig egy blokk. Ha E igaz, akkor a blokk végrehajtódik, és a feldolgozás folytatódik a blokkot követő utasitáson.

```
IF P <A>
ELSEIF Q <B>
ELSEIF R <C>
ELSEIF S <D>
ELSE     <E>
```

Összefoglalva: Egy IF utasitást követhet néhány ELSEIF, melyeket befejezésként egy ELSE kell, hogy kövessen.

1.4 Ismétlődő utasitások

1.4.1 A _WHILE_ utasitás

Az utasitás alakja:

```
WHILE E <...>
```

Először az E logikai kifejezés értékelődik ki. Ha E igaz, akkor a blokk végrehajtódik és a végrehajtás visszatér az E kifejezés vizsgálatához. Ha E hamis, akkor a végrehajtás a blokkot követő első utasitásra tér át.

Például:

```
WHILE I.LT.100   <I=I+1; CALL ABC(I)>;
```

1.4.2 LOOP_ utasitás

Az utasitás alakja:

```
LOOP <...> WHILE E;
```

Ebben az esetben a blokk utasitásai hajtódnak végre először, és ez után értékelődik ki az E logikai kifejezés. Ha E igaz, akkor a blokk utasitásai ismét végrehajtódnak. Ha E hamis, akkor a végrehajtás a ciklust követő utasitásra tér át.

Például:

```
LOOP <
     A=A*2.0;
     B=A/B >
WHILE B.LT.C;
```

ahol I,J,K,N egész változó kell, hogy legyen és nem lehet tömbelem vagy kifejezés. A DO utasitásnak ez az alakja megfelel a FORTRAN DO utasitásnak. Egy standard FORTRAN ciklusutasitást jelent az

<I = J,K,N;....> ciklusutasitás is.

Egymásba skatulyázott ciklus:

<I=1,N1; <J=1,N2; <K=1,NS; A(I,J,K)= Kif. >>>

Például:

```
DO  I=1,N
      <X=FUN(I);
       Z=Z+B*X> ;
```

1.4.6 Végtelen_ciklus

A MORTRAN-ban van végtelen ciklus szervezési lehetőség is. Alakja:

```
      LOOP <....>  REPEAT
vagy  LOOP <....>
```

A REPEAT csak arra szolgál, hogy vizuálisan lezárja a ciklust. Ez különösen akkor hasznos, ha az több egymásba skatulyázott blokkot tartalmaz. Kérdés, hogy hogyan ugorjunk ki a ciklusból?

GO TO :LABEL:; ahol a cimke kivül esik a cikluson.

Használhatjuk a MORTRAN EXIT utasitást is a ciklus leállitására:

```
EXIT;     vagy    IF  E  EXIT;
                  IF  E  <.....EXIT;>
```

A végrehajtás az EXIT hatására a ciklust követő utasitásra tér át. Használhatjuk a ciklusutasitáson belül a NEXT utasitást, amely visszaugrat a ciklus elejére:

```
IF  E  NEXT;
IF  E  <....NEXT;>
NEXT   :LABEL:;
```

hajtását. Példánkban az első értékadás megváltoztatja az I változó értéket. A következőkben A(I,J) indexe a már módositott I változó lesz.

1.6 Makrók használata

A MORTRAN nyelv lehetőséget ad makrók használatára. Használhatunk standard MORTRAN makrókat, és definiálhatunk felhasználói makrókat is.

Egy makródefinició a következő alaku:

```
%'MINTA'='HELYETTESITÉS'
```

A makró-definició nem utasitás, igy nem kell pontosvesszővel lezárni. A 'MINTA' és a 'HELYETTESITÉS' karaktersorozatok, aposztrófok között. A 'MINTA' és a 'HELYETTESITÉS' is tartalmazhat pontosvesszőt. Az első példa a legegyszerübb makródefiniciót szemlélteti:

```
%'ARRAYSIZE'='50'
```

A MORTRAN programban bárhol előforduló "ARRAYSIZE" karaktersorozat helyére a preprocesszor az "50" karaktereket helyettesiti. Pl: A MORTRAN programban előforduló

```
DIMENSION A(ARRAYSIZE);
```

sornak a FORTRAN programban

```
DIMENSION A(50)
```

fog megfelelni.

A következő példában egy MORTRAN standard makrót is használunk:

```
OUTPUT Z; (F10.2);
```

Az OUTPUT Z; (F10.2); egy standard MORTRAN makrónak a hivása, amely helyett a preprocesszor a következő FORTRAN utasitásokat fogja generálni:

```
      WRITE(6,XXX)Z
XXX FORMAT (F10.2)
```

- Ismétlődő programrészeknek /pl. állandó deklarációs részeknek/ a program több helyére való bemásoltatására.

Mik a makrók használatának veszélyei, mire kell vigyázni?

Mivel az aktuális paraméter valóban "bármi" lehet, az esetleges hibás paraméterek behelyettesitése szintaktikusan hibás FORTRAN utasitást is szolgáltathat. Ez azonban egyes esetekben csak a FORTRAN fordításkor derül ki, amikor már nehezebb visszakeresni a MORTRAN forrásprogramban elkövetett eredeti hibát.

1.7 Vezérlő utasitások

A MORTRAN forrásprogramon belül elhelyezhetünk vezérlő utasitásokat. Ezek a vezérlő utasitások a preprocesszornak szólnak. A vezérlő utasitások formája kötött. A sor /kártya/ első karakter-poziciójára el kell helyezni egy % jelet. Ezt követi egy vezérlő karakter, melyet követhet még egy egész szám.

A legfontosabb vezérlő utasitások:

%% A MORTRAN forrásprogram végét jelzi. Ennek a vezérlő utasitásnak a használata kötelező.
A további vezérlő utasitások használata tetszés szerinti.

%F Átkapcsolás FORTRAN módra. A vezérlő utasitást követő programsorokat a preprocesszor beolvassa és kiirja változtatás nélkül. Az alapértelmezés MORTRAN mód.

