

Magiszter

Számítástechnikai szerkesztőség

ZTOOLS

Integrált segédprogramok

MS és Turbo C környezetben

Programvédelem nélkül!

Szoftver
IBM PC XT AT
és
kompatibilis mikroszámítógépekre



GRÄFF-INFOKER

ZTOOLS

Integrált C könyvtárak

195726

Áff Zoltán – INFOKER Számítástechnika Alkalmazási Kiszövetkezet

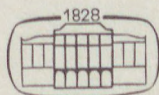
ZTOOLS

C könyvtárak

MS C 5.xx - től

Turbo C 1.5 - től

**MTA
KIK**



Akadémiai Kiadó, Budapest 1988

887301

Szoftver - dokumentáció
Sorozatszerkesztő: Pintér Tibor
A kiadványt szakmailag ellenőrizte: Schill Róbert

©Akadémiai Kiadó és Nyomda Vállalat
Gräff – INFOKER 1988
Minden jog fenntartva

REGYAR
TUDOMÁNYOS AKADÉMIAI
KÖNYVTÁRA

A szoftver - dokumentáció a $\text{M}\text{A}\text{T}\text{E}\text{X}$ programcsomaggal készült
©Akadémiai Kiadó és Nyomda Vállalat · Gräff – INFOKER

M. TUD. AKADÉMIAI KÖNYVTÁRA
Könyvtár 15.9.87 / 19. 88. sz.

Tartalomjegyzék

ZKEY

1. Billentyűzet-kezelés	11
1.1. Bevezetés	11
1.2. A ZKEY lemez tartalma	12
1.3. Használat	13
2. Konstansok	14
3. Típusdefiníciók	16
4. Globális változók	18
5. Eljárások, függvények	21
5.1. Inicializálás/lezárás	21
5.2. Billentyűzet-kezelés	22
5.3. WAIT függvény kezelése	26
5.4. HOT-KEY-k kezelése	27
5.5. MACRO kezelés	30
5.6. Késleltető eljárások	34
5.7. Hanggeneráló eljárások	35

ZWINDOW

6. Ablak-kezelés	41
6.1. Bevezetés	41
6.2. A ZWINDOW lemez tartalma	43
6.3. Használat	43
7. Konstansok	45

8. Típusdefiníciók	48
9. Globális változók	51
10. Eljárások, függvények	56
10.1. Inicializálás/lezárás	56
10.2. Ablak megnyitása és lezárása	58
10.3. Az aktuális ablak kezelése	62
10.4. Láthatóság állítása	64
10.5. Az ablak részeinek törlése	66
10.6. Kiírás az ablakba kurzor mozgatással	68
10.7. Kiírás kurzor mozgatás nélkül	71
10.8. Az ablak adatainak változtatása	76
10.9. Helyfoglalás lekérdezése	83
10.10. Az ablak tartalmának lekérdezése	85
10.11. Az ablakok zoom-olása	89
10.12. Státusz-sor kezelése	92
10.13. Az ablak mozgatása és megtekintése	95
10.14. Az ablak egyes adatainak lekérdezése	98
10.15. Ami eddig nem szerepelt	99

ZMENÜ

11. Menükezelés	107
11.1. Bevezetés	107
11.2. A ZMENU lemez tartalma	110
11.3. Használat	110
12. Konstansok	111
13. Típusdefiníciók	114
14. Globális változók	117
15. Eljárások, függvények	119
15.1. Inicializálás/lezárás	119
15.2. Menü létrehozása	121
15.3. Menü megnyitás, lezárás, meghívás	129
15.4. Menü adatainak lekérdezése	136

15.5. Menüelemek adatainak beállítása	139
15.6. Menü adatainak változtatása	142
15.7. Aktív elem kijelölése	150
15.8. Következő elem meghatározása	151

ZINPUT

16. Adatbevitel	155
16.1. Bevezetés	155
16.2. A ZINPUT lemez tartalma	156
16.3. Használat	157
17. Konstansok	158
18. Típusdefiníciók	161
19. Globális változók	163
20. Eljárások, függvények	165
20.1. Inicializálás/lezárás	165
20.2. Editor	167
20.3. Formátumos input	169
20.4. Mező input	173
20.5. Beépített ellenőrző függvények	175
20.6. Input táblák kezelése	178
20.7. Az input tábla elemei	183
20.8. Egyéb eljárások	189
Tárgymutató	191

ZKEY

Billentyűzet-kezelő eljárások

1. fejezet

Billentyűzet-kezelés

1.1. Bevezetés

A Zkey könyvtár olyan eljárásokat tartalmaz, amelyek a billentyűzet egyszerű kezelését teszik lehetővé. Fontosabb szolgáltatásai a következők.

- Lehetőséget ad a speciális billentyűk igen egyszerű kezelésére (F1, SHIFT/F3, CTRL/Y, ALT/Q, stb).
- Tartalmazza az ellenőrzött billentyű-beolvasást: megadható, hogy mely billentyűk használhatók funkcióként, és csak ezeket fogadja el.
- Egy ún. MACRO lehetőséget biztosít. Ez azt jelenti, hogy egyes billentyűkhöz hozzárendelhető egy billentyűsorozat. Az adott billentyű lenyomását a program úgy érzékeli, mintha az ehhez rendelt billentyűket sorban leütötték volna. Lehetőség van arra is, hogy ilyen makrókat a felhasználó a program futása közben definiáljon.
- HOT-KEY-k definiálása. HOT-KEY-nek azt a billentyűt nevezzük, amelyhez egy függvény rendelhető. Minden olyan alkalommal, amikor a program billentyű lenyomására vár, egy HOT-KEY billentyű lenyomása esetén, meghívódik a hozzá tartozó függvény. (Turbo Pascal: F2 billentyű)
- Megadható egy ún. WAIT függvény. Ha a program billentyű-lenyomásra vár, akkor a WAIT függvény hívódik meg. Mindez a legközelebbi billentyű-leütésig tart. A függvényt a programból lehet beállítani, meghívása le is tiltható.

- Bár nem tartoznak szorosan a billentyűzet-kezeléséhez, mégis ebbe a modulba kerültek a késleltető eljárások. Ezek segítségével lehetőség van a program leállítására adott ideig, vagy egy billentyű lenyomásáig.
- Még egy funkciócsoport tartozik ehhez a modulhoz. Ezek a hanggenerálással kapcsolatos függvények. Segítségükkel beállítható egy adott frekvenciájú hang, amely a program futása közben egyfolytában szól, vagy kikapcsolható az előzőleg beállított hang. Ez a lehetőség a Turbo C-ben megtalálható, a Microsoft C azonban nem biztosítja.

1.2. A ZKEY lemez tartalma

README.ZK	—	A legújabb információk
ZKEY.H	—	A ZKEY header file-ja
ZK_TC_S.LIB	—	Turbo C könyvtár — small
ZK_TC_L.LIB	—	Turbo C könyvtár — large
ZK_MSC_S.LIB	—	Microsoft C könyvtár — small
ZK_MSC_M.LIB	—	Microsoft C könyvtár — medium
ZK_MSC_L.LIB	—	Microsoft C könyvtár — large
ZKEY.C	—	A példaprogram forrásnyelven

1.3. Használat

A billentyűzet-kezelés használatához a programba be kell olvasni (includeolni) a ZKEY.H header file-t. Ez tartalmazza a megfelelő konstans és típusdefiniciókat, valamint a függvények mintáit (prototype). A program szerkesztésekor a következő könyvtárak közül a megfelelőt kell megadni.

ZK_TC_S.LIB	—	Turbo C small modell
ZK_TC_L.LIB	—	Turbo C large modell
ZK_MSC_S.LIB	—	Microsoft C small modell
ZK_MSC_M.LIB	—	Microsoft C medium modell
ZK_MSC_L.LIB	—	Microsoft C large modell

Célszerű a header file-t a C fordító INCLUDE alkönyvtárba tenni, a szükséges .LIB file-t pedig a LIB alkönyvtárba, így lehet a legegyszerűbben használni a programrendszert.

Igen fontos, hogy a billentyűzet-kezelés eljárásait az inicializáló eljárás meghívása előtt nem szabad használni, mert ez beláthatatlan következményekkel jár. Lásd a zk_begin() eljárás ismertetését !

2. fejezet

Konstansok

A ZKEY modul használata esetén a speciális billentyűkre nevekkkel (konstansokkal) lehet hivatkozni. Ezek a ZKEY.H file-ban vannak definiálva. A legalapvetőbeket leszámítva ezek a következők:

ESC	— az ESC billentyű
BKSP	— a BACKSPACE billentyű
RETURN	— az ENTER billentyű
ENTER	— az ENTER billentyű
C_ENTER	— CTRL/ENTER
LF	— LINE FEED (ugyanaz, mint a C_ENTER)
SPACE	— SPACE azaz szóköz
TAB	— tabulátor
BTAB	— Backtab, azaz SHIFT/TAB
A_A ... A_Z	— ALT/A ... ALT/Z
A_1 ... A_0	— ALT/1 ... ALT/0
A_MINUS	— ALT/<minus jel>
A_EQ	— ALT/<egyenlő jel>
F1 ... F10	— F1 ... F10
S_F1 ... S_F10	— SHIFT/F1 ... SHIFT/F10
C_F1 ... C_F10	— CTRL/F1 ... CTRL/F10
A_F1 ... A_F10	— ALT/F1 ... ALT/F10
HOME	— HOME billentyű

END	—	END billentyű
LEFT	—	balra nyíl
RIGHT	—	jobbra nyíl
UP	—	felfelé nyíl
DOWN	—	lefelé nyíl
PGUP	—	PGUP billentyű
PGDN	—	PGDN billentyű
INS	—	INS billentyű
DEL	—	DEL billentyű
C_HOME	—	CTRL/HOME
C_END	—	CTRL/END
C_LEFT	—	CTRL/balra nyíl
C_RIGHT	—	CTRL/jobbra nyíl
C_PGUP	—	CTRL/PGUP
C_PGDN	—	CTRL/PGDN
C_BKSP	—	CTRL/BACKSPACE
CTRL(char)	—	CTRL/<char> Pl: CTRL('Q')

3. fejezet

Típusdefiníciók

Ebben a részben a billentyűzet-kezeléssel kapcsolatos legfontosabb típus-definíciókat ismertetjük. A konstansokhoz hasonlóan ezek is a ZKEY.H file-ban vannak definiálva.

BOOL	— logikai érték, lehetséges értékei a TRUE és a FALSE, de a C nyelv szabványa szerint a FALSE azaz hamis a nulla, és minden más érték igaz.
KEY	— billentyű (nem karakter, mert speciális billentyűk is lehetnek). Ezek vehetik fel értékül az előző pontban felsorolt konstansokat. Ezeken kívül persze bármilyen karaktert is. PI: A_X, 'k', CTRL('Q'), '\x17'
LEN	— Valaminek a mérete, vagy valamitől vett távolsága.
MACRO	— Makrók definiálásakor használatos. A MACRO tulajdonképpen egy KEY tömb, amelyben egy billentyű-sorozat van felsorolva, amelyet egy adott billentyű leütése esetén szimulálni kell. Ez tehát egy pointer, ami KEY típusrú adatra mutat.
FREQ	— Frekvencia, ami hertz-ben értendő. Hanggenerálás esetén használható.
TIME	— Idő, századmásodpercben. Késleltetés és hanggenerálás esetén használatos.

PFV

- Pointer típus. Pontosabban függvényre mutató pointer. Amire mutat, az egy void függvény, mégpedig olyan, aminek egy paramétere sincs. Ilyen függvények WAIT függvényként és hanggeneráló függvényként használatosak. Például:
`void my_wait_fv(void);`

PFK

- Pointer típus. Egy olyan void függvényre mutató pointer, amelynek paramétere KEY típusú. Ilyen függvények használatosak HOT-KEY-k definiálásakor. Például:
`void my_hot(KEY);`

PFE

- Pointer típus. Olyan függvényre mutat, ami void, KEY és LEN típusú paramétereket kap. Azok a függvények, amelyekre ilyen pointer mutat, a program futása közben definiálható makrók esetén használatosak. Ilyen a következő függvény is:
`void def_macro_end(KEY mkey, LEN mlen);`

4. fejezet

Globális változók

A billentyűzet-kezelésnél jól használhatók a globális változók. Ezek segítségével lehet a HOT-KEY és MACRO billentyűket engedélyezni, vagy letiltani. A WAIT függvény és a hanggeneráló függvények működését is az ilyen változók megfelelő beállításával lehet befolyásolni.

wait_func_on

Név: wait_func_on – van-e háttér tevékenység

Használat: #include <zkey.h>

Deklaráció: BOOL wait_func_on;

Default: FALSE

Leírás: Logikai érték. Ha TRUE, akkor a billentyűzet-beolvasó eljárás meghívja a WAIT függvényt, vagyis minden alkalommal, amikor beolvasásra vár. (Ez jó alkalom például a háttér-printelés megvalósítására.) Ha értéke FALSE, akkor ez nem történik meg.

A WAIT függvény a megfelelő eljárással állítható be.

Lásd még: set_wait_func() readkey()

bell_on

Név: bell_on – sípolás engedélyezett

Használat: #include <zkey.h>

Deklaráció: BOOL bell_on;

Default: TRUE

Leírás: Ha értéke TRUE, akkor a programban engedélyezett a hanggenerálás, FALSE esetén nem. A ZKEY-ben definiált hanggeneráló eljárások ezt figyelembe veszik. Célszerű ennek a konvenciónak a betartása a felhasználó programjában is.

Az ellenőrzött billentyű-beolvasó eljárás például sipol, ha rossz billentyűt nyomtak meg. Többek között ezt lehet ennek a változónak a felhasználásával letiltani.

Lásd még: bell() set_bell_func()

hot_keys_on

Név: hot_keys_on – HOT-KEY-k engedélyezettek

Használat: #include <zkey.h>

Deklaráció: BOOL hot_keys_on;

Default: FALSE

Leírás: Logikai érték. Ha igaz, akkor működnek a program által beállított HOT-KEY-k, azaz leütésük esetén meghívódik a hozzájuk tartozó függvény. Ez persze azzal jár, hogy a beolvasás során HOT-KEY-nek definiált billentyűt sosem kaphatunk vissza.

Lásd még: add_hot_key() readkey() macro_keys_on

macro_keys_on

- Név:** macro_keys_on – MACRO-k engedélyezettek
- Használat:** #include <zkey.h>
- Deklaráció:** BOOL macro_keys_on;
- Default:** FALSE
- Leírás:** Logikai érték. Funkciója a hot_keys_on változóéhoz hasonló. Az eltérés csupán annyi, hogy ez a MACRO billentyűkre vonatkozik.
- Lásd még:** add_macro() readkey() hot_keys_on

5. fejezet

Eljárások, függvények

5.1. Inicializálás/lezárás

Azt a két eljárást írjuk le ebben a szakaszban, amelyekre a billentyűzet-kezelés használata előtt és után van szükség. Az inicializáló eljárást mindenképpen meg kell hívni billentyűzet-kezelő eljárás használata előtt.

zk_begin()

- Név:** zk_begin() – Billentyű-kezelés indul
- Használat:** #include <zkey.h>
BOOL zk_begin(LEN kb, LEN macb, LEN hotb);
- Leírás:** A billentyűzet-kezelést végző eljárások számára megfelelő méretű puffereket foglal le. Ezek méretét a három paraméter segítségével lehet megadni. A program belső billentyűzet-pufferének méretét a kb paraméter adja meg. Ennek akkorának kell lennie, hogy a leghosszabb MACRO-hoz tartozó összes billentyű egyszerre beleférjen. Ha nagyobb, nem baj. A hotb a programban egyszerre érvényes HOT-KEY-k maximális számát adja meg. A macb ugyanígy, csak a MACRO billentyűkre vonatkozik.
- Visszatérés:** TRUE, ha sikerült az allokalás. FALSE, ha nincs elég szabad memória.
- Lásd még:** zk_end()
- Példa:** zk_begin(64, 8, 8);

zk_end()

- Név:** zk_end() – Billentyűzet-kezelés vége
- Használat:** #include <zkey.h>
void zk_end(void);
- Leírás:** A billentyűzet-kezelés során használt puffereket felszabadítja. A program vége előtt célszerű meghívni.
- Visszatérés:** —
- Lásd még:** zk_begin()

5.2. Billentyűzet-kezelés

clear_keys()

- Név:** clear_keys() – Billentyűzet-puffer törlése.
- Használat:** #include <zkey.h>
void clear_keys(void);
- Leírás:** A program törli a saját belső pufferét és a DOS billentyűzet-pufferét is.
- Visszatérés:** —
- Lásd még:** key_pressed() readkey() unreadkey()
- Példa:** clear_keys();
key = readkey();

good_key()

- Név:** good_key() – billentyű-vizsgálat
- Használat:** #include <zkey.h>
BOOL good_key(KEY,MACRO);
- Leírás:** Megvizsgálja, hogy a KEY paraméter benne van-e a MACRO-ban. A MACRO ugyanis egy KEY tömb, amelynek az utolsó eleme 0.
- Visszatérés:** TRUE, ha a KEY megtalálható a MACRO-ban, FALSE, ha nem.
- Lásd még:** read_matchkey()
- Példa:** KEY keys[] = { F1,F2,F3,CTRL('Q'),0 };
ret = good_key(ESC,keys); /* ret == FALSE */
ret = good_key(F3, keys); /* ret == TRUE */

key_pressed()

- Név:** key_pressed() – a billentyűzet lekérdezése
- Használat:** #include <zkey.h>
BOOL key_pressed(void);
- Leírás:** Megvizsgálja, hogy van-e beolvasásra váró billentyű a program belső, vagy a DOS saját billentyű-pufferében. A program belső pufferébe az unreadkey() eljárással lehet betenni billentyűt, és így MACRO-t is.
- Visszatérés:** TRUE, ha van beolvasható billentyű (vagy lenyomtak egy billentyűt, vagy valamelyik eljárás visszatett egy billentyű-kódot a pufferbe), FALSE, ha nem.
- Lásd még:** readkey() unreadkey() call_macro()
- Példa:** if(key_pressed())
 printf(" van mit beolvasni ");
else
 printf(" nincs mit beolvasni ");

readkey()

Név: readkey() – billentyű-beolvasás

Használat: #include <zkey.h>
KEY readkey(void);

Leírás: Beolvas egy billentyűt. Ha van a program saját pufferében (ahova az unreadkey() tette), akkor onnan veszi a billentyű-kódot, ha ott nincs, akkor a DOS-tól kér. Ha sehol sem talál, akkor a wait_func_on változó értékétől függően meghívja a WAIT függvényt. Ugyanez az eljárás kezeli a HOT-KEY és MACRO funkciókat.

Visszatérés: Az első olyan billentyű kódja, amelynek lenyomására nem kellett függvényt aktivizálni és nem volt MACRO sem definiálva.

Lásd még: read_matchkey() unreadkey()

Példa: key = readkey();

readkey_no_hots()

Név: readkey_no_hots() – billentyű-beolvasás

Használat: #include <zkey.h>
KEY readkey_no_hots(void);

Leírás: Analóg a readkey() eljárással. Az a különbség, hogy nem figyeli a HOT-KEY-ket és a MACRO billentyűket, így minden elképzelhető billentyűt vissza tud adni. A WAIT függvényt azonban ez is használja.

Visszatérés: A lenyomott billentyű kódja.

Lásd még: readkey() unreadkey()

Példa: key = readkey_no_hots();

read_matchkey()

- Név:** `read_matchkey()` – ellenőrzött billentyű-olvasás
- Használat:** `#include <zkey.h>`
`KEY read_matchkey(MACRO good_keys, KEY key);`
- Leírás:** Beolvas egy billentyűt a `readkey()` eljárással, tehát az ott leírtak erre az eljárásra is érvényesek. Ha megkapta a billentyűt, megvizsgálja, hogy megegyezik-e a `good_keys` billentyűk valamelyikével, vagy a második paraméterben megadott billentyűvel. Ha nem, akkor meghívja a `bell()` függvényt, és vár egy újabb billentyűre.
- Visszatérés:** Az első megfelelő billentyű
- Lásd még:** `readkey()` `unreadkey()` `good_key()`
- Példa:**

```
KEY keys[] = { F1, F2, CTRL('Q'), C_HOME, 0 };  
key = read_matchkey(keys, ESC);  
key2 = read_matchkey(keys, F10);
```

unreadkey()

- Név:** `unreadkey()` – egy billentyű betétele a pufferbe
- Használat:** `#include <zkey.h>`
`BOOL unreadkey(KEY key);`
- Leírás:** A paraméterben megadott billentyűt visszateszi a program belső billentyű-pufferébe, ha az még nincs tele. Egymás után több billentyű is visszatehető. Az ezt követő beolvasás az utoljára visszarakott billentyűt fogja megtalálni.
- Visszatérés:** `TRUE`, ha sikerült a visszatétel, azaz nincs tele a belső puffer. `FALSE`, ha tele van a puffer és ezért nem sikerült visszarakni a billentyűt.
- Lásd még:** `readkey()` `key_pressed()`
- Példa:**

```
unreadkey(ESC);  
unreadkey(F10);  
key = readkey(); /* key == F10 */  
key = readkey(); /* key == ESC */
```


5.3. WAIT függvény kezelése

Ebbe a szakaszba mindössze egyetlen függvény került. Ennek az a feladata, hogy a programozó meg tudjon adni egy olyan tevékenységet, amit akkor kell végezni, amikor a program billentyű-leütésre várakozik.

set_wait_func()

- Név:** set_wait_func() – WAIT függvény beállítása
- Használat:** #include <zkey.h>
PFV set_wait_func(PFV wait_fv);
- Leírás:** Beállítja a WAIT függvényt. Ez a függvény hívódik meg abban az esetben, ha a program billentyű-leütésére vár. Be lehet például állítani egy olyan függvényre, ami kinyomtat egy karaktert.
- Visszatérés:** Az előzőleg beállított WAIT függvényre mutató pointer.
NULL, ha nem volt semmi beállítva.
- Lásd még:** readkey() wait_func_on
- Példa:**
- ```
void my_wait()
{ putchar('-'); }
...
old_func = set_wait_func(my_wait);
```



## 5.4. HOT-KEY-k kezelése

HOT-KEY azoknak a billentyűknek a gyűjtőneve amelyekhez hozzá lehet rendelni egy függvényt. A program minden esetben, ha billentyű-leütésre vár, és egy ilyen billentyűt kap válaszul, akkor meghívja a billentyűhöz tartozó függvényt, és paraméterként átadja a billentyű kódját is. Lehet ugyanis, hogy több billentyűhöz ugyanaz a függvény van hozzárendelve.

Jó példa a HOT-KEY-re a Turbo Pascal és a Turbo C fejlesztői környezetben az F2 funkció billentyű, melynek hatására az éppen szerkesztett file kimentése történik meg.

---

### add\_hot\_key()

---

- Név:** add\_hot\_key() – HOT-KEY beállítása
- Használat:** #include <zkey.h>  
BOOL add\_hot\_key(KEY hot\_key, PFK hot\_fv);
- Leírás:** Definiál egy HOT-KEY-t. Ezt követően a billentyű leütésére a megadott függvény meghívódik és paraméterként megkapja a billentyű-kódot (lehet, hogy több HOT-KEY-hez egy függvény tartozik). Megengedett egy billentyű többszöri definiálása, ekkor mindig az utolsó definíció van érvényben. Egy definíció törlése esetén mindig a megelőző definíció érvényes.
- Visszatérés:** TRUE, ha sikerült a definiálás, azaz nem volt tele a HOT-KEY puffer, FALSE, ha nem sikerült.
- Lásd még:** readkey() del\_hot\_key() ask\_hot\_key() call\_hot\_key()  
clear\_hot\_keys() hot\_keys\_on



**Példa:**

```
void fv(key)
KEY key ;
{ printf("%d",key); }
...
add_hot_key(F10,fv); /* beállítja az F10-re */
hot_keys_on = TRUE ; /* lehet hot-key */
key = readkey(); /* most nyomj F10-et */
```

---

## ask\_hot\_key()

---

**Név:** ask\_hot\_key() – HOT-KEY lekerdezés

**Használat:** #include <zkey.h>  
BOOL ask\_hot\_key(KEY key);

**Leírás:** Megvizsgálja, hogy az adott billentyű HOT-KEY-ként van-e definiálva.

**Visszatérés:** TRUE, ha így van definiálva. FALSE, ha nincs.

**Lásd még:** add\_hot\_key() del\_hot\_key() clear\_hot\_keys()

**Példa:**

```
if(ask_hot_key(F9))
 printf("igen");
else
 printf("nem");
```

---

## call\_hot\_key()

---

**Név:** call\_hot\_key() – HOT-KEY meghívása

**Használat:** #include <zkey.h>  
BOOL call\_hot\_key(KEY key);

**Leírás:** Egy HOT-KEY direkt meghívása. Ez azt jelenti, hogy meghívódik a HOT-KEY-hez rendelt függvény, és megkapja paraméterként a HOT-KEY kódját. Nem engedélyezi, hogy ugyanazt a HOT-KEY-t újra lehessen hívni ( nincs rekurzió).

**Visszatérés:** TRUE, ha van ilyen HOT-KEY. FALSE, ha nincs.

**Lásd még:** add\_hot\_key() del\_hot\_key() readkey()

**Példa:**

```
add_hot_key(F1,fv1);
call_hot_key(F1); /* fv1(F1); */
```



---

## clear\_hot\_keys()

---

**Név:** clear\_hot\_keys() – Összes HOT-KEY törlése

**Használat:** #include <zkey.h>  
void clear\_hot\_keys(void);

**Leírás:** Törli az összes HOT-KEY definíciót.

**Visszatérés:** —

**Lásd még:** add\_hot\_key() del\_hot\_key() ask\_hot\_key()

**Példa:**  
add\_hot\_key(F1,help\_fv);  
ret = ask\_hot\_key(F1); /\* ret == TRUE \*/  
clear\_hot\_keys();  
ret = ask\_hot\_key(F1); /\* ret == FALSE \*/

---

## del\_hot\_key()

---

**Név:** del\_hot\_key() – egy adott HOT-KEY törlése

**Használat:** #include <zkey.h>  
BOOL del\_hot\_key(KEY hot\_key);

**Leírás:** Törli az adott HOT-KEY-t. Ha többször volt definiálva, akkor az utolsó definíciót. A törlés után a megelőző definíció érvényes.

**Visszatérés:** TRUE, ha volt ilyen HOT-KEY. FALSE, ha nem volt.

**Lásd még:** add\_hot\_key() clear\_hot\_keys() ask\_hot\_key()

**Példa:**  
add\_hot\_key(F1,fv1);  
add\_hot\_key(F1,fv2);  
ret = call\_hot\_key(F1); /\* fv2(F1); \*/  
del\_hot\_key(F1);  
ret = call\_hot\_key(F1); /\* fv1(F1) \*/



## 5.5. MACRO kezelés

Egy adott billentyűhöz rendelhető billentyűk sorozata. Az adott billentyű lenyomása esetén a program úgy tekinti, mintha a hozzárendelt billentyűket ütötték volna le.

Például: a Turbo Pascal és a Turbo C fejlesztői környezetben az F3 billentyűhöz rendelt MACRO: {A\_F, 'L'}, azaz az ALT-F megnyomása után az L megnyomása. Ez a File menüben rögtön rááll a Load menüpontra.

---

### add\_macro()

---

**Név:** add\_macro() – MACRO beállítása

**Használat:** #include <zkey.h>  
BOOL add\_macro(KEY macro\_key ,MACRO keys);

**Leírás:** Egy MACRO-t definiál. Ettől kezdve, a MACRO billentyű, a megadott billentyűk lenyomásának felel meg a programban. Egy billentyű többszöri definiálása is lehetséges. Ekkor mindig az utolsó definíció van érvényben. Egy definíció törlése esetén pedig az előtte érvényes definíció fog élni.

**Visszatérés:** TRUE, ha sikerült a definiálás, azaz nem volt tele a macro-puffer. FALSE, ha nem sikerült a definiálás, mert már nincs hely a pufferben.

**Lásd még:** readkey() del\_macro() ask\_macro() call\_macro() clear\_macros() macro\_keys\_on

**Példa:**  
KEY keys[] = { F1,F2,F3,F4,F5,F6,ESC,0 };  
add\_macro(F10,keys);  
macro\_keys\_on = TRUE ; /\* lehet macro \*/  
key = readkey(); /\* most kell egy F10 \*/



---

## ask\_macro()

---

- Név:** ask\_macro() – MACRO key lekérdezés
- Használat:** #include <zkey.h>  
BOOL ask\_macro(KEY key);
- Leírás:** Megvizsgálja, hogy az adott billentyű MACRO billentyűként van-e definiálva.
- Visszatérés:** TRUE, ha így van definiálva. FALSE, ha nincs.
- Lásd még:** add\_macro() del\_macro() clear\_macros()
- Példa:**
- ```
if( ask_macro(F9) )
    printf("igen");
else
    printf("nem");
```

call_macro()

- Név:** call_macro() – MACRO meghívása
- Használat:** #include <zkey.h>
BOOL call_macro(KEY macro_key);
- Leírás:** Egy MACRO direkt meghívása. Ez azt jelenti, hogy a MACRO billentyűhöz tartozó kódokat sorban beteszi a program belső billentyű-pufferébe. Természetesen fordított sorrendben, hogy a kiolvasások jó sorrendben történjenek.
- Visszatérés:** TRUE, ha van ilyen MACRO billentyű, és volt elég hely az összes billentyűnek a pufferben. FALSE, ha nincs ilyen MACRO billentyű, vagy nem volt elég hely a pufferben. Ez utóbbi esetben törli az egész puffert.
- Lásd még:** add_macro() del_macro() readkey()
- Példa:**
- ```
KEY keys[] = { F1,F2,F3,F4,0 };
add_macro(AA,keys); /* ALT/A definiálása */
call_macro(AA); /* F1,F2,F3 a pufferbe */
```



---

## clear\_macros()

---

**Név:** clear\_macros() – az összes MACRO törlése

**Használat:** #include <zkey.h>  
void clear\_macros(void);

**Leírás:** Törli az összes MACRO billentyű definícióját.

**Visszatérés:** —

**Lásd még:** add\_macro() del\_macro() readkey() ask\_macro()

**Példa:**

```
add_macro(F1,keys);
ret = ask_macro(F1); /* ret == TRUE */
clear_macros();
ret = ask_macro(F1); /* ret == FALSE */
```

---

## del\_macro()

---

**Név:** del\_macro() – egy adott MACRO törlése

**Használat:** #include <zkey.h>  
BOOL del\_macro(KEY macro\_key);

**Leírás:** Törli az adott MACRO billentyű-puffert. Ha többször volt definiálva, akkor az előző definíció lesz érvényben.

**Visszatérés:** TRUE, ha volt ilyen MACRO billentyű. FALSE, ha nem.

**Lásd még:** add\_macro() clear\_macros() ask\_macro()

**Példa:**

```
add_macro(F1,keys1);
add_macro(F1,keys2);
ret = call_macro(F1); /* keys2 */
...
del_macro(F1);
ret = call_macro(F1); /* keys1 */
```



---

## read\_macro()

---

**Név:** read\_macro() – futás alatti MACRO definíció

**Használat:** #include <zkey.h>  
BOOL read\_macro(KEY macro\_key, LEN maxlen, KEY exitkey,  
MACRO puff, PFM mac\_end\_fv);

**Leírás:** Lehetővé teszi, hogy a felhasználó a program futása közben makrókat definiáljon. A mackey paraméter lesz az a MACRO billentyű, ami definiálható. A maxlen azt adja meg, hogy maximum hány billentyű tartozhat a MACRO billentyűhöz, ekkora ugyanis a puffer, aminek a puff paraméter adja meg a címét. Ide kerülnek a lenyomott billentyűk. A mac\_end\_fv pedig egy függvényre mutató pointer. Ez a függvény akkor hívódik meg, ha a billentyű definíciója véget ér. A függvény paraméterként megkapja a mackey értéket és azt, hogy hány billentyű tartozik a definíciójához.

A mackey akkor lesz MACRO billentyűnek definiálva, amikor a hozzá tartozó billentyűk beolvasása befejeződött. A mac\_end\_fv hívásakor a MACRO billentyű már definiálva van.

**Visszatérés:** TRUE, ha sikerült a MACRO definíciót elindítani. FALSE, ha nem, mert rosszak a paraméterek. Például: puff=NULL.

**Lásd még:** add\_macro() clear\_macros() ask\_macro()

**Példa:**  
ptr = (MACRO)calloc(sizeof(KEY), MAX);  
read\_macro(F1, MAX, A\_MINUS, ptr, NULL);



## 5.6. Késleltető eljárások

Ezek a függvények arra valók, hogy a program futását adott időre leállítsák.

Nem érdemes ugyanis `for(;;)` ciklussal időzíteni, mert az minden gépen más-más ideig fut.

---

### `zk_delay()`

---

- Név:** `zk_delay()` – feltétel nélküli várakozás
- Használat:** `#include <zkey.h>`  
`void zk_delay(TIME time_to_wait);`
- Leírás:** A megadott időre felfüggeszti a program futását. Az időt századmásodpercekben kell megadni.
- Visszatérés:** —
- Lásd még:** `zk_wait()`
- Példa:** `bell();`  
`zk_delay(200); /* két másodperc */`  
`bell();`

---

### `zk_wait()`

---

- Név:** `zk_wait()` – feltételes várakozás
- Használat:** `#include <zkey.h>`  
`void zk_wait(TIME time_to_wait);`
- Leírás:** Adott ideig vagy billentyű-lenyomásig vár. Akkor azonnal abba-hagyja a várakozást.
- Visszatérés:** —
- Lásd még:** `zk_delay()`
- Példa:** `zk_wait(500); /* 5 másodperc */`



## 5.7. Hanggeneráló eljárások

A hanggeneráló eljárások mindegyike olyan, hogy a már ismertetett `bell_on` nevű globális változótól teszi függővé, hogy kiad-e hangot, vagy sem.