%M Visszakapcsolás MORTRAN módra.

%E Uj lap kezdése a MORTRAN listán.

%L List. Kérjük a MORTRAN listát. Ez az alapértelmezés.

%N Nolist. Nem kérjük a MORTRAN listát.

%Un Unit Switch. /Egység kapcsoló/,
ahol n egy FORTRAN perifériaszám, pl. %U15.
A MORTRAN forrásprogramot több perifériáról is olvashatjuk. Ha a MORTRAN forrásprogram olvasása közben a

1.9 Korlátozások, figyelmeztetések

A MORTRAN nyelvü program irásánál legyünk tekintettel a következő formai kötöttségekre:

- A különböző terminálo és elhatároló jelek használata kötelező utasitások lezárására, karaktersorozatok, megjegyzések és cimkék jelölésére. Ezek a formai megkötések a FORTRAN nyelvben nincsenek meg, és igy talán először szokatlannak tünnek.

- Nem használhatjuk a 10000 és 10999 közé eső cimkéket, ez a tartomány a MORTRAN preprocesszor által generált cimkék számára van fenntartva. A cimkék viszont tartalmazhatnak betüket is.

A formai kötöttségeken kivül figyelembe kell venni azt, hogy mivel a MORTRAN nyelven irt programot először a preprocesszor leforditja FORTRAN-ra - melynek során nem végez szintaktikus vizsgálatot - kaphatunk szintaktikusan hibás FORTRAN programot is. Ilyen esetben a hiba megkeresése a látszatra hibátlan MORTRAN programban kényelmetlen lehet.

2. RATFOR

A RATFOR nyelv definicióját Kernighan adta meg 1975-ben [6]. A nyelvet néhány, RATFOR-ban megirt, a gyakorlatban jól használható programrendszerrel együtt ismerteti Kernighan és Plauger 1976-ban megjelent Software Tools cimü könyvében [5], melynek magyar forditása a Müszaki Könyvkiadó gondozásában megjelenés alatt áll.

2.1 Kódolási szabályok

A RATFOR program kódolásánál a következő szabályokat kell figyelembe venni:

- Egy utasitás bárhol kezdődhet az input sorban /kártyán/, és befejezhető ugyanabban a sorban vagy folytatható több soron keresztül is.
- Egy sorba több utasitás is irható, ilyenkor azokat pontosvesszővel kell egymástól elválasztani.
- A cimkézett utasitásokban a cimkének közvetlenül az utasitás előtt kell állnia, attól legalább egy betüköz karakterrel elválasztva.
- Kommentár bárhol elhelyezhető a programban és # jellel kell kezdődnie. A kommentár a # jeltől a sor végéig tart.
- A RATFOR sorfolytatási szabályai eltérnek a FORTRAN-étól. Minden olyan sor amely vesszővel /,/ vagy & jellel, vagy ! jellel végződik, befejezetlen utasitást jelöl. Mivel a befejezetlen utasitások általában vesszőket tartalmazó listákból állnak /pl: FORMAT/, vagy hosszu logikai kifejezések, ezért az automatikus folytatás elegendő. Aritmetikai kifejezések másik sorban történő folytatására a RATFOR-ban nincs lehetőség.
- Megjegyzést tartalmazó sor folytatására is van lehetőség, mivel a RATFOR figyelmen kivül hagyja a sornak a # utáni részét.
- Ha egy sorban csak egy cimke van, akkor a preprocesszor a cimke után egy CONTINUE utasitást ir.

nak karakterkészletéhez alkalmazkodva cseréltük ki a fenti karaktereket.

Az utasitások, illetve az utasitáscsoportok 50-es mélységig egymásba skatulyázhatók.

2.3 Feltételes utasitások

2.3.1 Az IF utasitás

Az utasitás alakja:

```
IF(E)
   utasitás1
ELSE
   utasitás2
```

Ha az E feltétel igaz, akkor az "utasitás1", különben az "utasitás2" hajtódik végre. Az ELSE rész elhagyható.

Az IF utasitások egymásba skatulyázhatók. Ilyenkor azonban vigyázni kell arra, hogy az utasitás egyértelmü legyen. Pl: az

```
IF (E1)
   IF (E2)
      utasitás1
   ELSE
      utasitás2
```

szerkezet nem egyértelmü; ajánlatos ilyen esetekben az utasitás-zárójelek használata, például:

```
IF (E1)[
     IF(E2)
          utasitás1 ]
     ELSE utasitás2
```

További példák az IF utasitás használatára:

```
IF (A.LT.B)
   CALL XYZ(I,J,K)
IF (A.GT.B)[
   CALL ZW(A,B)
   F1= .TRUE.
   ]
IF (I.LE.J)
   MIN=3
ELSE
[MIN=J; F1= .TRUE.]
```

esetben először az "utasitás", majd a "léptetés" kerül sorra. Ezután a "feltétel" kifejezés kiértékelése következik, és a folyamat ismétlődik. A ciklus végrehajtása addig tart, amig a "feltétel" hamis értékü nem lesz.

A preprocesszor a következő FORTRAN utasitásokat generálja:

```
      kezdőérték-beállitás
L1    IF(.NOT. feltétel) GOTO L3
      "A RATFOR utasitásból generált FORTRAN
      utasitások"
L2    léptetés
      GOTO L1
L3    CONTINUE
```

ahol 'L1','L2' és 'L3' a preprocesszor által generált cimke:

23000<=L1,L2,L3<=23999

A FOR utasitás egyenértékü a következő WHILE utasitással:

```
kezdőérték-beállitás
WHILE (feltétel){
utasitás
léptetés}
```

Ha akár a "kezdőérték-beállitás", akár a "léptetés" hiányzik, a kifejezés megfelelő tagja elhagyható. Ha a "feltétel" hiányzik, akkor ez végtelen ciklust eredményez.