---

### `bell()`

---

**Név:** `bell()` – hanggenerálás

**Használat:** `#include <zkey.h>`  
`void bell(void);`

**Leírás:** Ez az általános hanggeneráló függvény, amely figyeli a `bell_on` változót, és csak akkor ad ki hangot, ha engedélyezett. A hanggenerálás azt jelenti, hogy ha be van állítva egy hanggeneráló függvény, akkor azt hívja meg (ez nem csak zajt kelthet, villogtathat is), ha pedig nincs, azaz `NULL`, akkor a szokásos beep-et használja.

**Visszatérés:** —

**Lásd még:** `set_bell_func()` `zk_sound()`

**Példa:**

```
bell_on = TRUE ;
bell();
bell_on = FALSE ;
bell();
```



---

## set\_bell\_func()

---

- Név:** set\_bell\_func() – hanggeneráló függvény beállítása
- Használat:** #include <zkey.h>  
PFV set\_bell\_func(PFV bell\_fv);
- Leírás:** Beállítja a hanggeneráló függvényt. Ezután a bell() függvény ezt a megadott függvényt fogja használni. Ha a NULL-ra állítjuk, akkor a bell() a saját függvényét fogja használni.
- Visszatérés:** Az előzőleg beállított függvényre mutató pointer.
- Lásd még:** bell() zk\_sound() bell\_on
- Példa:**  
set\_bell\_func(my\_sound);  
bell();  
set\_bell\_func( NULL );  
bell();

---

## zk\_sound()

---

- Név:** zk\_sound() – adott hang kiadása
- Használat:** #include <zkey.h>  
void zk\_sound(FREQ freq, TIME time\_to\_beep);
- Leírás:** Adott frekvenciájú hangot adott ideig megszólaltat. A frekvencia hertz-ben, az idő, szokás szerint, a másodperc század részében adható meg. Célszerű úgy használni, hogy definiálunk egy függvényt, amely a megfelelő paraméterekkel ezt az eljárást hívja meg. Ezt a függvényt, mint hanggeneráló függvényt adjuk meg.
- Visszatérés:** —
- Lásd még:** bell(), set\_bell\_func()
- Példa:**  
void my\_sound(void);  
{ zk\_sound(3000,40); }  
...  
set\_bell\_func(my\_sound);  
bell();



---

## set\_sound()

---

- Név:** set\_sound() – adott hang megszólaltatása
- Használat:** #include <zkey.h>  
void set\_sound(FREQ freq);
- Leírás:** Megszólaltatja az adott frekvenciájú hangot. Ez a hang a függvény meghívása után tovább fog szólni, egészen egy no\_sound() hívásig.
- Visszatérés:** —
- Lásd még:** no\_sound() bell() set\_bell\_func()
- Példa:**

```
for(cnt=500; cnt<=3500; cnt+=5)
 set_sound(cnt);
no_sound();
```

---

## no\_sound()

---

- Név:** no\_sound() – hang kikapcsolása
- Használat:** #include <zkey.h>  
void no\_sound(void);
- Leírás:** Kikapcsolja a hangot, amit a set\_sound() állított be.
- Visszatérés:** —
- Lásd még:** set\_sound()







# ZWINDOW

Ablak kezelő eljárások







## 6. fejezet

# Ablak-kezelés

### 6.1. Bevezetés

A ZWINDOW legfontosabb szolgáltatásai a következők:

- Elsősorban a ZWINDOW az ún. „virtuális screen” technikát használva egy igen intelligens ablak-kezelést valósít meg. Ez azt jelenti, hogy csaknem tetszőleges méretű „screen” hozható létre, akár 100X200-as is. Hogy a screen mely része, hol látsszon a képernyőablakban külön megadható. Az ablakoknak számos jellemzője adható meg. Ilyen például az ablakban érvényes margók, az ablak helyének és méretének változtathatósága vagy, hogy van-e kurzor az ablakban stb.
- Az ablakok láthatatlanná is tehetőek, és ugyanúgy lehet dolgozni velük, mint a látható ablakokkal. Bármikor ismét láthatóvá téve az ablakot megjelennek a láthatatlanság alatt bekövetkezett változások.
- Az ablakok között tetszőlegesen lehet mozogni, azaz ki lehet választani az aktuális ablakot. Az aktuális ablak mindig a többi „fölött” van. Minden ablakhoz tulajdonságok rendelhetők, amelyek csak akkor érvényesek, ha az ablak éppen aktuális. Például különböző keretet lehet definiálni egy ablakhoz, arra az esetre, ha az ablak aktuális, és arra amikor nem.
- Az ablakok szöveggel kitölthetők. Nem csak a belsejükbe, hanem az ablak felső és alsó keretére is (cím, alsó cím) lehet írni. A kiíró eljárások egy része automatikusan állítja az ablakban a kurzor pozícióját, más részük nem. A kiírandó szövegek tartalmazhatnak vezérlő karaktereket, amelyek segítségével például a kiírás színét lehet változtatni. Nincs megkötés arra



nézve, hogy mely ablakba lehet írni. Bármely ablakba irányulhat az output, nemcsak az aktuálisba, ekkor természetesen a kiírt szövegnek csak az a része látszik, amit más ablakok nem takarnak.

- Az ablakok bizonyos részei színezhethők. Például az egész ablak, az ablak kurzor utáni része, az aktuális sor kurzor utáni része, stb.
- Lehetőség van az ablakok „zoom”-olására. Ezen azt kell érteni, hogy az ablak kinagyítható és visszakicsinyíthető. Le lehet kérdezni, hogy egy adott ablak éppen ki van-e nagyítva.
- Az ún. státusz sor kezelését külön eljárások egyszerűsítik. Státusz sort lehet létrehozni, törölni, módosítani, stb.
- Az ablakok paramétereit futás közben szinte tetszés szerint lehet változtatni. Vannak eljárások például az ablakok keretének, címének, színének megváltoztatására. Az ablaknak a képernyőn elfoglalt helye is módosítható. Az erre alkalmas eljárással lehetővé tehetjük a felhasználónak, hogy ő mozgassa az ablakokat a képernyőn, oda ahova akarja.
- Az ablakkezelés inicializálását végző eljárás meghatározza a számítógépnek a program szempontjából legfontosabb paramétereit, mint például a képernyőn lévő sorok és oszlopok számát (EGA kártya 43 soros módja! ), a kártya típusát (Hercules, EGA, CGA). Külön ki kell emelni, hogy a program automatikusan teszteli, hogy lehet-e az adott gépen direkt a képernyőbe írni. Ha igen, akkor az összes eljárás így működik, ha nem, akkor a BIOS-t kell használni.



## 6.2. A ZWINDOW lemez tartalma

|                     |   |                               |
|---------------------|---|-------------------------------|
| <b>README.ZW</b>    | — | A legújabb információk        |
| <b>ZWINDOW.H</b>    | — | A ZWINDOW header file-ja      |
| <b>ZW_TC_S.LIB</b>  | — | Turbo C könyvtár – small      |
| <b>ZW_TC_L.LIB</b>  | — | Turbo C könyvtár – large      |
| <b>ZW_MSC_S.LIB</b> | — | Microsoft C könyvtár – small  |
| <b>ZW_MSC_M.LIB</b> | — | Microsoft C könyvtár – medium |
| <b>ZW_MSC_L.LIB</b> | — | Microsoft C könyvtár – large  |
| <b>ZWINDOW.C</b>    | — | A forrásnyelű példaprogram    |

## 6.3. Használat

A legfontosabb tudnivaló, hogy a ZWINDOW nem használható a ZKEY könyvtár nélkül. Tehát ami a ZKEY használatánál le volt írva, az most is érvényes. Ezen kívül a következőket kell még tudni:

Az ablakkezelő eljárások használatához a programba be kell hívni a ZWINDOW.H header file-t, amely a megfelelő konstans, változó és típusdefiníciókat tartalmazza. Ebben vannak a függvények prototype-jai is.

A következő könyvtárak közül a megfelelőt hozzá kell szerkeszteni a programhoz:

|                     |   |                           |
|---------------------|---|---------------------------|
| <b>ZW_TC_S.LIB</b>  | — | Turbo C small modell      |
| <b>ZW_TC_L.LIB</b>  | — | Turbo C large modell      |
| <b>ZW_MSC_S.LIB</b> | — | Microsoft C small modell  |
| <b>ZW_MSC_M.LIB</b> | — | Microsoft C medium modell |
| <b>ZW_MSC_L.LIB</b> | — | Microsoft C large modell  |



Célszerű a ZWINDOW.H file-t a C fordítónk INCLUDE alkönyvtárába, a library-ket pedig a LIB alkönyvtárba tenni az egyszerűbb használhatóság kedvéért.

Igen fontos, hogy az ablakkezelés eljárásait nem szabad használni az ablakkezelő modul inicializálása előtt Ennek ismertetése a `zw_begin()` eljárásnál található meg.



## 7. fejezet

# Konstansok

Ezek a konstansok a következő szakaszban ismertetendő RET\_VALUE típus lehetséges értékei. Az eljárások többsége az alábbi konstansokat adja vissza, az elvégzendő tevékenység jelzésére:

|                   |   |                                                                                                                                                                   |
|-------------------|---|-------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>GOOD</b>       | — | minden rendben volt                                                                                                                                               |
| <b>BADNUM</b>     | — | rossz ablakszámot kapott az eljárás                                                                                                                               |
| <b>NOTOPEN</b>    | — | a hivatkozott ablak nincs nyitva                                                                                                                                  |
| <b>BADPOS</b>     | — | rossz kurzor vagy margó pozíció lett megadva. Esetleg az ablakot nem lehet az adott helyen létrehozni.                                                            |
| <b>BADPAR</b>     | — | az eljárás rossz paramétert kapott                                                                                                                                |
| <b>FULLWINDOW</b> | — | az ablakkezelő modul inicializálásakor megadott maximális ablakszámot nem lehet túllépni. Erre történt kísérlet.                                                  |
| <b>NOROOM</b>     | — | nincs elég szabad memória az ablaknyitáshoz                                                                                                                       |
| <b>NOTACTIVE</b>  | — | a hivatkozott ablak nem látható, azaz nem aktív ablak                                                                                                             |
| <b>CURRENT</b>    | — | a hivatkozott ablak már aktuális                                                                                                                                  |
| <b>NOTHIDDEN</b>  | — | a hivatkozott ablak nem láthatatlan, vagyis éppen aktív.                                                                                                          |
| <b>NOMOVE</b>     | — | az ablak nem move-olható, ezért az ablakokat mozgó <code>wmove()</code> , <code>wchange_pos()</code> és <code>move_window()</code> eljárás nem tud vele dolgozni. |
| <b>NOWDEF</b>     | — | NULL pointert kapott az eljárás az ablak leíró struktúrára mutató pointer helyett                                                                                 |



|                 |   |                                                                                                                                                                        |
|-----------------|---|------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NOZOOMED</b> | — | a hivatkozott ablak nincs kinagyítva, így erre az ablakra a <code>zoom.in()</code> eljárás hatástalan                                                                  |
| <b>NOZOOM</b>   | — | a hivatkozott ablak nem zoom-olható, az ilyen ablakra a <code>zoom.in()</code> és <code>zoom.out()</code> eljárás hatástalan.                                          |
| <b>MAXSIZE</b>  | — | a hivatkozott ablak teljes méretű, azaz a képernyőn látható mérete a teljes, definiált méret. Az ilyen ablakkal a <code>zoom.out()</code> eljárás nem tud mit kezdeni. |

A következő konstansok az ablakok keretének és címének megadását könnyítik meg. A leggyakrabban használt keretek ugyanis előre definiálva vannak.

|                |   |                                   |
|----------------|---|-----------------------------------|
| <b>TITLE0</b>  | — | üres cím, azaz semmi              |
| <b>BORDER0</b> | — | nincs keret                       |
| <b>BORDER1</b> | — | egy vonalas keret                 |
| <b>BORDER2</b> | — | két vonalas keret                 |
| <b>BORDER3</b> | — | vissz. dupla, függ. szimpla vonal |
| <b>BORDER4</b> | — | vissz. szimpla, függ dupla vonal  |

Ezek a konstansok az ablakok egyes jellemzőit adják meg. Mindegyik egyetlen bit egy két bájtos szóban. Az ablak létrehozásakor, vagy az ablakok jellemzőinek megváltoztatásakor használhatók.

|                |   |                                                                                            |
|----------------|---|--------------------------------------------------------------------------------------------|
| <b>WACTIVE</b> | — | aktív, azaz látható az ablak                                                               |
| <b>WOPEN</b>   | — | nyitott az ablak                                                                           |
| <b>WMOVE</b>   | — | az ablak helyét meg lehet változtatni a megfelelő eljárásokkal                             |
| <b>WSIZE</b>   | — | az ablak méretét meg lehet változtatni                                                     |
| <b>WNOCUR</b>  | — | az ablakban nincs kurzor, vagyis olyan a mérete, hogy nem látszik.                         |
| <b>WNOZOOM</b> | — | az ablakot nem lehet zoom-olni                                                             |
| <b>WADJUST</b> | — | ahhoz, hogy az ablak ráférjen a képernyőre koordinátái a megnyitás előtt megváltoztathatók |
| <b>WTLEFT</b>  | — | az ablak címe a keret a bal szélén legyen                                                  |
| <b>WBLEFT</b>  | — | az ablak alsó címe a keret bal szélére kerüljön                                            |



**WBRIGHT** — az ablak alsó címe a keret jobb szélére kerüljön

Az ablakkezelés inicializálásakor és lezárásakor használatos konstansok az alábbiak. Ezek a `zw_begin()` és a `zw_end()` eljárás paramétereiként használhatók.

**CLR\_SCR** — le kell törölni a képernyőt

**SAVE\_SCR** — meg kell tartani a képernyőt

**CLOSE\_ALL** — a program által létrehozott ablakokat le kell zárni

A színek definiálásához a következő konstansok állnak rendelkezésre, a TURBO C <CONIO.H> header file-nak megfelelően:

BLACK BLUE GREEN CYAN RED MAGENTA BROWN LIGHTGRAY  
DARKGREY LIGHTBLUE LIGHTGREEN LIGHTCYAN LIGHTRED LIGHTMAGENTA YELLOW WHITE

**BG** — a háttérszín szorzója.  $PI: BG * BLUE + WHITE$  -  
kék alapon fehér.

**BLINK** — a villogás konstansa.

$PI: BG * BLACK + YELLOW + BLINK$

**DEF\_ATTR** — a hivatkozott ablak definiáláskor megadott színe.  
Ez az ablak saját színe

**CUR\_ATTR** — a hivatkozott ablak aktuális kiírási színe. Ezzel a  
színnel írtak utoljára az ablakba.



## 8. fejezet

# Típusdefiníciók

|                  |                                                                                                                                                                                                                                                                                                                 |
|------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>ATTR</b>      | — karakter, ablak, stb. színe                                                                                                                                                                                                                                                                                   |
| <b>POS</b>       | — pozíció az ablakban, vagy a képernyőn a (0,0) ponttól számítva                                                                                                                                                                                                                                                |
| <b>BOOL</b>      | — logikai érték. Vagy TRUE, vagy FALSE.                                                                                                                                                                                                                                                                         |
| <b>LEN</b>       | — valaminek a hossza vagy mérete                                                                                                                                                                                                                                                                                |
| <b>WNUM</b>      | — az ablak azonosító száma                                                                                                                                                                                                                                                                                      |
| <b>RET_VALUE</b> | — a legtöbb ablakkezelő eljárás által visszaadott érték. Az eljárás lefutásáról ad tájékoztatást. Például megmondja, hogy rossz ablakra hivatkoztunk.                                                                                                                                                           |
| <b>ADDR</b>      | — pointer egy karaktersorozatra.                                                                                                                                                                                                                                                                                |
| <b>PFA</b>       | — pointer egy ATTR értéket adó függvényre. A függvény két ATTR paraméterrel rendelkezik. Ilyen függvények használhatók az ablakok átszínezésekor a színek módosítására. Erre a wchange_wattr() eljárás ismertetésénél lehet példát találni. Egy ilyen függvény a következő:<br>ATTR fv(ATTR sattr, ATTR cattr); |



**WDATA**

— ablak leíró struktúra. Segítségével definiálni, majd megnyitni lehet az ablakot. Erre példát a wopen() eljárásnál találhatunk.

A következő elemekből áll:

**control** — Az ablakoknak több jellemzőjük van, mint például a move-olhatóság, van benne kurzor, vagy nincs, stb. Ezek már korábban szerepeltek. Az ott ismertetett konstansok mindegyike egy bit ebben az ablak leíró szóban.

**upr,upc** — upper row, upper col, azaz a képernyőn megjelenő ablak bal felső sarkának koordinátái a képernyőn. A képernyő bal felső sarka a (0,0) pont.

**lor,loc** — lower row, lower col, azaz a képernyőn megjelenő ablak jobb alsó sarkának koordinátái.

**title** — az ablak címe. Ez az ablak keretének a felső részébe kerül a WTLEFT tulajdonságtól függően balra, vagy középre.

**btitle** — az ablak alsó címe. A WBLEFT és WBRIGHT tulajdonságoktól függően az ablak keretének alsó részén bal vagy jobb oldalon jelenik meg. Ha egyik sincs megadva, akkor középen.