A FOR utasitás legegyszerübb formájában egy FORTRAN DO utasitással egyenértékü. Összetett logikai kifejezésekkel azonban bonyolult funkciókat is megvalósithatunk. A FOR utasitás alkalmas olyan ciklusképzésre is, amelynél nem aritmetikai feltételeket kell figyelembe venni. Példák:

```
FOR (I=1; I.LE.N; I=I+1) SUM=SUM+C(I)
FOR (I=1; I.LE.N.AND. C(I).NE.0; I=I+1)
     V(I)=1/C(I)
FOR (;J.GT.0; J=J-1) CALL ABC(J)
```

2.4.3 A REPEAT-UNTIL utasitás

Az utasitás alakja:

```
REPEAT
    utasitás
UNTIL (feltétel)
```

Példa:

```
DO I=1, N [
X=Y(I)
Z=Z+A*X+1.0/X**2
]
```

2.4.5 A BREAK utasitás

Az utasitás alakja:

BREAK

A BREAK utasitás lehetőséget nyujt egy ciklusból /például végtelen ciklusból/ való kilépésre. Ha egy ciklusban a vezérlés egy BREAK utasitásra kerül, akkor a végrehajtás a ciklust követő utasitásokon folytatódik.

Csak egyetlen ciklus fejeződik be egy BREAK utasitással, akkor is, ha a BREAK többszörösen egymásba skatulyázott ciklusok belsejében van elhelyezve. Például:

```
REPEAT [
    CALL ABC(X)
    IF(X.LT.100.0) BREAK ]
```

2.4.6 A NEXT utasitás

Az utasitás formája:

NEXT

Hatására a NEXT utasitást tartalmazó ciklus azonnal áttér a következő ciklislépésre, kihagyva a ciklustörzs hátralevő részét. A WHILE vagy az UNTIL esetén a vezérlés a "feltétel"--részre kerül. A DO és a REPEAT esetén a ciklus elejére, FOR esetén a "léptetés"-re. Például:

```
WHILE (X.GT.Y) [
        CALL  A(X)
        IF(X.GT.Z) NEXT
        I=I+1 ]
```

```
      :
SUBROUTINE XYZ
COMMON /ADAT/ A,B,C
      :
```

2.7 A RATFOR hibaüzenetei:

- CAN'T OPEN INCLUDE
- DEFINITION TOO LONG
- ERROR AT LINE n
- ILLEGAL BREAK
- ILLEGAL NEXT
- INVALID FOR CLAUSE
- INCLUDE NESTED TOO DEEPLY
- ILLEGAL ELSE
- ILLEGAL RIGTH BRACE
- MISSING LEFT PAREN
- MISSING PARENTHESIS IN CONDITION
- MISSING COMMA IN DEFINE
- MISSING RIGTH PAREN
- MISSING QUOTE
- NON-ALPHANUMERIC NAME
- STACK OVERFLOW IN PARSER
- TOKEN TOO LONG
- TOO MANY DEFINITIONS
- TOO MANY CHARACTERS PUSHED BACK
- UNEXPECTED EOF
- UNEXPECTED BRACE OR EOF
- UNBALANCED PARENTHESIS
- WARNING: POSSIBLE LABEL CONFLICT

2.8 A RATFOR Job Control eljárások ismertetése:

A RATFOR nyelv használatának megkönnyitése érdekében összeállitottunk néhány Job Control eljárást. Ezen eljárások a KFKI Számitóközpont SOFT.PROCLIB eljáráskönyvtárában találhatók.

- RATFORC : előforditás+FORTRAN forditás
- RATFORCL : előforditás+FORTRAN forditás+szerkesztés
- RATFORCG : előforditás+FORTRAN forditás+szerkesztés+ +végrehajtás

2.10 Egy RATFOR program

```
#  TEST PROGRAM USING RESERVED KEYWORDS

 DEFINE(ARRAYSIZE,100)
 DEFINE(PRIMESIZE,3000)
 DEFINE(OUTPUT,6)

INTEGER SIEVE1
INTEGER PRIMES(PRIMESIZE)
INTEGER COUNT

COUNT=SIEVE1(10000, PRIMES)
WRITE(OUTPUT,100)COUNT
WRITE(OUTPUT,101)(PRIMES(I),I=1,COUNT)
100 FORMAT(I8,' PRIMES LESS THAN 10,000'//)
101 FORMAT(1X,10I10)
STOP
END

INTEGER FUNCTION SIEVE1(MAX, PRIMES)
    #  GENERATE PRIMES
    #    USES ALGORITHM 310(A1), PRIME GENERATOR 1,
    #    B. A. CHARTRES, CACM 10(SEPT 1967), 569
INTEGER MAX; INTEGER PRIMES(PRIMESIZE)
INTEGER Q(ARRAYSIZE),DQ(ARRAYSIZE)
INTEGER I,SVSZ,PRMCNT,N
LOGICAL PRIME
    #  INITIALIZE
PRIMES(1)=2; SVSZ=2; PRMCNT=2; PRIMES(2)=3; Q(2)=9; DQ(2)=6;

    #  NOW LOOK FOR PRIMES

FOR (N=5; N <= MAX; N=N+2) {
      PRIME=.TRUE.
      FOR (I=2; I <= SVSZ; I=I+1)  {
                              #  RUN THE NUMBER N THROUGH THE SIEVE
          IF ( N == Q(I)) {        # CAUGHT BY THE SIEVE?
               PRIME=.FALSE.
               Q(I)=N+DQ(I)
               IF (I == SVSZ) {
                    SVSZ=SVSZ+1; Q(SVSZ)=PRIMES(SVSZ)**2
                    DQ(SVSZ)=2*PRIMES(SVSZ)
                    BREAK
                    }
               }
          }
      IF (PRIME) { PRMCNT=PRMCNT+1; PRIMES(PRMCNT)=N  }
      }
SIEVE1=PRMCNT
RETURN; END
```

és szemantikus elemzést. Emiatt előfordulhat, hogy a preprocesszor a behelyettesitás elvégzése után szintaktikusan hibás FORTRAN programot állit elő, továbbá, hogy a hiba a generált FORTRAN programban egészen máshol jelentkezik, mint ahol az eredeti RATFOR, vagy MORTRAN programban azt elkövettük.

- A fentemlitett problémák ellenére bizonyos feladatoknál célszerü a FORTRAN helyett a RATFOR-t, illetve a MORTRAN-t választani programozási nyelvként, mivel a kódolás ezekben a nyelvekben lényegesen egyszerübb, a létrehozott program pedig sokkal áttekinthetőbb lesz.

Mindkét nyelv használható a KFKI Számitóközpont R-40-es számitógépen, és a RATFOR-nak van kisgépes változata /PDP-11-es/ is. Ennek részletes leirását l. [3]. A MORTRAN nyelv részletes leirását tartalmazó User's guide a KFKI Programkönyvtárban található meg [4].

Irodalomjegyzék

[1] Dahl-Dijkstra-Hoare: Strukturált programozás. Müszaki Könyvkiadó /1978/.

[2] Aszalós János - Erki Irén: Bevezetés a strukturált programozásba. SZÁMOK kiadás, 1980.

[3] Kovács Kálmán: FLECS és RATFOR, FORTRAN-ra alapozott programozási nyelv. KFKI, 1980.

[4] A.J.Cook - L.J.Shustek: A user's guide to MORTRAN2. Computer Research Group, Stanford Linear Accelerator Center; Stanford, California, /1975/.

[5] B.W.Kernighan - P.J.Plauger: Software Tools. Addison-Wesley Publishing Co., New York, /1976/.

[6] B.W.Kernighan: RATFOR-A preprocessor for a rational FORTRAN, Software Practice and Experience, 5 /1975/, 395-406.

3. Összefoglalás

A két nyelv rövid ismertetése után összefoglalva a következőket mondhatjuk:

- A RATFOR és a MORTRAN nyelv eleget tesz azoknak a követelményeknek, melyeket a strukturált programozás támaszt egy programozási nyelvvel szemben. Mindkét nyelv olyan utasitásokkal van kiegészitve a FORTRAN-hoz képest, melyek kiküszöbölik a FORTRAN gyengeségeit. Azonban az input/output utasitások terén nem jelentenek ujat.

- Formailag a RATFOR közelebb áll a FORTRAN-hoz, mint a MORTRAN. A standard FORTRAN kommentárjai, sorfolytatás konvenciói, valamint az aritmetikai IF és a DO utasitás nem kompatibilis a RATFOR-ral. Minden más FORTRAN utasitás egyben RATFOR utasitás is.

- A MORTRAN viszont azon kivül, hogy bővitéseket tartalmaz a FORTRAN-hoz képest, formailag is eltér attól. Ezért a FORTRAN utasitások általában nem szintaktikusan helyes MORTRAN utasitások. Ha egy FORTRAN programot MORTRAN nyelvre akarunk átirni, jelentősen át kell alakitanunk. Mód van viszont arra, hogy MORTRAN és FORTRAN modulokból, modul részekből állitsuk össze a programunkat, de ilyenkor használni kell a MORTRAN vezérlő paramétereit.

- A MORTRAN-ban és RATFOR-ban irt programokban számolni kell azzal, hogy a forditás idejéhez hozzáadódik még a preprocesszor futási ideje. A MORTRAN és a RATFOR preprocesszorok a programot FORTRAN-ra lényegesen hosszabb idő alatt forditják le, mint a FORTRAN fordító futási ideje ugyanazon program forditásánál. Ez az időarány MORTRAN-nal körülbelül háromszoros, RATFOR-nál megközelitően ötszörös faktort jelent.

- Némi kényelmetlenséget okoz a hibakeresés is. Ez a kényelmetlenség a preprocesszor alkalmazásából következik. A preprocesszor egy szöveghelyettesitő program, makróprocesszor, és nem forditóprogram, és igy nem végez részletes szintaktikus

Példa RATFOR eljárás hivására:

```
// EXEC RATFORCG,INPUT='QWER.RATFPROG',PACK='xxxxxx',LIST=
```

A RATFOR forrásprogramot a "QWER.RATFPROG" nevü file tartalmazza, mely a "xxxxxx" lemezen található. A procedurák használata során mindenképpen kapunk RATFOR forrásnyelvü programlistát. Alapértelmezés szerint a preprocesszor által generált FORTRAN kódról nem adnak listát az eljárások. Amennyiben mégis szükségünk van a FORTRAN listára, akkor hiváskor a paraméterlistán felül kell birálni a 'LIST=DUMMY' alapértelmezést a 'LIST=' paraméter megadásával.

2.9 Korlátozások, figyelmeztetések

A RATFOR nyelvü program irásánál legyünk tekintettel a következőkre:

- Néhány FORTRAN utasitás, mint az aritmetikai IF és a DO nem használható RATFOR programban, de az összes többi FORTRAN utasitás egyben RATFOR utasitás is.

- Van néhány formai szabály, melyekre tekintettel kell lenni: Megjegyzések #-al történő jelölése; cimkéknek közvetlenül az utasitás előtt kell lenniük; egy sorba több utasitás is irható, de azokat pontosvesszővel el kell választani, stb.

- A cimkéket 23000 és 23999 között nem lehet használni. Ez a tartomány a RATFOR preprocesszor által generált cimkék számára van fenntartva.

- A RATFOR megengedi ugyan a sorfolytatást, de az aritmetikai utasitásokat nem folytathatjuk.

A formai kötöttségeken tul figyelembe kell venni azt, hogy mivel a RATFOR nyelven irt programot először a preprocesszor leforditja FORTRAN-ra - miközben nem végez szintaktikus ellenőrzést - kaphatunk szintaktikusan hibás FORTRAN programot is. Ilyen esetben a hiba megkeresése a látszólag hibátlan RATFOR programban kényelmetlen lehet.