**border** — az ablak kerete, ha nem ő a kurrens ablak. Ez egy nyolc karakter hosszú stringre mutató pointer, a string karakterei pedig sorra a keret bal felső, felső, jobb felső, jobb oldali, jobb alsó, alsó, bal alsó és bal oldali karakterét adják meg.

**act\_border** — az aktív ablak kerete.

**fill\_ch** — törlőkarakter, amivel törléskor az ablakot tele kell írni. Ez általában a SPACE.

**sattr** — az ablak belsejének a színe, azaz az ablak saját színe.

**battr,tattr** — az ablak keretének és címének a színe, ha az ablak nem a kurrens ablak.



`act_battr`, `act_tattr` — a keret és a cím színe.  
ha az ablak a kurrens ablak.

`depth`, `width` — az ablak valódi mérete.  
Ez nagyobb lehet, mint amekkora a képernyőn  
látszik. Ha azonban kisebb, akkor a képernyőn  
megjelenő ablakból kiszámolható értékekre lesz  
beállítva.



## 9. fejezet

# Globális változók

Talán nem meglepő, hogy a globális változók is ZWINDOW.H file-ban vannak definiálva. Most tehát ezeknek az ismertetése következik:

---

### MIN\_ROW

---

**Név:** MIN\_ROW – a legkisebb sorszámú sor

**Használat:** #include <zwindow.h>

**Deklaráció:** POS MIN\_ROW;

**Default:** 0

**Leírás:** A program által használható legkisebb számú sor megadására szolgál. Ennél feljebb egyetlen ablak sem lehet a képernyőn. Például akkor érdemes alkalmazni, amikor a legfelső sorba egy menüt készítünk. Ilyenkor a MIN\_ROW-t eggyel megnövelve, nem tudunk olyan ablakot létrehozni, amelybe belenyúlna, sőt a már meglévő ablakok sem mozgathatók úgy, hogy azt takarják.

**Lásd még:** MIN\_COL MAX\_ROW



---

## MIN\_COL

---

**Név:** MIN\_COL – a legkisebb oszlopszám

**Használat:** #include <zwindow.h>

**Deklaráció:** POS MIN\_COL;

**Default:** 0

**Leírás:** A MIN\_ROW változó funkcióját tölti be oszlopok esetében.

**Lásd még:** MIN\_ROW MAX\_COL

---

## MAX\_ROW

---

**Név:** MAX\_ROW – a legnagyobb sor száma

**Használat:** #include <zwindow.h>

**Deklaráció:** POS MAX\_ROW;

**Default:** A zw\_begin() eljárás által lekérdezett képernyő méret.

**Leírás:** Ez a változó adja meg a használható sorok számánál eggyel nagyobb értéket! Kezdetben úgy van beállítva, hogy a képernyő legalsó sorának indexénél eggyel nagyobb. Ez CGA és HERCULES kártya esetén 25, egyes EGA kártyák esetén 25, 29, 32 vagy 43, esetleg még ezektől is eltérő. A megfelelő értéket a zw\_begin() eljárás állítja be. Felhasználására a MIN\_ROW-nál leírtak érvényesek.

**Lásd még:** MIN\_ROW MAX\_COL zw\_begin()



---

## MAX\_COL

---

- Név:** MAX\_COL – a legnagyobb oszlopszám
- Használat:** #include <zwindow.h>
- Deklaráció:** POS MAX\_COL;
- Default:** A zw\_begin() eljárás által meghatározott képernyőméret (vízszintes irányban).
- Leírás:** A legnagyobb használható oszlopszámnál eggyel nagyobb értéket tartalmaz! A kezdeti értékét a zw\_begin() inicializáló eljárás határozza meg. Lehetséges értékei általában 40 vagy 80, de egyes EGA kártyákon van 132 oszlopos mód is van.
- Lásd még:** MAX\_ROW MIN\_COL zw\_begin()

---

## wcurrent

---

- Név:** wcurrent – az aktuális ablak száma
- Használat:** #include <zwindow.h>
- Deklaráció:** WNUM wcurrent;
- Default:** Az egész képernyő ablakának száma
- Leírás:** Ez a változó mindig az aktuális ablak számát tartalmazza. Az aktuális ablak használatához szükséges. A zw\_begin() hívása előtt egy rossz ablakszámot tartalmaz, utána az egész képernyőnyi ablak számát, ami megnyitás nélkül is nyitva van. A képernyőn a kurzor mindig a kurrens ablak saját kurzorának megfelelő helyen van. Ez látszik is, ha a kurrens ablakban nincs letiltva a kurzor, azaz nem tüntettük el a hide\_cursor() eljárással.
- Lásd még:** zw\_begin() wselect() hide\_cursor()



---

## is\_ega

---

|                    |                                                                                                                                                                                                                                                                              |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Név:</b>        | <code>is_ega</code> – EGA kártya                                                                                                                                                                                                                                             |
| <b>Használat:</b>  | <code>#include &lt;zwindow.h&gt;</code>                                                                                                                                                                                                                                      |
| <b>Deklaráció:</b> | <code>BOOL is_ega;</code>                                                                                                                                                                                                                                                    |
| <b>Default:</b>    | A <code>zw_begin()</code> eljárás állítja be a megfelelő értékre.                                                                                                                                                                                                            |
| <b>Leírás:</b>     | Ez a változó adja meg, hogy EGA kártya van-e a gépben. Ha értéke <code>FALSE</code> , akkor az <code>is_color_mon</code> változó segítségével lehet dönteni CGA és Hercules között. Az <code>is_color_mon</code> EGA esetén is lehet hamis, ha monochrome módban használják. |
| <b>Lásd még:</b>   | <code>is_color_mon</code> <code>is_snow</code> <code>is_direct</code>                                                                                                                                                                                                        |

---

## is\_color\_mon

---

|                    |                                                                                                             |
|--------------------|-------------------------------------------------------------------------------------------------------------|
| <b>Név:</b>        | <code>is_color_mon</code> – színes monitor                                                                  |
| <b>Használat:</b>  | <code>#include &lt;zwindow.h&gt;</code>                                                                     |
| <b>Deklaráció:</b> | <code>BOOL is_color_mon ;</code>                                                                            |
| <b>Default:</b>    | A <code>zw_begin()</code> eljárás állítja be a megfelelő értékre.                                           |
| <b>Leírás:</b>     | Megadja, hogy a számítógéphez színes monitor van-e kapcsolva. Ezt használhatja a <code>bw()</code> eljárás. |
| <b>Lásd még:</b>   | <code>is_ega</code> <code>is_snow</code> <code>is_direct</code> <code>bw()</code>                           |



---

**is\_snow**

---

**Név:** `is_snow` – havazás

**Használat:** `#include <zwindow.h>`

**Deklaráció:** `BOOL is_snow ;`

**Default:** A `zw_begin()` eljárás állítja be a megfelelő értékre.

**Leírás:** Azt adja meg, hogy megjeleníthető-e havazás a képernyőn. CGA esetén állítódik igazra. Újabb típusú CGA kártyán FALSE-ra kell állítani a `zw_begin()` hívása után. Ekkor az ablakokba irányuló output a havazás figyelése nélkül megy a képernyőre; így lényegesen gyorsabb.

**Lásd még:** `is_ega is_color_mon is_direct`

---

**is\_direct**

---

**Név:** `is_direct` – direct képernyőkezelés

**Használat:** `#include <zwindow.h>`

**Deklaráció:** `BOOL is_direct ;`

**Default:** A `zw_begin()` eljárás állítja be a megfelelő értékre.

**Leírás:** Ez a változó adja meg, hogy lehet-e az adott gépen direkt a képernyő memóriába írni. Ha értéke FALSE, akkor csak BIOS-on keresztül szabad a képernyőt használni. A `zw_begin()` eljárás határozza meg az értékét, ezt jobb nem változtatni, mert az összes ablakkezelő eljárás használja.

**Lásd még:** `is_ega is_color_mon is_snow`



## 10. fejezet

# Eljárások, függvények

Az ablakkezelés eljárásainak a prototype-jai a ZWINDOW.H file-ban találhatók. Itt most funkció szerint csoportosítva következnek, az egyes csoportokon belül pedig névsor szerint találhatók.

### 10.1. Inicializálás/lezárás

---

#### zw\_begin()

---

**Név:** `zw_begin()` – ablakkezelés inicializálása

**Használat:** `#include <zkey.h>`

`#include <zwindow>`

`BOOL zw_begin(int begin_code, LEN maxw, LEN maxs);`



**Leírás:** Ezzel az eljárással tudjuk az egész ablakkezelést inicializálni. Ezt kötelező meghívni minden függvényhasználat előtt. Ez az eljárás foglalja le a megfelelő puffereket, valamint határozza meg a korábban már ismertetett globális változók értékeit.

Az egyszerre nyitva tartható ablakok számát a `maxw`, az egymásra rakható státusz sorok számát pedig a `maxs` paraméterrel lehet megadni. Ezen kívül egy ablakot nyit, ami akkora, mint az egész képernyő. Ennek az ablaknak a tartalmát a paramétertől függően állítja be. Ha a paraméter `SAVE_SCR`, akkor a program indulásakor talált képernyőt másolja bele, és az ablakban érvényes kurzorpozíziót is ennek megfelelően állítja, míg `CLR_SCR` esetén feltölti az ablakot SPACE-ekkel, a kurzort pedig a bal felső sarokba helyezi.

Aktuális ablaknak is ezt az ablakot állítja be.

**Visszatérés:** `TRUE`, ha minden helyfoglalás és inicializálás rendben ment.  
`FALSE`, ha valami nem sikerült.

**Lásd még:** `zw_end()` `MIN_ROW` `MIN_COL` `MAX_ROW` `MAX_COL`

**Példa:** `ret = zw_begin(SAVE_SCR, 16, 8);`

---

## `zw_end()`

---

**Név:** `zw_end()` – ablakkezelés befejezése

**Használat:** `#include <zkey.h>`  
`#include <zwindow>`  
`void zw_end(int end_code);`



**Leírás:** Ezt az eljárást a program végén kell meghívni, amikor már nem akarunk több ablakot kezelni. Az összes ablak által lefoglalt, és az ablakkezelés által használt puffereket szabadítja fel.

Ha a paraméter `SAVE_SCR`, akkor a képernyő tartalma nem változik, vagyis úgy marad, ahogy a `zw_end()` hívásakor volt. Ha `CLR_SCR` paraméterrel hívjuk meg, akkor letörli a képernyőt, a kurzort pedig a bal felső sarokba állítja. A harmadik és egyben utolsó lehetőség a `CLOSE_ALL`. Ha ezt használjuk, akkor az összes ablak eltűnik a képernyőről, kivéve azt az egyet, amit a `zw_begin()` eljárás hozott létre. Ha a `zw_begin()`-t `SAVE_SCR` paraméterrel hívtuk meg, akkor ez azt jelenti, hogy a program kilépéskor visszaállítja a meghívásakor talált képernyőt.

**Visszatérés:** —

**Lásd még:** `zw_begin()`

**Példa:**

```
zw_begin(SAVE_SCR, 16, 8);
...
zw_end(CLOSE_ALL);
```

## 10.2. Ablak megnyitása és lezárása

Itt most azokat az eljárásokat soroljuk fel, amelyek segítségével ablakokat nyithatunk a képernyőn, majd ha már nincs rájuk szükség, le is zárhatjuk őket.

---

### wopen()

---

**Név:** `wopen()` – ablak megnyitása

**Használat:**

```
#include <zkey.h>
#include <zwindow>
RET_VALUE wopen(WDATA *wdata_ptr);
```



**Leírás:**

Ezzel az eljárással lehet megnyitni az előre definiált ablakokat. Ez azt jelenti, hogy a memóriában egy WDATA struktúra megfelelően van kitöltve és a wopen() eljárás ennek a címét kapja meg. Ezek felhasználásával a wopen() meghívja a legalsó szintű ablaknyitó rutint. A WDATA struktúra többféle módon tölthető ki. Eleve be lehet építve a program szövegébe, mint egy statikus, vagy dinamikus változó, betölthető lemezzről, esetleg a program maga tölti ki.

Ez az eljárás alkalmas arra is, hogy láthatatlan ablakot hozunk létre. Ehhez az ablak control szavában nem kell belállítani a WACTIVE bitet. Így létrejön az ablak, de nem lesz látható és persze nem lesz aktív sem. Ha később láthatóvá tesszük, de aktívvá nem, akkor létrehoztunk egy ablakot úgy, hogy a kurrens ablak nem változott. Ezt még egyébként a wxwopen eljárással tehetjük meg.

**Visszatérés:** Ha sikeres volt az ablak megnyitása, akkor egy pozitív számot ad vissza. Ez lesz az ablaknak azonosítója, amivel hivatkozni lehet rá a későbbiekben.

Ha az eredmény nem pozitív, akkor valami hiba lépett fel. Ekkor a visszaadott érték a hiba kódja. Ezek a konstansok ismertetésénél találhatók.

**Lásd még:** wxopen() wxwopen() wclose() WDATA

**Példa:**

```
WDATA wind[] = {{ WACTIVE|WMOVE|WNOZOOM, /* tulajdonsá-
gai */
0,0,10,40, /* helye a képernyőn */
" ez a title ", /* title */
" ez az also cim " /* btitle */
BORDER1,BORDER2, /* keretei */
'-', /* kitöltő karakter */
BG*BLACK+BROWN, /* ablak színe */
BG*BLUE+WHITE, /* keret színe */
BG*BLUE+WHITE, /* title színe */
BG*GREEN+RED, /* aktiv keret színe */
BG*GREEN+YELLOW, /* aktiv title színe */
40,160}}; /* magas, széles */
wopen(wind);
```



---

## wxopen()

---

**Név:** wxopen() – ablak megnyitása

**Használat:** #include <zkey.h>  
#include <zwindow>  
RET\_VALUE wxopen(POS upr, POS upc, POS lor, POS loc,  
ADDR title,  
ATTR sattr, ATTR batttr,  
ATTR act.batttr);

**Leírás:** Ez a közepes szintű ablaknyitó eljárás. Ennek még nincs túl sok paramétere, így könnyen használható, de ezért cserébe le kell mondanunk néhány tulajdonság megadási lehetőségéről.

Amit nem lehet megadni, azok közül az aktív színek, cím és keret ugyanazt az értéket veszi fel, mint a nem aktív megfelelőjük. Az ablak magassága és szélessége pedig a megadott adatokból számítódik, ami annyit jelent, hogy nem lesz "virtuális" az ablak. Akkora lesz, amekkora látszik belőle a képernyőn. Az ablak kitöltő karakter a SPACE . Ez az eljárás azonnal aktív, azaz látható ablakot hoz létre. Ezzel tehát nem lehet létrehozni nem látható, vagy nem kurrens ablakot. A létrehozott ablak WMOVE és WSIZE egyszerre, valamint zoom-olható is.

**Visszatérés:** Ha pozitív a visszatérési érték, akkor az egyben a létrehozott ablak száma, ha negatív, akkor az a hiba kódja.

**Lásd még:** wopen() wxopen() wclose() whide() wshow()

**Példa:** wxopen(0,0,10,40,"title",  
BG\*BLACK+LIGHTGRAY,  
BG\*BLUE+LIGHTGRAY,  
BG\*BLUE+WHITE);



---

## wxxopen()

---

**Név:** wxxopen() – ablak megnyitása

**Használat:** `#include <zkey.h>`  
`#include <zwindow>`  
`RET_VALUE wxxopen(int control,`  
`POS upr, POS upc, POS lor, POS loc,`  
`ADDR title, ADDR btitle,`  
`ADDR border, ADDR act_border,`  
`char fill_ch`  
`ATTR sattr,`  
`ATTR battr, ATTR tattr,`  
`ATTR act_battr, ATTR act_tattr,`  
`LEN depth, LEN width);`

**Leírás:** Ez az az ablaknyitó eljárás, ami valóban mindent tud. Ennek megfelelően igen sok paramétert kell megadni.

Ezt az eljárást használja az összes többi ablaklétrehozó eljárás is, csak a paraméterek jó részét maga határozza meg.

Ha az ablak control szavában a WACTIVE bit be van állítva, akkor a most megnyitott ablakot ez az eljárás mindjárt kurrens ablakká is teszi. Ha nincs beállítva, akkor természetesen nem, mivel ilyenkor meg sem jelenik az ablak a képernyőn.

**Visszatérés:** Ha pozitív a visszatérési érték, akkor a megnyitott ablak száma, ha negatív, akkor annak a hibának a kódja, ami miatt nem sikerült a megnyitás.

**Lásd még:** wopen() wxopen() wclose() wshow() whide()

**Példa:**

```
wxxopen(WACTIVE|WMOVE|WSIZE|WNOCUR
0,0,10,40, /* helye */
TITLE0, /* nincs címe */
TITLE0, /* nincs also cím */
BORDER3,BORDER2, /* keretek */
SPACE, /* kitöltő karakter */
23,14,14,15,15, /* színei */
0,0); /* nem lesz virtuális */
```



---

## wclose()

---

**Név:** wclose() – ablak lezárása

**Használat:** #include <zkey.h>  
#include <zwindow>  
RET\_VALUE wclose(WNUM wnum);

**Leírás:** Ezzel az eljárással lehet egy korábban megnyitott ablakot lezárni. A lezárás eltünteti az ablakot a képernyőről és felszabadítja az általa lefoglalt memóriát is. Paraméterként az ablak számát kapja, amit valamelyik ablaklétrehozó eljárás adott vissza. Ha a kurrens ablakot zárjuk be, akkor az előző kurrens ablak válik aktuálissá.