2.5 A DEFINE utasitás

Az utasitás formája:

DEFINE (név, karakterek)

A "név" tetszőleges, alfanumerikus karakterekből álló szimbólum. A programban előforduló "név" helyére a preprocesszor a "karaktereket" helyettesiti. Példák:

```
DEFINE (MAXI,100)
DEFINE (BLANK,' ')
DEFINE (CHAR,LOGICAL*1)
```

A DEFINE utasitásnak egy gyakorlati felhasználását a következő pontban mutatjuk be.

2.6 Az INCLUDE utasitás

Az utasitás formája:

INCLUDE filenév

Hatására a megadott file tartalma az INCLUDE utasitás előfordulásának helyén behelyettesitődik a forrásnyelvü programba. A "filenév" tetszőleges alfanumerikus karaktersorozat lehet, de a program elején egy DEFINE utasitásban ezt a nevet egy logikai periféria azonositó számmá kell alakitani. A preprocesszor által generált FORTRAN beolvasó utasitások erre a számra hivatkoznak.

Az INCLUDE egyik gyakori alkalmazása a COMMON deklarációk beillesztése. Ezzel elérjük azt, hogy minden programrészben azonosak legyenek a közös adatterületek. Például:

```
DEFINE (DATA,12)
  :
SUBROUTINE XYZ
INCLUDE DATA
  :
```

ahol a 12-es logikai periféria azonositó által meghatározott DD kártyán definiált file egy COMMON utasitást tartalmaz. Az előfeldolgozás után a preprocesszor a következő FORTRAN kódot generálja:

Az "utasitás" egyszer végrehajtódik. Ezután kiértékelődik a "feltétel", és ha ez hamis, az "utasitás" ujból végrehajtódik. A ciklus akkor áll le, ha a "feltétel" igazzá válik. Az UNTIL rész elhagyható. Ha a "feltétel" hiányzik, az eredmény egy végtelen ciklus, amelyből valahogy másképp kell kilépni, lásd a BREAK utasitást /2.4.6 alpont/.

A preprocesszor a következő FORTRAN utasitásokat generálja:

```
L CONTINUE
  " A RATFOR utasitásból generált FORTRAN
  utasitás"
  IF (.NOT. feltétel) GOTO L
  CONTINUE
```

ahol 'L' a preprocesszor által generált cimke:

23000 <= L <= 23999

Példa:

```
REPEAT
CALL XYZ(A,B,C)
UNTIL (A.LT.B)
```

2.4.4 A DO utasitás

Az utasitás alakja:

```
DO határok
   utasitás
```

A "határok" a szokásos FORTRAN DO utasitásban szereplő specifikációt jelentik. Ebben a DO utasitásban azonban nem szerepel cimke /a preprocesszor generálja a DO utasitáshoz a cimkét/. Minden DO utasitás helyettesithető FOR utasitással, de forditva nem.

A preprocesszor a következő FORTRAN utasitásokat generálja:

```
  DO L határok
  " A RATFOR utasitásból generált FORTRAN
  utasitások"
L CONTINUE
```

ahol 'L' a preprocesszor által generált cimke:

23000 <= L <= 23999

Ha kettőnél több eset szétválasztása a feladat, akkor használhatunk IF-ELSE láncot. Például:

```
IF (CH.EQ.A1)
   N1=+1
ELSE IF (CH.EQ.A2)
   N1=-1
ELSE IF (CH.EQ.A3)
   N1=0
ELSE CALL ERROR
```

2.4 Ismétlődő utasitások

2.4.1 WHILE utasitás

Az utasitás alakja:

```
WHILE (E)
   utasitás
```

Először az E logikai kifejezés értékelődik ki. Ha E igaz, akkor az "utasitás" végrehajtódik, és a vezérlés visszatér az E kifejezés vizsgálatához. Ha E hamis, akkor az "utasitás" nem hajtódik végre.

A preprocesszor a következő FORTRAN utasitásokat generálja:

```
L1  IF(.NOT.E) GOTO L2
    " A RATFOR utasitásból generált FORTRAN
    utasitások "
    GOTO L1
L2  CONTINUE
```

ahol 'L1' és 'L2' a preprocesszor által generált cimke:

23000 <= L1,L2 <= 23999

```
P1: WHILE (A(I).NE.0.AND.I.LE.M)
    I=I+1
```

2.4.2 A FOR utasitás

Az utasitás alakja:

```
FOR (kezdőérték-beállitás; feltétel; léptetés)
    utasitás
```

A "kezdőérték-beállitás" és a "léptetés" tetszőleges FORTRAN utasitások lehetnek. Először a "kezdőérték-beállitás" történik meg. Ha ezután a "feltétel" hamis értékü, akkor az "utasitás" és a "léptetés" nem kerül végrehajtásra. Ellenkező

- A cimke számok tetszőleges sorozata /de legfeljebb 6 karakter hosszu lehet/.

- Karakter-sorozatokat aposztrofok /'/ közé kell tenni.

- A RATFOR-ban a deklarációs lehetőségek megegyeznek a FORTRAN-ban használtakkal, kiegészitve azzal, hogy van CHARACTER tipus is, amit azonban a preprocesszor INTEGER tipusra fordit.

- A preprocesszor saját cimkéit a 23000-től 23999-ig terjedő intervallumban generálja, tehát a felhasználó ezen a tartományon kivül eső egész számot használhat cimkeként.

- A RATFOR nyelvben a FORTRAN logikai és relációs müveleti jelek alábbi röviditett formáit is lehet használni:

FORTRAN	RATFOR
.LT.	<
.LE.	<= vagy =<
.EQ.	==
.GE.	>= vagy =>
.GT.	>
.NE.	<> vagy >< vagy ¬=
.AND.	&
.OR.	!
.NOT.	¬

2.2 A nyelv szerkezete

Utasitás: egy érvényes FORTRAN utasitás egyben egy érvényes RATFOR utasitás is. Kivéve:

- a DO utasitást, melynek alakja eltér a FORTRAN DO utasitástól;
- a FORTRAN aritmetikai IF utasitás hiányzik a RATFOR nyelv utasitáskészletéből.

Utasitáscsoportok: Az utasitások csoportositathatók. Az utasitáscsoportot szögletes zárójelek [ ] közé kell tenni. Az eredeti RATFOR preprocesszor a szögletes zárójelek helyett kapcsos { } zárójeleket használt. Az ESZR gépek perifériái-

1.10 Egy MORTRAN program

```
"  TEST PROGRAM USING RESERVED KEYWORDS "

 % !ARRAYSIZE! = !100!
 % !PRIMESIZE! = !3000!

INTEGER SIEVE1;
INTEGER PRIMES(PRIMESIZE);
INTEGER COUNT;

COUNT=SIEVE1(10000, PRIMES);
OUTPUT COUNT;(I8,' PRIMES LESS THAN 10,000'//);
OUTPUT (PRIMES(I),I=1,COUNT);(1X,10I10);
STOP;
END;

INTEGER FUNCTION SIEVE1(MAX, PRIMES);

    "  GENERATE PRIMES
       USES ALGORITHM 310(A1), PRIME GENERATOR 1,
       B. A. CHARTRES, CACM 10(SEPT 1967), 569
    "
INTEGER MAX; INTEGER PRIMES(PRIMESIZE);
INTEGER Q(ARRAYSIZE),DQ(ARRAYSIZE);
INTEGER I,SVSZ,PRMCNT,N;
LOGICAL PRIME;

    "  INITIALIZE "

/PRIMES(1),SVSZ,PRMCNT/=2; PRIMES(2)=3; Q(2)=9; DQ(2)=6;

    "  NOW LOOK FOR PRIMES "

:PRIME TEST:
FOR N=5 TO MAX BY 2 <
    PRIME=.TRUE.;
    FOR I=2 TO SVSZ <       " RUN THE NUMBER N THROUGH THE SIEVE"
        IF N .EQ. Q(I) <        " CAUGHT BY THE SIEVE? "
            PRIME=.FALSE.;
            Q(I)=N+DQ(I);
            IF I .EQ. SVSZ <
                SVSZ=SVSZ+1; Q(SVSZ)=PRIMES(SVSZ)**2;
                DQ(SVSZ)=2*PRIMES(SVSZ);
                NEXT :PRIME TEST:;
                >
            >
        >
    IF (PRIME) < PRMCNT=PRMCNT+1; PRIMES(PRMCNT)=N; >
    >
SIEVE1=PRMCNT;
RETURN; END;
%%
```

preprocesszor talál egy %Un vezérlő utasitást, akkor az n perifériaszámu perifériáról fog tovább olvasni, majd a %% jel után visszatér az eredeti periférára. Ha a %% vezérlőutasitás előtt egy másik %Un utasitást talál, akkor ujabb periféria átkapcsolás történik. Ez a vezérlő utasitás helyettesiti a más nyelvekben pl. a RATFOR-ban is előforduló "INCLUDE", vagy "COPY" utasitást.

A további vezérlő utasitásokat lásd a MORTRAN USER'S GUIDE-ban [4].

1.8 A MORTRAN Job Control eljárások ismertetése

A MORTRAN nyelv használatának megkönnyitése érdekében összeállitottunk néhány Job Control eljárást. Ezen eljárások a KFKI Számitóközpont SOFT.PROCLIB eljáráskönyvtárában találhatók.

- MORTC : előforditás+FORTRAN forditás.
- MORTCL : előforditás+FORTRAN forditás+szerkesztés.
- MORTCLG : előforditás+FORTRAN forditás+szerkesztés+ +végrehajtás.
- MORTMANU: kézikönyv kinyomtatása.

Példa MORTRAN eljárás hivására:

```
// EXEC MORTCLG,INPUT='QWER.MORTPROG',PACK='xxxxxx',LIST=
```

A MORTRAN forrásprogramot a "QWER.MORTPROG" nevü file tartalmazza, mely a "xxxxxx" lemezen található. A procedurák használata során MORTRAN forrásnyelvü programlistát mindenképpen kapunk. Alapértelmezés szerint a preprocesszor által generált FORTRAN kódról az eljárások nem adnak listát. Amennyiben mégis szükségünk van a FORTRAN listára, akkor hiváskor a paraméterlistán felül kell birálni a 'LIST=DUMMY' alapértelmezést a 'LIST=' paraméter megadásával.

A MORTRAN kézikönyv kinyomtatása a következőképpen történik:

```
// EXEC MORTMANU
```

ahol XXX egy, a MORTRAN preprocesszor által generált cimke. Definiáljuk a következő makrót:

```
%'DUMP Z; '='OUTPUT Z;(F10.2);'
```

Ennek a makrónak a hivása:

```
DUMP Z;
```

Lehetőség van paraméterezett makrók definiálására is. Egy makrónak legfeljebb 9 paramétere lehet. A makrók definiálásánál a mintában a paramétereket #-jellel kell jelölni, melyeknek pozicionális jelentése van.

A behelyettesitésnél a #-jelet követnie kell egy sorszámnak, ez a sorszám határozza meg, hogy melyik paramétert kell a megfelelő helyre behelyettesiteni. /A sorszám 1 és 9 közé eső pozitiv egész szám/.

Lássunk egy egyszerü példát: makró definiálása:

```
%'PLUS# ;'='#1=#1+1'
 MINTA       BEHELYETTESITÉS
```

Ennek az egyparaméteres makrónak a hivása például lehet a következő:

```
PLUS A(I);
```

ahol A(I) az aktuális paraméter.

A makró kifejtésének eredménye FORTRAN-ban:

```
A(I)=A(I)+1
```

Az igy definiált PLUS makró hivásának eredményeképpen tehát egy olyan FORTRAN utasitás generálódik, amely a paraméter értékét 1-gyel növeli.