**Visszatérés:** GOOD, ha sikerült lezárni az ablakot, értéke nulla. Ha nem, akkor egy negatív szám, ami a hiba kódja.

**Lásd még:** wopen() wxopen() wxxopen()

**Példa:** while( !wclose(number) );

## 10.3. Az aktuális ablak kezelése

A kurrens ablak annyiban tér el a többitől, hogy az ablakok közül "legközelebb" van a felhasználóhoz, azaz a többi ablak fölött van. A kurrens ablaknak mindig más lehet a színe, mint a nem kurrensnek, mert színeket kurrens és nem kurrens esetre az ablak megnyitásakor külön meglehet adni. Egyébként minden műveletet ugyanúgy lehet a kurrens és nem kurrens ablakokra alkalmazni: kiírni, törölni, mozgatni stb.



---

## wselect()

---

**Név:** wselect() – az aktuális ablak kijelölése

**Használat:** #include <zkey.h>  
#include <zwindow>  
RET\_VALUE wselect(WNUM wnum);

**Leírás:** Az aktuális (kurrens) ablak kijelölésére szolgál. A jelenleg aktuális ablak keretét átszínezi a megfelelő színekre, majd az újonnan kijelölt ablakot kiveszi addigi helyéről és a többi ablak fölé helyezi. Közben természetesen az ablak keretét is megfelelően átszínezi. A kurzort a képernyőn az új aktuális ablak kurzorának helyére állítja be. Végül beállítja a wcurrent globális változó értékét.

Ha az az ablak, amit aktuálissá tettünk, láthatatlan volt, akkor egyúttal láthatóvá, azaz aktívvá is válik.

**Visszatérés:** GOOD, azaz nulla, ha sikeres volt a kurrens ablak kijelölése, negatív hibakód, ha nem.

**Lásd még:** whide() wshow()

**Példa:** wselect(wnum1);  
wselect(wnum2);



## 10.4. Láthatóság állítása

Minden nyitott ablaknak két állapota lehet. Vagy aktív, és akkor látható, vagy nem. Ha nem aktív, akkor nem látszik a képernyőn, de azért létezik, lehet bele írni, letörölni, változtatni a helyét a képernyőn, stb. Ezek a változások akkor látszódnak, ha ismét aktívvá tesszük az ablakot. Két eljárás foglalkozik az ablakok láthatóságával.

---

### **whide()**

---

**Név:** **whide()** – az ablak láthatatlanná tétele

**Használat:** `#include <zkey.h>`  
`#include <zwindow>`  
`RET_VALUE whide(WNUM wnum);`

**Leírás:** Ez az eljárás az adott ablakot láthatatlanná teszi, azaz eltünteti a képernyőről. Ez nem lezárás, az ablak nyitva marad, csak nem látszik. Bármikor újra láthatóvá tehető.

Ha a kurrens ablakot tesszük nem láthatóvá, akkor az előzőleg aktuális ablakot jelöli ki kurrens ablaknak.

**Visszatérés:** GOOD, azaz nulla, ha sikerült, különben negatív hibakód.

**Lásd még:** `wshow()` `wselect()`

**Példa:**

```
number = wopen(wdata_ptr);
whide(number);
wprint(number, "szoveg", DEF_ATTR);
wshow(number);
wclose(number);
```



---

## wshow()

---

**Név:** wshow() – az ablak láthatóvá tétele

**Használat:** #include <zkey.h>  
#include <zwindow>  
RET\_VALUE wshow(WNUM wnum);

**Leírás:** Ez az eljárás képes rá, hogy újra megjelenítsen egy régebben láthatatlan tett ablakot. Az ablakot a saját helyén teszi láthatóvá, azaz csak az a része fog feltűnni, amit a fölötte levő ablakok nem takarnak. (A wselect() eljárás is láthatóvá tesz, de aktuálissá is egyúttal.)

**Visszatérés:** GOOD, ha minden rendben, vagy negatív hibakód.

**Lásd még:** whide() wselect()

**Példa:**  
number = wopen(wdata\_ptr);  
whide(number);  
...  
wshow(number);

---

## wxhide()

---

**Név:** wxhide() – az ablak láthatatlanná tétele

**Használat:** #include <zkey.h>  
#include <zwindow>  
RET\_VALUE wxhide(WNUM wnum);

**Leírás:** Ideiglenesen nem aktívá teszi az ablakot, de nem tünteti el a képernyőről. Akkor hasznos, ha egyszerre több műveletet akarunk végrehajtani az ablakon (letörölni, kiírni, pozicionálni stb.), és nem akarjuk, hogy közben az ablak össze-vissza villogjon a képernyőn. Ilyenkor meghívjuk a wxhide() eljárást, végrehajtjuk a változtatásokat, majd meghívjuk a wshow() eljárást. Célszerű ugyanis, ha a wxhide() és a wshow() hívások között más ablakokkal nem foglalkozunk, mert furcsa dolgok történhetnek ez esetben.



**Visszatérés:** GOOD, azaz nulla, ha sikerült, különben negatív.

**Lásd még:** `whide()` `wshow()`

**Példa:**

```
wxhide(number);
werase(number, DEF_ATTR);
wprint(number, "szoveg", DEF_ATTR);
wchange_cur(number, 0, 0);
wshow(number);
```

## 10.5. Az ablak részeinek törlése

Most azokat az eljárásokat tárgyaljuk, amelyek az ablakok egészét vagy egy részét törlik. Ez azt jelenti, hogy az ablak adott részét feltöltődik az ablak definiálásakor megadott kitöltő karakterrel.

---

### `werase()`

---

**Név:** `werase()` – az ablak törlése

**Használat:** `#include <zkey.h>`  
`#include <zwindow>`  
`RET_VALUE werase(WNUM wnum, ATTR attr);`

**Leírás:** A num számú ablakot teljes egészében letörli, a kurzort pedig az ablak bal felső sarkába állítja. A törlés a megadott színnel történik. Megadható a DEF\_ATTR és CUR\_ATTR attributum is.

**Visszatérés:** GOOD, ha sikerült. Ha nem, akkor negatív hibakód.

**Lásd még:** `weraeos()` `weraeol()` DEF\_ATTR CUR\_ATTR

**Példa:**

```
number = wopen(wdata_ptr);
werase(number, BG*RED+WHITE);
werase(number, DEF_ATTR);
```



---

## weraeos()

---

- Név:** weraeos() – az ablak végének törlése
- Használat:** `#include <zkey.h>`  
`#include <zwindow>`  
`RET_VALUE weraeos(WNUM wnum, ATTR attr);`
- Leírás:** Az adott ablakon belül érvényes kurzor pozíció utáni részt törli le, a kurzor helyét is beleértve. A törlés a megadott színnel történik. Ez lehet egy konkrét szín, vagy DEF\_ATTR, ill. CUR\_ATTR.
- Visszatérés:** GOOD, ha sikerült, negatív hibakód különben.
- Lásd még:** werase() weraeol() DEF\_ATTR CUR\_ATTR
- Példa:** `number = wxopen(0,0,10,40,"title",23,14,15);`  
`wchange_cur(number,4,5);`  
`weraeos(number,BG*BLUE+LIGHTGRAY+BLINK);`

---

## weraeol()

---

- Név:** weraeol() – az ablak sorvégének törlése
- Használat:** `#include <zkey.h>`  
`#include <zwindow>`  
`RET_VALUE weraeol(WNUM wnum, ATTR attr);`
- Leírás:** Az adott ablakban letörli a kurzor sorának végét, azaz a kurzor után álló részt, a kurzor pozíciót is beleértve. A törlés a megadott színnel történik. Ez lehet egy konkrét szín, vagy DEF\_ATTR, ill. CUR\_ATTR.
- Visszatérés:** GOOD, ha sikerült, negatív hibakód különben.
- Lásd még:** werase() weraeos() DEF\_ATTR CUR\_ATTR
- Példa:** `number = wxopen(0,0,10,40,"title",23,14,15);`  
`wchange_cur(number,0,7);`  
`weraeol(number,BG*BLUE+LIGHTGRAY+BLINK);`



## 10.6. Kiírás az ablakba kurzor mozgatással

Ebben a szakaszban a kiíró eljárásokat vesszük sorba, ezek állítják be, a kurzor pozícióját a soronkövetkező kiírásoknak megfelelően. A kurzort az utolsó kiírt karakter utáni pozícióra állítják. Ha a kurzor utolsó pozícióján áll, akkor a következő sor elejére. Az ablak utolsó sorának utolsó oszlopába való kiírás után az ablak tartalma egy sorral feljebb gördül (scroll)

Ha az ablak valódi mérete nagyobb, mint amekkora a képernyőn látható, és a kurzor a kiírás során olyan helyre kerül, ami már nem esik a képernyő látható részébe, akkor az ablak tartalma annyival kerül feljebb, hogy a kurzor még éppen megjelenjen a képernyőn levő ablakban. Ez teljesen automatikus, nem kell vele külön törődni.

---

### wprint()

---

**Név:** wprint() – kiírás az ablakba

**Használat:** #include <zkey.h>  
#include <zwindow>  
RET\_VALUE wprint(WNUM wnum, ADDR str, ATTR attr);



**Leírás:**

Kiírja a stringet az adott színnel az adott ablakba. Ha kell, akkor átmegy a következő sorba, vagy scrollozza az ablakot. A kurzort az utoljára kiírt karakter utáni pozícióra állítja, ahova a következő kiírás kerül. A szín természetesen lehet DEF\_ATTR és CUR\_ATTR is.

Úgy ír ki, hogy a margókat is figyelembe veszi. Ha azonban a kurzor az első karakter kiírása előtt nem volt a bal margótól jobbra, akkor azt nem állítja be, tehát lehet vele a bal margótól balra is írni. Ha a sor végére ér, akkor a következő sor bal margójára áll. Igen fontos, hogy néhány vezérlőkarakter kivételével minden karaktert ki tud írni. A vezérlőkarakterek a következők:

\1<num> — A kiírási szín átállítása az adott színre. <num> szokás szerint októálsan adandó meg! A string következő karakterei már ezzel a színnel kerülnek kiírásra.

\2 — a kiírási szín visszaállítása a paraméterben megadott színre.

\3 — Attribútum átugrás beállítása. Ez azt jelenti, hogy a string következő karakterei úgy íródnak ki, hogy azt az attribútumot, ami előzőleg ott volt, nem változtatják meg.

\4 — Az ablakkitöltő karakter íródik ki e helyett a karakter helyett. Akkor van értelme, ha nem a SPACE a kitöltő karakter.

\5 — A karakter kiírása helyett az aktuális sor végét törli le az éppen aktuális kiírási színnel.

\6 — A karakter kiírása helyett letörli az ablaknak a kurzor jelenlegi pozícióját követő részt.

\t — A tabulátor karakter helyett annyi SPACE kerül kiírásra, hogy elérjünk a következő tabulátor pozícióra.

\n — Soremelés, és a kurzor a következő sor elejére kerül. Ha kell, akkor scrolloz is. (Ez a decimális 10-es kód)

**Visszatérés:** GOOD, ha rendben van, különben negatív hibakód.

**Lásd még:** watprint() wprint\_words() DEF\_ATTR CUR\_ATTR

**Példa:**

```
wchange_mar(num,2,40);
wprint(num,"alma\1\45
szín\2nem
szín\nkorte",23);
```



---

## watprint()

---

**Név:** `watprint()` – kiírás az adott helyre

**Használat:** `#include <zkey.h>`  
`#include <zwindow>`  
`RET_VALUE watprint(WNUM wnum, POS row, POS col,`  
`ADDR str, ATTR attr);`

**Leírás:** Ugyanazt csinálja, mint a `wprint()`, azzal a különbséggel, hogy kiírás előtt beállítja a kurzor helyét az ablakban. A vezérlő karakterek, a szín ugyanaz, mint a `wprint()` esetén.

**Visszatérés:** `GOOD`, ha rendben van, különben negatív hibakód.

**Lásd még:** `wprint()`

**Példa:** `watprint(num, 3, 6,`  
`"alma\t\tkorte\n",`  
`DEF_ATTR);`

---

## wprint\_words()

---

**Név:** `wprint_words()` – kiírás szótöréssel

**Használat:** `#include <zkey.h>`  
`#include <zwindow>`  
`RET_VALUE wprint_words(WNUM wnum, ADDR str, ATTR`  
`attr);`

**Leírás:** Az ablakba úgy írja ki az adott szöveget, hogy nem tör ketté egyetlen szót sem, hanem átviszi a következő sorba.

Sajátossága, hogy ha a stringben több `SPACE` vagy `\t` van egymás után, akkor azt egyetlen `SPACE`-nek tekintí. Az utolsó szó után tesz egy `SPACE`-t, tehát egy szó két részét nem lehet két különböző hívással kiírni.

Ez a kiírás is a margók közé ír, annyira, hogy a legelső karakter kiírása előtt is megnézi, hogy a kurzor a margókon belül van-e. Ha nem, akkor megfelelően beállítja, és csak utánna ír. A szín a szokásos módon értelmezendő. A `wprint()` eljárásnál ismertetett vezérlőkarakterek szintén használhatók.



**Visszatérés:** GODD, ha rendben van, különben negatív hibakód.

**Lásd még:** `wprint()` `watprint()`

**Példa:**

```
num = wxopen(0,0,10,40,"title",23,14,15);
wchange_mar(num,10,30);
wprint_words(num,"Ezek itt a szavak szepen"
" sorban. Valamikor majd "
"el fogja torni.",DEF_ATTR);
```

## 10.7. Kiírás kurzor mozgatás nélkül

Most azok az eljárások következnek, amelyek úgy írnak az ablakba hogy a kurzort nem igazítják, azaz ott marad, ahol eredetileg volt. Ez azt is jelenti, hogy ki lehet írni az ablaknak egy olyan részébe is, ami nem látszik, sem az írás alatt, sem az írás után.

---

### watput\_attr()

---

**Név:** `watput_attr()` – attributum-változtatás

**Használat:** `#include <zkey.h>`  
`#include <zwindow>`  
`RET_VALUE watput_attr(WNUM wnum, POS row, POS col,`  
`LEN field, ATTR attr);`

**Leírás:** Az adott ablakban egy adott mezőt a megadott színűre változtat. `DEF_ATTR` és `CUR_ATTR` is használható. A kiírás a kurzor pozíciójától teljesen független.

**Visszatérés:** GOOD, vagy negatív hibakód.

**Lásd még:** `watput_str()` `watput_attrstr()` `watput_char()`

**Példa:**

```
num = wopen(wdata_ptr);
watput_attr(num,3,0,10,BG*BLUE+WHITE);
watput_attr(num,3,5,2,DEF_ATTR);
```



---

## watput\_str()

---

- Név:** watput\_str() – kiírás adott mezőbe
- Használat:** #include <zkey.h>  
#include <zwindow>  
RET\_VALUE watput\_str(WNUM wnum, POS row, POS col,  
LEN field, ADDR str);
- Leírás:** Az ablakban adott mezőjében kiírja a megadott stringet úgy, hogy az attribútumokat nem változtatja meg. Ha a string hosszabb a mezőnél, akkor a vége elvész, ha rövidebb, akkor SPACE-ek kerülnek a mező végére. Ez az eljárás az összes karaktert képes kiírni, a vezérlőkaraktereket nem kezeli. A kurzor pozíciójától teljesen független.
- Visszatérés:** GOOD, vagy negatív hibakód.
- Lásd még:** watput\_attr()
- Példa:** num = wxlopen(0,0,10,40,"hihi",23,15,BORDER4);  
watput\_str(num,2,3,25,"ez a fejecske:\2 itt");

---

## watput\_attrstr()

---

- Név:** watput\_attrstr() – színes kiírás a mezőbe
- Használat:** #include <zkey.h>  
#include <zwindow>  
RET\_VALUE watput\_attrstr(WNUM wnum, POS row, POS col,  
LEN field,  
ADDR str, ATTR attr);
- Leírás:** Ezzel az eljárással egy mezőbe lehet kiírni. Az benne a többlet, az előzőekhez képest, hogy meg lehet adni a kiírási szint, sőt két vezérlőkarakter is használható, amelyek segítségével egyszerre többszínű szöveg írható. A két vezérlőkarakter az \1 és \2, jelentésüket a wprint() eljárás leírásnál írtunk le. Az eljárás éppen ezért az 1 és 2 kódú karakterek kivételével az összes többi ki tudja írni. A kurzort nem vizsgálja.



**Visszatérés:** GODD, vagy negatív hibakód.

**Lásd még:** `watput_str()` `watput_attr()`

**Példa:**

```
n = wopen(wdata_ptr);
watput_attrstr(n,2,0,20,"ez\1\45 "
 "jo\2 itt.",23);
```

---

## watput\_char()

---

**Név:** `watput_char()` – karakter kiírása

**Használat:** `#include <zkey.h>`  
`#include <zwindow>`  
`RET_VALUE watput_char(WNUM wnum, POS row, POS col,`  
`char ch);`

**Leírás:** A karaktert az ablak adott pozíciójába írja. Az ott érvényben levő attribútumot nem befolyásolja. Bármilyen karakter használható.

**Visszatérés:** GOOD, vagy negatív hibakód.

**Lásd még:** `watput_charattr()` `wput_char()`

**Példa:**

```
num = wopen(wdata_ptr);
watput_char(num,0,1,'\x1B');
watput_char(num,1,2,'Q');
```

---

## watput\_charattr()

---

**Név:** `watput_charattr()` – színes karakter kiírása

**Használat:** `#include <zkey.h>`  
`#include <zwindow>`  
`RET_VALUE watput_charattr(WNUM wnum, POS row, POS`  
`col, char ch, ATTR attr);`

**Leírás:** Az adott karaktert az ablak adott pozíciójába írja. Az ott érvényes attribútumot is megváltoztatja. Kezeli a DEF\_ATTR és CUR\_ATTR színeket is. Bármilyen karakter használható.

**Visszatérés:** GOOD, vagy negatív hibakód.



Lásd még: `watput_char()` `wput_charattr()`

Példa: 

```
num = wopen(wdata_ptr);
watput_charattr(num, 0, 1, '\x1B', 23);
watput_charattr(num, 1, 2, 'Q', DEF_ATTR);
```

---

## `wput_attr()`

---

Név: `wput_attr()` – attributum-változtatás

Használat: 

```
#include <zkey.h>
#include <zwindow>
RET_VALUE wput_attr(WNUM wnum, ATTR attr);
```

Leírás: Az adott attributumra változtatja az ablakban érvényes kurzor-pozíció színét, de a karakteren nem változtat.

Visszatérés: GOOD, vagy negatív hibakód.

Lásd még: `wput_charattr()` `watput_char()`

Példa: 

```
num = wopen(wdata_ptr);
wchange_attr(num, 2);
wput_attr(num, 23);
```

---

## `wput_char()`

---

Név: `wput_char()` – karakter kiírása

Használat: 

```
#include <zkey.h>
#include <zwindow>
RET_VALUE wput_char(WNUM wnum, char ch);
```

Leírás: Az adott karaktert kiírja az ablakban érvényes kurzor-pozícióba, de azt nem változtatja meg. Az ott érvényben levő attribútumot szintén nem változtatja. Bármilyen karakter használható.

Visszatérés: GOOD, vagy negatív hibakód.

Lásd még: `wput_charattr()` `watput_char()`

Példa: 

```
num = wopen(wdata_ptr);
wchange_char(num, 2, 4);
wput_char(num, '\x1B');
```



---

## wput\_charattr()

---

- Név:** wput\_charattr() – színes karakter kiírása
- Használat:** `#include <zkey.h>`  
`#include <zwindow>`  
`RET_VALUE wput_charattr(WNUM wnum, char ch, ATTR attr);`
- Leírás:** Az adott karaktert az ablak adott pozíciójába kiírja. Az ott érvényben levő attribútumot a megadottra állítja. Használható a DEF\_ATTR és a CUR\_ATTR is. Bármilyen karakter használható.
- Visszatérés:** GOOD, vagy negatív hibakód.
- Lásd még:** wput\_char() wput\_charattr()
- Példa:**  
`num = wopen(wdata_ptr);`  
`wchange_cur(num, 2, 7);`  
`wput_charattr(num, '\x1B', BG*BLUE+RED);`

---

## wput\_attrscreen()

---

- Név:** wput\_attrscreen() – ablak-tartalom kiírása
- Használat:** `#include <zkey.h>`  
`#include <zwindow>`  
`RET_VALUE wput_attrscreen(WNUM wnum, ADDR str);`
- Leírás:** Az adott ablakba egy stringet ír úgy, hogy a string minden sora az ablak új sorába kerüljön. Ez azt jelenti, hogy ha az ablak szélességénél hosszabb a string adott sora, akkor levágja a végét, ha pedig rövidebb, akkor a sor végét SPACE-ekkel tölti fel.  
A stringben a '\n' karakterek jelzik a sorok végét. Az 1 és 2 kódú vezérlő karaktereket értelmezi az eljárás.  
A wget.screen() és a wget\_attrscreen() eljárások által visszaadott szövegek kiírására is alkalmas.
- Visszatérés:** GOOD, vagy negatív hibakód.



**Lásd még:** `wput_charattr()` `watput_char()`

**Példa:**  
`puff = malloc(wsize_attrscreen(wnum));`  
`wget_attrscreen(wnum, puff);`  
`wput_attrscreen(wnum2, puff);`

## 10.8. Az ablak adatainak változtatása

A program megváltoztathatja a már korábban létrehozott ablak néhány jellemzőjét, mint például az ablak címe, alsó címe, kurzor pozíciója. Az itt felsorolt eljárásokkal be lehet állítani az ablakban érvényes margókat is, amelyeket az ablak létrehozásakor nem lehet megadni.

---

### `wchange_cur()`

---

**Név:** `wchange_cur()` – a kurzor pozicionálása

**Használat:** `#include <zkey.h>`  
`#include <zwindow>`  
`RET_VALUE wchange_cur(WNUM wnum, POS row, POS col);`

**Leírás:** Az ablakban a kurzor aktuális pozícióját beállítja a megadott helyre, ha az érvényes pozíció. Ha ez a pozíció éppen nem lenne a képenyőn látható ablakban, akkor annyival mozgatja (scroll) az ablakot a megfelelő irányba, hogy éppen megjelenjen. Elég tehát a kurzort ügyesen állítgatni ahhoz, hogy az ablakot minden irányban mozgatni tudjuk.

**Visszatérés:** GOOD, vagy negatív hibakód.

**Lásd még:** `w_cur()` `wchange_mar()`

**Példa:**  
`num = wopen(wdata_ptr);`  
`wchange_cur(num, 2, 7);`



---

## wchange\_mar()

---

**Név:** wchange\_mar() – a margók beállítása

**Használat:** #include <zkey.h>  
#include <zwindow>  
RET\_VALUE wchange\_mar(WNUM wnum, POS left,  
POS right);

**Leírás:** Beállítja az ablak bal és jobb margóját. Egyes kiírások ehhez tartják magukat, nem írnak a margókon kívülre.

Az ablak létrehozásakor a margók úgy állítódnak be, hogy az egész ablak írható legyen.

**Visszatérés:** GOOD, vagy negatív hibakód.

**Lásd még:** wchange\_cur()

**Példa:**  
num = wxopen(0,0,10,40,"title",23,15,14);  
wchange\_mar(num,15,25);  
wprint\_words(num,"alma fuge dio eper szilva",23);

---

## wchange\_title()

---

**Név:** wchange\_title() – az ablak címének cseréje

**Használat:** #include <zkey.h>  
#include <zwindow>  
RET\_VALUE wchange\_title(WNUM wnum, ADDR title);

**Leírás:** Az adott értékre változtatja az ablak címét. A változás azonnal látható lesz.

**Visszatérés:** GOOD, vagy negatív hibakód.

**Lásd még:** wchange\_btitle()

**Példa:**  
num = wxopen(0,0,10,40,"title",23,15,14);  
wchange\_title(num,"new title");



---

## wchange\_btitle()

---

- Név:** wchange\_btitle() – az ablak alsó címének cseréje
- Használat:** `#include <zkey.h>`  
`#include <zwindow>`  
`RET_VALUE wchange_btitle(WNUM wnum, ADDR btitle);`
- Leírás:** A megadottra változtatja az ablak alsó címét. A változás azonnal látható lesz.
- Visszatérés:** GOOD, vagy negatív hibakód.
- Lásd még:** wchange\_title()
- Példa:** `num = wxopen(0,0,10,40,"title",23,15,14);`  
`wchange_btitle(num,"bottom title");`

---

## wchange\_border()

---

- Név:** wchange\_border() – az ablak kereteinek cseréje
- Használat:** `#include <zkey.h>`  
`#include <zwindow>`  
`RET_VALUE wchange_border(WNUM wnum,`  
`ADDR border, ADDR act_border);`
- Leírás:** Megváltoztatja az ablak kereteit. A változás azonnal látható lesz.
- Visszatérés:** GOOD, vagy negatív hibakód.
- Lásd még:** wchange\_title() wchange\_btitle()
- Példa:** `num = wxopen(0,0,10,40,"title",23,7,15);`  
`wchange_border(num,BORDER2,BORDER2);`



---

## wchange\_battr()

---

- Név:** wchange\_battr() – az ablak keretszínének cseréje
- Használat:** #include <zkey.h>  
#include <zwindow>  
RET\_VALUE wchange\_battr(WNUM wnum,  
ATTR battr, ATTR act\_battr);
- Leírás:** Megváltoztatja az ablak keretszíneit. A változás azonnal látható lesz.
- Visszatérés:** GOOD, vagy negatív hibakód.
- Lásd még:** wchange\_tattr() wchange\_sattr() wchange\_border()
- Példa:** num = wxopen(0,0,10,40,"title",23,7,15);  
wchange\_battr(num,14,15);

---

## wchange\_tattr()

---

- Név:** wchange\_tattr() – az ablak címszínének cseréje
- Használat:** #include <zkey.h>  
#include <zwindow>  
RET\_VALUE wchange\_tattr(WNUM wnum,  
ATTR tattr, ATTR act\_tattr);
- Leírás:** Megváltoztatja az ablak címének a színét. A változás azonnal látható lesz.
- Visszatérés:** GOOD, vagy negatív hibakód.
- Lásd még:** wchange\_battr() wchange\_sattr() wchange\_title()
- Példa:** num = wxopen(0,0,10,40,"title",23,7,15);  
wchange\_tattr(num,14,15);



---

## wchange\_sattr()

---

**Név:** wchange\_sattr() – az ablak belsejének átszínezése

**Használat:** #include <zkey.h>  
#include <zwindow>  
RET\_VALUE wchange\_sattr(WNUM wnum, ATTR sattr);

**Leírás:** Az ablakban levő információk közül azokat, amelyek az ablak saját színével voltak kiírva, a megadott színre átszínezi, egyúttal az ablak saját színét is átállítja.

**Visszatérés:** GOOD, vagy negatív hibakód.

**Lásd még:** wchange\_battr() wchange\_tattr()

**Példa:** num = wxopen(0,0,10,40,"title",23,7,15);  
wchange\_sattr(num,15);

---

## wchange\_wattr()

---

**Név:** wchange\_wattr() – az ablak belsejének átszínezése

**Használat:** #include <zkey.h>  
#include <zwindow>  
RET\_VALUE wchange\_wattr(WNUM wnum, ATTR sattr, PFA attr\_fv);

**Leírás:** Az ablakban levő információk közül azokat, amelyek az ablak saját színével voltak kiírva, a megadott színre átszínezi. Ami nem az ablak színével volt kiírva, azt a felhasználó által megadott színre cseréli. Ezt az attr\_fv pointer segítségével végzi el. A pointer egy olyan függvényre mutat, amely megmondja, hogy melyik szín helyett melyik másik színt kell használni. A függvény megkapja paraméterként az ablak saját színét és azt a színt, ami helyett egy másikat kell megadnia. A visszatérési érték az a szín, amire át kell színezní. Ha csak arra akarjuk használni az eljárást, hogy az ablak teljes tartalmát az adott színűre állítsuk, akkor az attr\_fv paraméterben NULL értéket kell megadnunk.



**Visszatérés:** GOOD, vagy negatív hibakód.

**Lásd még:** `wchange_sattr()`

**Példa:**

```
ATTR my_conv(ATTR sattr, ATTR convattr)
{ if(convattr==RED) return(BLUE);
 else return(sattr); }
...
wchange_wattr(wnum, GREEN, my_conv);
```

---

## `wchange_control_set()`

---

**Név:** `wchange_control_set()` – tulajdonságok beállítása

**Használat:**

```
#include <zkey.h>
#include <zwindow>
RET_VALUE wchange_control_set(WNUM wnum, int control);
```

**Leírás:** Az ablakban beállítja a control paraméterben megadott tulajdonságokat ( WMOVE, WSIZE, WNOZOOM, stb). Az ablakok vezérlő bitjei mind használhatók, a WACTIVE és WOPEN kivételével. Beállítható, hogy az ablakban ne legyen többé kurzor, az ablakot ne lehessen move-olni, az ablak címe ne középen legyen, hanem az ablak bal szélén, stb.

**Visszatérés:** GOOD, vagy negatív hibakód.

**Lásd még:** `wchange_control_reset()`

**Példa:**

```
wchange_control_set(wnum, WBLEFT|WTLEFT);
```



---

## wchange\_control\_reset()

---

- Név:** wchange\_control\_reset() – tulajdonságok letiltása
- Használat:** #include <zkey.h>  
#include <zwindow>  
RET\_VALUE wchange\_control\_reset(WNUM wnum, INT control);
- Leírás:** Az adott ablakban a control paraméterben megadott tulajdonságokat jelző biteket törli az ablak jellemzői közül. Visszaállítható vele az ablakban a kurzor, letiltható az ablakok move-olhatósága, stb. A WACTIVE és a WOPEN bit itt is figyelmen kívül marad.
- Visszatérés:** GOOD, vagy negatív hibakód.
- Lásd még:** wchange\_control\_set()
- Példa:** wchange\_control\_reset(wnum, WNOCUR|WTLEFT|WMOVE);

---

## wchange\_pos()

---

- Név:** wchange\_pos() – az ablak helyének cseréje
- Használat:** #include <zkey.h>  
#include <zwindow>  
RET\_VALUE wchange\_pos(WNUM wnum,  
POS upr, POS upc, POS lor, POS loc);
- Leírás:** Megegyezik a wmove() eljárással.
- Lásd még:** wmove()



## 10.9. Helyfoglalás lekérdezése

Ebben a szakaszban arról lesz szó, hogy az ablak egyes részeinek éppen aktuális memóriaigénye hogyan kérhető le. Erre C-ben igen nagy szükség van, így nem kell állandóan óriási területeket allokálni. Megtudható egy ablak adott sorának vagy az egész ablak tartalmának tárolásához szükséges hely nagysága. Az ablak tartalmát lekérdező eljárások a következő szakaszban kerülnek sorra.

---

### `wsize_line()`

---

- Név:** `wsize_line()` – a sor méretének lekérdezése
- Használat:** `#include <zkey.h>`  
`#include <zwindow>`  
`RET_VALUE wsize_line(WNUM wnum, POS row);`
- Leírás:** Az adott ablak adott sorában levő szöveg tárolásához szükséges byte-ok számát adja meg. Nem veszi figyelembe az attributumokat, csak a szöveget. Figyelmen kívül hagyja a sor végén levő kitöltő karaktereket, ellenben beleszámolja a helyigénybe a záró nullához szükséges byte-ot is.
- Visszatérés:** A tároláshoz szükséges byte-számot adja meg. Ez legalább egy. A negatív érték hibakód.
- Lásd még:** `wsize_screen()` `wget_line()`
- Példa:**
- ```
num = wxopen(0,0,10,40,"title",23,15,14);  
wattrprint(num,3,0,"Ez a szoveg",DEF_ATTR);  
size = wsize_line(num,3);
```

wscreen()

- Név:** `wscreen()` – screen-beli szöveg méretének lekérdezése
- Használat:** `#include <zkey.h>`
`#include <zwindow>`
`RET_VALUE wscreen(WNUM wnum);`
- Leírás:** Az adott ablakban levő szöveg tárolásához szükséges byte-ok számát adja meg. Természetesen nem a képernyőn látható adatok alapján dolgozik, hanem az ablak valódi méretei és tartalma a mérvadó.
Nem veszi figyelembe az attributumokat, csak a szöveget. Figyelmen kívül hagyja a sorok végén levő kitöltő karaktereket, ellenben beleszámolja a helyigénybe a sorok végére rakandó '\n' karaktert és záró nullához szükséges byte-ot is.
- Visszatérés:** Ha nagyobb mint nulla, akkor a szükséges hely mérete, ha kisebb, akkor a hibakód.
- Lásd még:** `wget_screen()` `wscreen_line()` `wscreen_attrscreen()`
- Példa:**
`num = wxopen(0,0,10,40,"title",23,15,14);`
`watprint(num,3,0,"Ez a szoveg\n itt most",23);`
`size = wscreen(num);`

wscreen_attrscreen()

- Név:** `wscreen_attrscreen()` – screen méretének lekérdezése
- Használat:** `#include <zkey.h>`
`#include <zwindow>`
`RET_VALUE wscreen_attrscreen(WNUM wnum);`
- Leírás:** Az adott ablakban levő szövegnek a színinformációkkal együtt való tárolásához szükséges byte-ok számát adja meg. A szövegből figyelmen kívül hagyja a sorok végén levő kitöltő karaktereket. A színeknél pedig csak a szín váltásokhoz szükséges vezérlő karaktereket számolja. Beleszámolja még a sorvégi '\n' karaktereket, és a záró nullát is.

Visszatérés: Ha nagyobb nullánál, akkor a szükséges hely, ha kisebb, akkor a hibakód.