A FORTRAN nyelven programozók számára talán szokatlan a makrók használata. Mire jó a makró?

- A nyelv további utasitásokkal való bővitésére.
- A program paraméterezésére /1. első példa, ARRAYSIZE makró: a tömb méretének rögzitését a forditás, ill. az előfeldolgozás idejére halasztjuk/.

Az EXIT és a NEXT kulcsszót követheti egy cimke. Azt a ciklust, amelyben ilyen utasitás fordul elő, cimkézett ciklusnak hivjuk. Alakja:

```
EXIT  :LABEL:;
```

Ilyenkor a végrehajtás a ciklus ezen utasitásáról átugrik a cikluson kivüli LABEL-cimkéjü utasitásra.

A NEXT :LABEL: utasitással a ciklus elejére ugrathatunk, például: ha van három egymásba skatulyázott ciklusunk:

```
:CIM: LOOP <....
    :CIM1: LOOP <NEXT:CIM1:;...
        :CIM2: LOOP <...
         NEXT :CIM2:....
                    >
          ....;
              >
         EXIT :CIM:;....;>
    :CIM3:....
```

Az EXIT utasitásnál figyelmen kivül lehet hagyni az egymásba skatulyázást /belső ciklusból ki lehet ugratni magasabb szinten álló utasitásra is, nemcsak az adott szintet közvetlenül befoglaló szintre/.

1.5 Többszörös értékadás

Például: a

```
/I,A(I,K),J/=SQRT(X/2.0);
```

MORTRAN utasitás FORTRAN-ban igy forditódik át:

```
I=SQRT(X/2.0)
A(I,K)=I
J=A(I,K)
```

A példából látható, hogy többszörös értékadásnál a programozónak ügyelnie kell az esetleges konverziók hatására, továbbá arra is, hogy az értékadások során egy változó értékének megváltozása befolyásolhatja a további értékadások végre-

1.4.3 UNTIL _utasitás

Az utasitás alakja:

UNTIL E <.....>

A logikai kifejezés értékelődik ki először. Ha E hamis, akkor a blokk utasitásai végrehajtódnak és visszatérünk az E kiértékeléséhez. Ha E igaz, akkor a vezérlés a blokkot követő utasitásra kerül.

Például:

```
UNTIL  A.LT.B <CALL XYZ(A,B)> ;
```

1.4.4 FOR _utasitás

Az utasitás alakja:

FOR V=E TO F BY G <...>

ahol V a ciklus-változó, E, F és G pedig tetszőleges aritmetikai kifejezések. V lehet valós vagy egész változó, vagy tömbelem. Az aritmetikai kifejezések lehetnek pozitivak és negativak, valamint értékük és előjelük változhat a ciklus végrehajtása során. A blokk utasitásai végrehajtódnak, ha a

G*(V-F).GT.0 feltétel teljesül.

A blokk utasitásai egyszer sem hajtódnak végre, ha a feltétel nem teljesül. A F O R utasitás két további lehetséges formája:

```
FOR V=E BY G TO F <....>
FOR V=E TO F <.....>
```

Például:

```
FOR   N=MIN TO MAX BY 2 <
          SUM=SUM+A(N); B=SUM/N > ;
```

1.4.5 DO_ utasitás

Az utasitás alakja:

DO I=J,K,N <.....>

Például: IF A.LT.B <C=D; E=F;> G=H;

Ha az IF utasitásban szereplő blokk egyetlen utasitásból áll, az utasitás tulajdonképpen megfelel a FORTRAN logikai IF utasitásnak, pl.:

```
IF   A.EQ.B   <X = Y>;
```

Az ilyen utasitást az eredeti FORTRAN irásmódnak megfelelő

```
IF  (A.EQ.B)   X = Y;
```

alakban is irhatjuk.

1.3.2 IF_..._ELSE utasitás

Az utasitás formája:

```
IF E <....>  ELSE <....>
```

Például:

```
IF A.LT.B  <....;>  ELSE <...;>  K=L;
```

Az IF-ELSE utasitások egymásba skatulyázhatók, például:

```
IF P <A>
ELSE < IF Q <B>
       ELSE < IF R <>
              ELSE < IF S<D>
                     ELSE <E>
                     >
            >
     >
```

1.3.3 ELSEIF_utasitás

Az utasitás alakja:

```
IF E1  <A1>
ELSEIF E2  <A2>
ELSE  <A3>
```

Például:

1. MORTRAN

A MORTRAN nyelvet és a MORTRAN preprocesszort A.J.Cook és L.J.Shustek dolgozta ki, ill. készitette a californiai Stanford Linear Accelerator Laboratory-ben. Ennek egy változata, a MORTRAN2 áll rendelkezésünkre [4].

A MORTRAN programozási nyelvben - hasonlóan a többi bővitett FORTRAN változathoz -, lehetőség van olyan vezérlési strukturák, valamint ciklusutasitások használatára, melyek a fejlettebb programozási nyelvekben találhatók. Ezen kivül használhatunk:

- egymásba skatulyázott blokkokat;
- többszörös értékadást;
- használhatjuk a nyelv standard makróit, definiálhatunk uj makrókat;
- különböző vezérlő karakterekkel szabályozhatjuk a program listázását; stb.

Sem a MORTRAN-ban, sem a RATFOR-ban nincs lehetőség a vektorokon és a tömbökön kivül más, magasabbrendü adatstrukturák használatára.

1.1 Kódolási szabályok

Egy program kódolásánál a következő szabályokat kell figyelembe venni:

- Egy utasitás bárhol kezdődhet az input sorban /kártyán/ és bárhol befejezhető.
- Az utasitást pontosvessző /;/ zárja le.

Ez biztositja a szabad formátumu programozás lehetőségét.

- Karakter sorozatot aposztrofok /'/ közé kell tenni. pl: 'SZÖVEG'.
- Kommentár bárhol elhelyezhető a programban, és idézőjelek /"/ közé kell tenni, pl: "COMMENT".