Lásd még: `wget_attrscreen()` `wscreen()`

Példa:

```
num = wxopen(0,0,10,40,"title",23,15,14);
watprint(num,3,0,"Ez \1\3a\2
szoveg",CUR_ATTR); size = wsize_attrscreen(num);
```

10.10. Az ablak tartalmának lekérdezése

Ebben a részben az ablak tartalmát, vagy annak egy részét lekérdező eljárásokat ismertetjük. Ezek némelyike az előző pontban ismertetett helyigény-lekérdező eljárásokkal együtt használható a legjobban.

`wget_char()`

Név: `wget_char()` – karakter lekérdezése

Használat: `#include <zkey.h>`
`#include <zwindow>`
`RET_VALUE wget_char(WNUM wnum, POS row, POS col);`

Leírás: Az adott ablak adott pozícióján álló karaktert kérdezi le.

Visszatérés: Ha nagyobb nullánál, akkor a kívánt karakter kódja, ha negatív, akkor a hiba kódja.

Lásd még: `wget_attr()` `wput_char()`

Példa:

```
num = wxopen(0,0,10,40,"title",23,15,14);
watprint(num,3,0,"Ez a szoveg",DEF_ATTR);
ch = wget_char(num,3,5); /* az 's' betu */
```

wget_attr()

Név: wget_attr() – a szín lekérdezése

Használat: #include <zkey.h>
#include <zwindow>
RET_VALUE wget_attr(WNUM wnum, POS row, POS col);

Leírás: Az adott ablak adott pozíciója álló karakter színét kérdezi le.

Visszatérés: Ha nagyobb nullánál, akkor a kívánt karakter színe, ha negatív, akkor a hiba kódja.

Lásd még: wget_char() wput_attr()

Példa:
num = wxopen(0,0,10,40,"title",23,15,14);
watprint(num,3,0,"Ez a szoveg",79);
attr = wget_attr(num,3,5); /* 79 */

wget_str()

Név: wget_str() – mező lekérdezése

Használat: #include <zkey.h>
#include <zwindow>
RET_VALUE wget_str(WNUM wnum, POS row, POS col, LEN field, ADDR puffer);

Leírás: Az ablak adott mezőjét leolvassa és beteszi a puffer által címzett helyre. Az attributumokat figyelmen kívül hagyja, a mezőt azonban minden esetben végigolvassa. A puffernek megfelelő méretűnek kell lennie.

Egy field hosszúságu mező tárolásához (field+1) byte kell. Ugyanis kell, hogy legyen helye a záró nullának is!

Visszatérés: GOOD, vagy negatív hibakód.

Lásd még: wget_line() wget_screen()

Példa:

```
#define MAX 4
char puff[MAX+1];
num = wxopen(0,0,10,40,"title",23,15,14);
wvprint(num,3,0,"Ez a szoveg",79);
wget_str(num,3,5,MAX,puff); /* 'szov' */
```

wget_line()

Név: wget_line() – sor lekérdezése

Használat: #include <zkey.h>
#include <zwindow>
RET_VALUE wget_line(WNUM wnum, POS row, ADDR puff);

Leírás: Az ablak adott sorában levő szöveget leolvassa és beteszi a puff által címzett helyre. A pufferek megfelelő méretet le lehet kérdezi a wsize_line() függvénnyel, ami beleszámolja a záró nullához szükséges byte-ot is.

Visszatérés: GOOD, vagy negatív hibakód.

Lásd még: wsize_line() wget_str() wget_screen()

Példa: puff = malloc(wsize_line(num,3));
wget_line(num,3,puff);

wget_screen()

Név: wget_screen() – az ablak szöveges tartalmának lekérdezése

Használat: #include <zkey.h>
#include <zwindow>
RET_VALUE wget_screen(WNUM wnum, ADDR puff);

Leírás: Az ablakban levő szöveget leolvassa, és beteszi a puff által címzett helyre. A pufferek elég nagynak kell lennie, amit a legegyszerűbben úgy lehet biztosítani, hogy lekérdezzük a szükséges hely méretét a wsize_screen() függvénnyel, ami beszámolja a sorvégén szükséges '\n'-okat és a záró nullát is.

Visszatérés: GOOD, vagy negatív hibakód.

Lásd még: `wget_line()` `wscreen()` `wput_attrscreen()`

Példa:

```
puff = malloc(wscreen(num));  
wget_screen(num, puff);  
wput_attrscreen(num2, puff);
```

`wget_attrscreen()`

Név: `wget_attrscreen()` – az ablak tartalmának lekérdezése

Használat:

```
#include <zkey.h>  
#include <zwindow>  
RET_VALUE wget_attrscreen(WNUM wnum, ADDR puff);
```

Leírás: Az ablakban levő szöveget leolvassa a színinformációkkal együtt. Az eredményt a puffer által címzett helyre teszi. A puffernek elég nagyra kell lennie: a megfelelő méretet le lehet kérdezni a `wscreen()` függvénnyel, ami beleszámolja a sorvégén szükséges '\n' karaktereket, a színek állításához szükséges vezérlő karaktereket és a záró nullát. Ha az így lekérdezett stringet a `wput_attrscreen()` eljárással kiírjuk, akkor azt kapjuk, amit leolvastunk: vagyis ha a kiírásakor használt ablak kisebb, mint a leolvasott, akkor csak az első sorok jelennek meg, és azoknak is csak az eleje ha pedig nagyobb ablakba írunk ki, akkor a leolvasott adatokon kívüli rész SPACE-ekkel lesz feltöltve.

Visszatérés: GOOD, vagy negatív hibakód.

Lásd még: `wscreen()` `wget_screen()`

Példa:

```
puff = malloc(wscreen(num));  
wget_attrscreen(num, puff);  
wput_attrscreen(num2, puff);
```

wget_wdata()

- Név:** `wget_wdata()` – az ablak adatainak lekérdezése
- Használat:** `#include <zkey.h>`
`#include <zwindow>`
`RET_VALUE wget_wdata(WNUM wnum, WDATA *puff);`
- Leírás:** Az adott ablak érvényben levő adatai alapján kitölti a WDATA struktúrát. Ez több célra is használható. Egyrészt a program felhasználhatja ezeket az adatokat, másrészt elmentheti egy file-ba, hogy a program egy későbbi futásakor ezt beolvasva a felhasználó azokkal az ablakokkal dolgozhasson, amelyeket utoljára beállított.
- Visszatérés:** GOOD, vagy negatív hibakód.
- Lásd még:** `wopen()`
- Példa:** `ptr = (WDATA*) malloc(sizeof(WDATA));`
`wget_wdata(wnum, ptr);`
`printf(" bal felso sarok:%d %d ", ptr->upc, ptr->upc);`

10.11. Az ablakok zoom-olása

Ha egy ablak a képernyőn kisebbnek látszik, mint amekkora valójában, akkor lehet zoom-olni. (Feltéve, hogy nincs megadva az ablakra a tiltó WNOZOOM tulajdonság).

Minden ablakot maximum akkorára lehet kinagyítani, amekkora a valódi mérete, hiszen a definiált méreteken kívül nem lehet kiírni. Ha azonban az ablak valódi mérete elég nagy, akkor a képernyő mérete a mérvadó. A zoom is figyelembe veszi a MIN_ROW, MIN_COL, MAX_ROW, MAX_COL globális változókat, vagyis zoom-olással sem lehet a védett területeket eltakarni.

Ki lehet az ablakokat nagyítani, vissza lehet kicsinyíteni, valamint le lehet kérdezni, hogy nagyított az ablak vagy sem.

wzoom_out()

Név: wzoom_out() – az ablak kinagyítása

Használat: #include <zkey.h>
#include <zwindow>
RET_VALUE wzoom_out(WNUM wnum);

Leírás: Az adott ablakot kinagyítja olyan nagyra, amilyen nagyra csak lehet. Ez azt jelenti, hogy az ablak valódi méretén és a képernyő mérete közül a nagyobbra. A nagyítás úgy történik, hogy az ablak közepe helyben marad, mintha innen nyílna ki az ablak. A bevezetőben említett globális változókkal védett területeket nem foglalja el. A nem zoom-olható ablakok természetesen nem nagyíthatók.

Visszatérés: GOOD, vagy negatív hibakód.

Lásd még: wzoom_in() is_wzoom()

Példa:

```
num = wopen(wdata_ptr);
if ( is_wzoom(num) ) /* ha zoom-olt */
    wzoom_in(num); /* akkor kicsinyít */
else /* egyébként */
    wzoom_out(num); /* nagyít */
```

wzoom_in()

Név: wzoom_in() – az ablak kicsinyítése

Használat: #include <zkey.h>
#include <zwindow>
RET_VALUE wzoom_in(WNUM wnum);

Leírás: Az adott ablakot lekicsinyíti a kinagyítás előtti méretére.

Visszatérés: GOOD, vagy negatív hibakód.

Lásd még: wzoom_out() is_wzoom()

Példa:

```
num = wopen(wdata_ptr);
if( is_wzoom(num) ) /* ha zoom-olt */
    wzoom_in(num); /* akkor kicsinyít */
else /* egyébként */
    wzoom_out(num); /* nagyít */
```

is_wzoom()

Név: is_wzoom() – a zoom-oltság lekérdezése

Használat: #include <zkey.h>
#include <zwindow>
BOOL is_wzoom(WNUM wnum);

Leírás: Megadja, hogy az adott ablak éppen ki van-e nagyítva. Ha nem, akkor lehet nagyítani, ha igen, akkor lehet kicsinyíteni.

Visszatérés: GOOD, vagy negatív hibakód.

Lásd még: wzoom_in() wzoom_out()

Példa:

```
num = wopen(wdata_ptr);
if( is_wzoom(num) ) /* ha zoom-olt */
    wzoom_in(num); /* akkor kicsinyít */
else /* egyébként */
    wzoom_out(num); /* nagyít */
```


10.12. Státusz-sor kezelése

A státusz-sor általában a képernyő legalsó sorában található. Itt jelennek meg a program használatához legfontosabb billentyűk funkciói, vagy a program által éppen végrehajtott tevékenységekről szóló információk. Jó példa a Turbo Pascal és a Turbo C, mindkettő használja ezt a lehetőséget.

A státusz-sorok egymás "fölé" rakhatók. Új státusz sor létrehozásakor az előző státusz sor eltűnik. Ha töröljük az utolsót, akkor ismét felbukkan az előző státusz-sor.

push_state()

Név: push_state() – új státusz-sor létrehozása

Használat: #include <zkey.h>
#include <zwindow>
BOOL push_state(ADDR str, ATTR attr);

Leírás: Létrehoz egy újabb státusz-sort, az előzőek „fölött”. A státusz-sort letörli, mielőtt az új státuszt kiírja, tehát nem marad ott az előző hosszabb szöveg vége az új rövidebb szöveg mögött. Lehet színes státusz-sorokat is megadni, mert az eljárás kezeli az 1 és 2 kódú vezérlőkaraktereket. Ezek használata a wprint() eljárásnál található.

Ha éppen annyi státusz sor van, amennyit a zw_begin() eljárásnak megadtunk, akkor ez az eljárás hatástalan.

Ha a zw_begin() eljárás hívásához megszámloljuk, hogy hány ablakot használ a program, akkor a státusz sorokra csupán egyet kell számolni.

Visszatérés: TRUE, ha sikerült létrehozni a státusz-sort, FALSE, ha már nem lehet többet létrehozni.

Lásd még: pop_state() change_state() hide_state()

Példa:

```
push_state("egyes state sor",23);  
push_state("kettes state sor",79);  
pop_state();  
pop_state();
```

pop_state()

Név: pop_state() – a státusz-sor törlése

Használat:

```
#include <zkey.h>  
#include <zwindow>  
BOOL push_state(void);
```

Leírás: Törli az utoljára létrehozott státusz-sort. Az előző státusz-sor látszik.

Visszatérés: TRUE, ha sikerült törölnie a státusz-sort, FALSE, ha nem volt mit törölni.

Lásd még: push_state() change_state() hide_state()

Példa:

```
push_state("Saving editor file ...",23);  
save_file();  
pop_state();
```

change_state()

Név: change_state() – a státusz-sor módosítása

Használat:

```
#include <zkey.h>  
#include <zwindow>  
BOOL change_state(ADDR str, ATTR attr);
```

Leírás: Megváltoztatja az utoljára létrehozott státusz-sor tartalmát. Ezáltal annak előző tartalma elvész.

Visszatérés: TRUE, ha sikerült a változtatás. FALSE, ha nem. Ez akkor lehet, ha még nincs egyetlen státusz-sor sem.

Lásd még: push_state() hide_state() show_state()

Példa:

```
push_state("Press any key ...",23);  
change_state("\1\56F1\2: Egyes ",79);
```

change_state_end()

Név: `change_state_end()` – a státusz-sor módosítása

Használat: `#include <zkey.h>`
`#include <zwindow>`
`BOOL change_state_end(ADDR str, ATTR attr);`

Leírás: Az utoljára létrehozott státusz-sor végét változtatja meg.
Jó alkalom arra, hogy a CAPS, SCROLL és NUMLOCK billentyűk állását megjelenítsük. Figyelni kell azonban, hogy ha erre a célra használjuk, akkor minden `push_state()`, `change_state()` és `pop_state()` hívás után valóban használjuk !

Visszatérés: TRUE, ha sikerült a változtatás. FALSE, ha nem.

Lásd még: `push_state()` `hide_state()` `show_state()`

Példa: `change_state_end("CAPS NUM", BG*LIGHTGRAY);`

hide_state()

Név: `hide_state()` – a státusz-sor eltüntetése

Használat: `#include <zkey.h>`
`#include <zwindow>`
`BOOL hide_state(void);`

Leírás: A státusz-sorokat változatlanul hagyja, de láthatatlanná teszi.
A `show_state()` eljárással válhatnak ismét láthatóvá.

Visszatérés: TRUE, ha sikerült. FALSE, ha nem. Ez akkor lehet, ha nincs státusz-sor, vagy már eleve láthatatlan.

Lásd még: `show_state()`

Példa: `push_state("egyres state sor", 23);`
`hide_state();`
`...`
`show_state();`

show_state()

- Név:** `show_state()` – a státusz-sor megjelenítése
- Használat:** `#include <zkey.h>`
`#include <zwindow>`
`BOOL show_state(void);`
- Leírás:** Az előzőleg láthatatlanná tett státusz-sort ismét láthatóvá teszi.
- Visszatérés:** TRUE, ha sikerült, FALSE, ha nem. Ez akkor lehet, ha nincs státusz-sor, vagy már eleve látható.
- Lásd még:** `hide_state()`
- Példa:** `push_state("egy sor", 23);`
`hide_state();`
`...`
`show_state();`

10.13. Az ablak mozgatása és megtekintése

Ez a szakasz írja le az ablakok mozgatására és megtekintésére szolgáló eljárásokat.

Az ablak mozgatása azt jelenti, hogy az ablakot a képernyőn el lehet tolni, és a méretét is meg lehet változtatni.

Az ablakok megtekintésére akkor van szükség, ha az ablak valódi mérete nagyobb, mint amennyi a képernyőn látható. Ekkor a scroll segítségével az ablakot a képernyőn minden irányban mozgathatjuk.

wmove()

Név: wmove() – az ablak mozgatása programból

Használat: #include <zkey.h>
#include <zwindow>
RET_VALUE wmove(WNUM wnum, POS upr, POS upc, POS lor, POS loc);

Leírás: Az adott screen-hez tartozó képernyő-ablak koordinátáit változtatja meg, ha ezek értelmes adatok. Az eddigi képernyő ablakot eltünteti, és az új helyen létrehozza. Lehet az ablak méretét és helyét is változtatni, akár egyszerre is. Az ablak pozíciója nem változik, azaz ugyanazok az ablakok maradnak alatta és fölötte, amik addig voltak.

Ha az ablakban be van állítva a WADJUST bit, akkor rossz ablakpozíciók esetén megpróbálja az ablakot eltolni úgy, hogy a képernyőre kerüljön.

Visszatérés: GOOD, ha sikerült a mozgatás. Ha nem, akkor negatív hibakód.

Lásd még: move_window()

Példa: num = wxopen(0,0,10,20,"",23,15,14);
wmove(num,1,0,10,20); /* lejjebb csuszik */

move_window()

Név: move_window() – az ablak mozgatása

Használat: #include <zkey.h>
#include <zwindow>
RET_VALUE move_window(WNUM wnum, KEY exit_key);

Leírás: Lehetőséget ad a felhasználónak arra, hogy maga mozgassa az ablakokat oda, ahova akarja. Egyúttal módosíthatja a méretüket is. Ezt addig teheti, amíg az exit billentyűt meg nem nyomja. A mozgatás idejére egyébként az ablak kerete és alsó címe megváltozik. Hogy mire, azt a BORDER_MOVE és BTITLE_MOVE globális változók adják meg.

A következő billentyűk használhatók:

UP DOWN LEFT RIGHT SHIFT/UP SHIFT/DOWN
SHIFT/LEFT SHIFT/RIGHT CTRL/LEFT CTRL/RIGHT

Visszatérés: GOOD, ha sikerült a mozgatás. Ha nem, akkor negatív hibakód.