Melyek röviden a strukturált programozás követelményei? A program listája legyen érthető, logikailag tiszta, áttekinthető, könnyen bővithető, a program legyen könnyen kezelhető, tesztelhető, és legyen megfelelően dokumentálva.

A strukturált programozás az alkalmazott programozási nyelvvel szemben is követelményeket támaszt. Az egyik ilyen követelmény, hogy a nyelvi eszközök a kódolás során ne homályositsák el a program világos szerkezetét. Egy program követhetőségét nagymértékben befolyásolják az alkalmazott program strukturák /ciklusképző utasitások, a feltétel nélküli és feltételes vezérlésátadó utasitások, összetett utasitások/. Ezek az utasitások általában a következők:

- IF ... THEN ... ELSE
- CASE ... OF ...
- FOR ... DO
- REPEAT .. UNTIL
- WHILE ... DO
- BEGIN ... END stb.

Az olyan programozási nyelvek, melyek a strukturált programozás követelményeinek eleget tesznek, tartalmazzák a fentemlitett utasitásokat, vagy azokhoz hasonló szerkezeteket.

A FORTRAN nyelvből ezek az utasitások nagyrészt hiányoznak, a vezérlési és ciklusképző utasitások készlete szegényes. A tapasztalat azt mutatja, hogy ez, különösen a "GO TO" utasitások "fegyelmezetlen" használata nagymértékben áttekinthetetlenné teszi a program szerkezetét, megnehezíti a program tesztelését, megértését, módositását [1,2].

Hasonlóképpen nehéz kielégiteni azt a követelményt, hogy a program könnyen vezérelhető legyen. A FORTRAN kötött formátumu I/O utasitásainak használata ezt nem teszi lehetővé.

A FORTRAN programozók a nyelv hátrányait többek között a következőképpen próbálhatják meg kiküszöbölni:

oldal

2.4.4 DO utasitás 24
2.4.5 A BREAK utasitás 25
2.4.6 A NEXT utasitás 25
2.5 A DEFINE utasitás 26
2.6 Az INCLUDE utasitás 26
2.7 A RATFOR hibaüzenetei 27
2.8 A RATFOR Job Control eljárások ismertetése 27
2.9 Korlátozások, figyelmeztetések 28
2.10 Egy RATFOR program 29
3. Összefoglalás 30
Irodalomjegyzék 32

ABSTRACT

In recent years several extended versions of the programming language FORTRAN have been published. These versions add program structures to FORTRAN such as new control flow statements (conditional branches, loops, group of statements etc.) making structured programming easy.

The present report describes two extended versions of FORTRAN: the MORTRAN and the RATFOR languages.

АННОТАЦИЯ

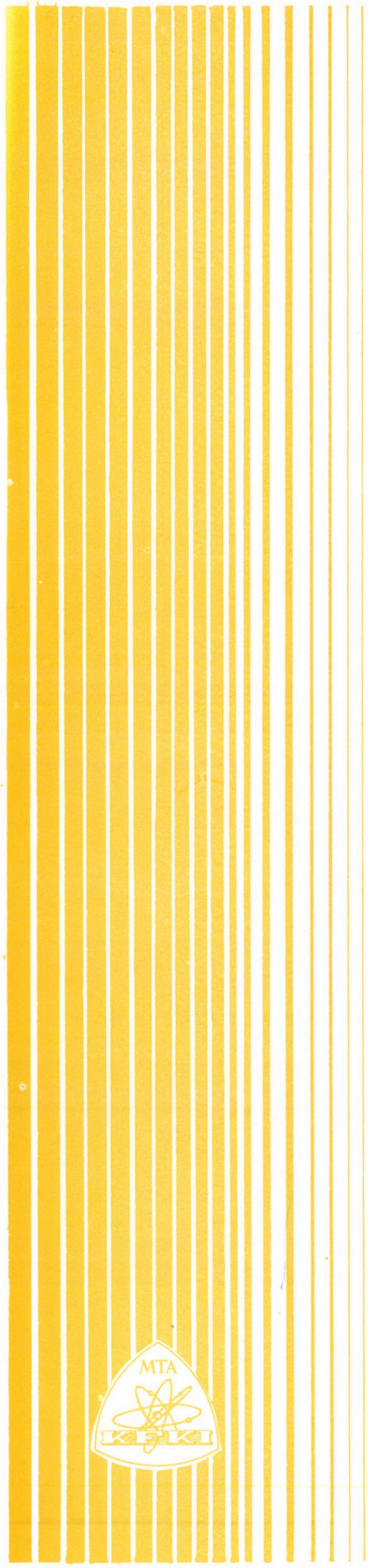
За последние годы было разработано несколько расширенных вариантов языка программирования ФОРТРАН. Эти варианты языка ФОРТРАН дополнены такими программными структурами /команды циклов, команды управления, составные операторы и т.д./, которые дают возможность для написания хороших структурных программ.

В этой работе описаны два таких расширенных варианта языка программирования ФОРТРАН: МОРТРАН и РАТФОР.

KIVONAT

Az utóbbi években a FORTRAN programozási nyelvnek több bővitett változatát dolgozták ki. Ezek a FORTRAN nyelvet olyan programstrukturákkal (ciklusképzési és vezérlésátadási utasitások, összetett utasitások stb) egészitik ki, amelyek lehetségessé teszik jól strukturált programok irását.

Ebben a dolgozatban a FORTRAN nyelv két ilyen kiterjesztését, a MORTRAN és RATFOR nyelvet ismertetjük.



Kiadja a Központi Fizikai Kutató Intézet
Felelős kiadó: Lőcs Gyula
Szakmai lektor: Zimányi Magdolna
Példányszám: 350 Törzsszám: 81-576
Készült a KFKI sokszorosító üzemében
Felelős vezető: Nagy Károly
Budapest, 1981. október hó