Lásd még: wmove() show_window()

Példa: move_window(num, ESC);

show_window()

Név: show_window() – az ablak megtekintése

Használat: #include <zkey.h>
#include <zwindow>
RET_VALUE show_window(WNUM wnum, KEY exit_key);

Leírás: Lehetőséget ad a felhasználónak arra, hogy a képernyőn levő ablakban megtekintse az ablak teljes tartalmát, amíg az exit funkcióju billentyűt le nem nyomjuk. Ennek során az ablak kerete és alsó címe megváltozik. Ezt a BORDER_SHOW és BTITLE_SHOW globális változók szabályozzák.

A következő billentyűk használhatók:

UP DOWN LEFT RIGHT HOME END PGUP PGDN
CTRL/PGUP CTRL/PGDN

Visszatérés: GOOD, vagy negatív hibakód.

Lásd még: move_window()

Példa: show_window(num, ESC);

10.14. Az ablak egyes adatainak lekérdezése

w_open()

- Név:** w_open() – a nyitottság lekérdezése
- Használat:**

```
#include <zkey.h>
#include <zwindow>
BOOL w_open(WNUM wnum);
```
- Leírás:** Megadja, hogy van-e nyitott ablaka az adott számmal. Akkor lehet rá szükség, ha nem tudjuk, hogy az adott számú ablakot lezárta-e már a program.
- Visszatérés:** TRUE, ha van, FALSE, ha nincs.
- Lásd még:** wopen() wclose()
- Példa:**

```
num = wopen(wdata_ptr);
ret = w_open(num); /* TRUE */
wclose(num);
ret = w_open(num); /* FALSE */
```

w_active()

- Név:** w_active() – az ablak láthatóságának lekérdezés
- Használat:**

```
#include <zkey.h>
#include <zwindow>
BOOL w_active(void);
```
- Leírás:** Megadja, hogy az adott ablak látható, azaz aktív-e.
- Visszatérés:** TRUE, ha az ablak látható. FALSE, ha nem látható, vagy esetleg nincs is nyitva.
- Lásd még:** whide() wshow()

Példa:

```
while(num);  
ret = w_active(num); /* FALSE */  
wshow(num);  
ret = w_active(num); /* TRUE */
```

w_cur()

Név: w_cur() – a kurzorpozíció lekérdezése

Használat:

```
#include <zkey.h>  
#include <zwindow>  
RET_VALUE w_cur(WNUM wnum, POS* row, POS* col);
```

Leírás: Az ablakban éppen érvényes kurzorpozíciót kérdezi le.

Visszatérés: GOOD, vagy negatív hibakód.

Lásd még: wchange_cur()

Példa:

```
wchange_cur(num, 3, 6);  
w_cur(num, &row, &col); /* row=3, col=6 */
```

10.15. Ami eddig nem szerepelt

Már csak néhány eljárás van hátra. Ezek igazából nem illettek egyik szakaszba sem.

hide_cursor()

Név: `hide_cursor()` – a kurzor eltüntetése

Használat: `#include <zkey.h>`
`#include <zwindow>`
`void hide_cursor(void);`

Leírás: A kurzort láthatatlanná teszi, bármely ablakban is van. Ez egészen addig érvényben lesz, amíg a `show_cursor()` eljárás hatására újra láthatóvá nem válik a kurzor.

Visszatérés: —

Lásd még: `show_cursor()`

Példa: `hide_cursor();`
`wchange_cur(wnum, 5, 6);`
`weraeos(wnum, DEF_ATTR);`
`wchange_cursor(wnum, 0, 0);`
`show_cursor();`

show_cursor()

Név: `show_cursor()` – a kurzor megjelenítése

Használat: `#include <zkey.h>`
`#include <zwindow>`
`void show_cursor(void);`

Leírás: A korábban láthatatlanná tett kurzort újra láthatóvá teszi.

Visszatérés: —

Lásd még: `hide_cursor()`

Példa: `show_cursor();`

adjust_window()

Név: adjust_window() – az ablak igazítása

Használat: #include <zkey.h>
#include <zwindow>
BOOL adjust_window(POS* upr, POS* upc, POS* lor, POS* loc);

Leírás: Az adott koordinátákat úgy módosítja, hogy beleférjen az ezekkel létrehozott ablak a MIN_ROW, MIN_COL, MAX_ROW, MAX_COL változók által meghatározott képernyőrészbe. A módosítás csak megfelelő irányú eltolást jelent, az ablak méretét nem változtatja meg. Ha a koordináták eleve jók voltak, akkor nem történik semmi.

Ezt az eljárást használja az ablakokat megnyitó eljárás is abban az esetben, ha az ablak vezérlőszavában a WADJUST bit be van állítva.

Visszatérés: TRUE, ha sikerült elhelyeznie az ablakot. Ekkor a koordinátákat is megváltoztatja. FALSE, ha nem sikerült az illesztés, ekkor a koordináták változatlanok maradnak.

Lásd még: wxopen() wxxopen() WADJUST

Példa:
MIN_ROW = 3 ; MIN_COL = 5 ;
upr=-5; upc=-5; lor=upr+10; loc=upc+40;
ret = adjust_window(&upr,&upc,&lor,&loc);
printf("%d %d %d %d ",upr,upc,lor,loc);

refresh_screen()

Név: refresh_screen() – a képernyő újraépítése

Használat: #include <zkey.h>
#include <zwindow>
void refresh_screen(void);

Leírás: Az összes ablakot újra felépíti és újra írja az egész képernyőt. Erre akkor lehet szükség, ha a DOS kiír a képernyőre, mondjuk egy "WRITE ERROR READING DRIVE ..." hibaüzenetet.

Visszatérés: —

Példa: refresh_screen();

bw()

Név: bw() – az attributum konverzió

Használat: #include <zkey.h>
#include <zwindow>
ATTR bw(ATTR original_attr);

Leírás: Ez az eljárás végzi a színek konvertálását. Kezdetben csak BW kártya esetén szükséges konverziót végzi, de más alkalmazása is elképzelhető.

A BW konverziót úgy oldja meg, hogy a nagy fényességet és a villogást változatlanul hagyja. Ha az előtér világosabb volt a háttérnél, akkor normál színeket alkalmaz, míg ha a háttér volt világosabb, akkor inverzbe teszi.

Ha valaki ezt használja, akkor ír egy saját bw() eljárást, a linker ezt az eljárást fogja a programhoz szerkeszteni, nem a ZWINDOW könyvtárban levőt.

Igen szép megoldás, ha a program csak úgynevezett „logikai” színeket használ. Például a számokat egytől valameddig. Az egyes logikai színekhez tartozó valódi színt pedig a display típusa alapján egy tömbből veszi. Ekkor a színek változtatására is egy igen egyszerű segédprogram készíthető.

Visszatérés: a konvertált attributum.

Példa:

```
ATTR bw(ATTR attr)
{ if( is_ega && is_color_mon )
    return(ega_colors[attr]);
  if( is_color_mon )
    return(cga_colors[attr]);
  return(bw_colors[attr]);
}
```


ZMENÜ

Menükezelő eljárások

11. fejezet

Menükezelés

11.1. Bevezetés

A ZMENU igen kényelmes, rugalmas és tetszőlegesen konfigurálható menükezelést tesz lehetővé. Vannak olyan lehetőségei is, amelyek teljesen szokatlanok, így talán a „menükezelő” elnevezés nem is fedi pontosan a lényegét. A legfontosabb lehetőségek rövid ismertetése itt található, míg részletes leírás az egyes eljárások bemutatásánál.

- A menük létrehozására több, különböző szintű eljárás is használható. Ez azt jelenti, hogy az egyes eljárások már néhány paraméter megadása mellett is képesek felépíteni a menüt oly módon, hogy a tulajdonságok egy részét maguk határozzák meg. Erre a legjobb példa az az eljárás, amely egy ablak tartalmából egy menüt készít. Csupán ki kell írunk az ablakba a menü elemeit — az ablak színétől eltérő színnel — és az eljárás létrehozza a menüt, amit azonnal fel is lehet használni. Ez igen egyszerű lehetőséget nyújt szöveg-érzékeny segítség (context-sensitive help) létrehozására. Ehhez egy adatbázisban el kell tárolni a szövegeket a hozzájuk kapcsolódó kulccsal együtt. Minden szövegben szerepelnie kell néhány kiemelt szónak, ami annyit jelent, hogy előtte és utána a ZWINDOW könyvtár leírásában található módon színvezérlő karaktereknek kell lenniük. Az ilyen szöveget ki lehet írni egy ablakba, és felépíteni belőle egy menüt. A menüből aztán ki lehet választani valamelyik kiemelt szót, amelyhez elővehetjük az adatbázisból a help szöveget, és kiirathatjuk. Ebből újból egy menüt lehet készíteni stb.

Természetesen vannak olyan eljárások is, amelyek sok paraméterrel rendelkeznek, segítségükkel a menük összes tulajdonságát meg lehet adni.

- Az egyszer már létrehozott menüket nem feltétlenül szükséges törölni. Egyszerűen láthatatlanná tehetők. Ebben az esetben a menü megőrzi minden paramétert. Például azt, hogy melyik elem az aktuális, vagy mely elemek vannak megjelölve stb.
- A menü, természetesen, elemekből épül fel. A ZMENU azonban megengedi, hogy egy menüelem ne csak egy, hanem több sorból álljon. Általában a menüelem egy téglalap az ablakon belül.

Lehetőség van például egészen speciális függőleges menüelemek létrehozására is. Téglalap alakú menüelemek találhatók a ZMENU.C példaprogramban.

- A menüelemeket el lehet rejtteni (hidden). A rejtett, hidden, elem a menüben való válogatás során nem érhető el, amit megváltozott színe is jelez. Természetesen ennek fennáll az ellenkezője is: a már hidden elemeket újra kiválaszthatóvá tehetjük.
- A ZMENU kétféle menüt tud kezelni. Az egyik az ún. kiválasztó menü (choose), ebből egyetlen elemet lehet csak kiválasztani. A másik a „válogató” menü (select), amelyben több elemet lehet kijelölni, és a program egy másik részén le lehet kérdezni, hogy melyek a kijelölt elemek.
- A menük számos tulajdonságát a program futása közben meg lehet változtatni. Például el lehet mozgatni a menü ablakát a képernyőn, meg lehet változtatni a menüelemek színét, a menü ablakának a keretét, sőt maguk a menüelemek is változhatnak. Erre egy jó példa a Turbo Pascal-ban az a menüelem, amivel be lehet állítani, hogy a fordítás a memóriába, vagy lemezre történjen. Ez a menüelem minden kiválasztásakor a két lehetséges formája közül az előző kiválasztás ellenkezőjét veszi fel.
- Minden menühöz meg lehet adni, hogy mely billentyűkkel vezérelhető. Ez azt jelenti, hogy definiálhatunk egy válogató menüt, majd pedig megadjuk, hogy a menüben például az F10 billentyűvel lehet kiválasztani egy elemet. Az F1...F9 billentyűk valami más célt szolgálhatnak. (Lásd a következő pontot). Megadható továbbá egy olyan billentyű, amelynek segítségével választás nélkül ki lehet lépni a menüből (pl. ESC).
- A menüben való válogatás során a megadott billentyűkkel nem csak egy előre meghatározott tevékenységet lehet aktivizálni, hanem függvényeket is meg lehet adni a menükezelő eljárás számára, amelyek a megfelelő

esetekben meghívhatók. Ez a ZMENU fontos lehetősége: megadható egy függvény, amit a menü megnyitásakor meghív a menükezelő eljárás. Ennek a párját a menü lezárásakor kell aktivizálni. Van két további függvénypár, amelyek a menüben való válogatás során hívódnak meg. Az egyik akkor, amikor a menüben rálépünk egy elemre, a másik akkor, amikor lelépünk róla. Végül, de nem utolsó sorban megadható egy olyan függvény is, amit akkor kell meghívni, ha az előző pontban említett billentyűk valamelyikét ütik le.

Természetesen ezeket a függvényeket nem kell mindig minden egyes menühöz megadni. Mindegyiknek van egy alapértelmezése, amelyet akkor lehet alkalmazni, amikor a menüben való válogatás során azt kívánjuk, hogy mindig jelenjen meg az adott menüelemhez tartozó help szöveg. Egy másik lehetséges felhasználás, amikor egy menüelem kiválasztása egy újabb menü megnyitását jelenti. Így tehát eljutottunk az egymásba ágyazott menükhöz, amelyeket nagyon egyszerűen lehet a ZMENU segítségével kezelni.

- A felsoroltakon kívül még számos lehetőség kínálkozik. Ezek közül még egy: Az egyes menük hatással is lehetnek egymásra, és ezért a hatásukat automatikusan átvezeti a megfelelő azt jelenti, hogy az egyik menüben való válogatás során a menükezelő meghívja valamelyik — a felhasználó által megadott — függvényt, és azzal lehetősége van például arra, hogy egy másik menüben megváltoztassa az aktuális elemet. Ha ráadásul ettől a menütől is függött egy harmadik menü, akkor a változás ott is teljesen automatikusan megtörténik.

11.2. A ZMENU lemez tartalma

README.ZK	—	A legújabb információk
ZMENU.H	—	A ZMENU header file-ja
ZM_TC_S.LIB	—	Turbo C könyvtár – small
ZM_TC_L.LIB	—	Turbo C könyvtár – large
ZM_MSC_S.LIB	—	Microsoft C könyvtár – small
ZM_MSC_M.LIB	—	Microsoft C könyvtár – medium
ZM_MSC_L.LIB	—	Microsoft C könyvtár – large
ZMENU.C	—	A példaprogram forrásnyelven

11.3. Használat

A ZMENU használatához feltétlenül szükségesek a ZKEY és a ZWINDOW könyvtárak. Ezek nélkül nem működik.

Az ott ismertetett teendőkön kívül még két dolgot kell tennünk. Az egyik az, hogy a ZMENU.H file-t a programunkba include-oljuk (ez az utasítás a ZKEY.H és a ZWINDOW.H file-ok include-olása után kell álljon), a másik, hogy az alábbi könyvtárak közül a megfelelőt hozzászerekszük a programhoz:

ZM_TC_S.LIB	—	Turbo C small modell
ZM_TC_L.LIB	—	Turbo C large modell
ZM_MSC_S.LIB	—	Microsoft C small modell
ZM_MSC_M.LIB	—	Microsoft C medium modell
ZM_MSC_L.LIB	—	Microsoft C large modell

Célszerű a ZWINDOW.H header file-t a C fordítónkat tartalmazó könyvtár INCLUDE alkönyvtárába, a könyvtár file-okat pedig a LIB alkönyvtárba tenni.

Igen fontos, hogy a menükezelés eljárásait nem szabad a menükezelő modul inicializálása előtt használni. (Lásd a zm_begin() eljárás ismertetését !)

12. fejezet

Konstansok

A ZMENU.H file a menük egyszerűbb kezelésének érdekében tartalmazza az alább definiált konstansokat. Ebben a fejezetben rövid ismertetés következik.

A ZWINDOW program leírásánál már szerepelt a RET_VALUE típus ismertetése. Az alábbi konstansok ennek a típusnak a lehetséges értékei közül valók:

BADMNUM	—	Rossz menüazonosítót kapott az eljárás. Az adott számú menü nem létezik.
BADITEM	—	Rossz elemsorszám. Az adott menüben nincs ilyen sorszámú menüelem.
CURR_ITEM	—	A megadott elem már aktuális, nem lehet újra aktuálissá tenni.
HIDDEN_ITEM	—	A megadott elem egy rejtett elem. Csak láthatóvá tenni lehet.
NORM_ITEM	—	Az adott menüelem nem rejtett, ezért nem lehet láthatóvá tenni.
NULLFVPTR	—	Az eljárás NULL pointert kapott, amikor függvényre mutató pointert kellett volna kap.
BADCHAR	—	A menüben nincs az adott betűhöz tartozó elem.
FULLMEM	—	Nincs elég szabad memória az adott menü létrehozásához.

ESC_PRESSED	—	A felhasználó által megadott, vagy az előre definiált ESC.KEYS billentyűk valamelyikét ütötték le. Ezzel léptek ki a menüből úgy, hogy nem választottak ki előtte semmit.
NOTFOUND	—	A menük megjelölt elemeinek lekérdezésekor már nincs több elem.
NULLSIZE	—	A menüelem területe nulla.

A következő két konstanssal meg lehet adni a felhasználó által definiált függvényben, hogy a függvény meghívása után ki kell-e a programnak lépnie a menüből, vagy sem.

MEXIT	—	Ki kell lépni a menüből
MNOEXIT	—	Nem kell kilépni

A menük megnyitásakor és lezárásakor használatos konstansok:

OPEN_MENU	—	A menü ablakát meg kell nyitni, de nem kell aktuálissá tenni.
ACTIVE_MENU	—	A menü ablakát meg kell nyitni, és aktuális ablakká is kell tenni.
KEEP_MENU	—	A menüből való kilépés után a menü nem tűnik el a képernyőről.
HIDE_MENU	—	A menüből való kilépés után a menü ablakát el kell rejteni, de nem kell bezárni.
CLOSE_MENU	—	A menüből való kilépés után a menü ablakát be kell zárni, de a menü maga megmarad.
REMOVE_MENU	—	A menüből való kilépés után a menüt és a hozzá tartozó ablakot törölni kell.

A menüelemek állapotának lekérdezésekor a következő konstansok valamelyike az eredmény:

MACTIVE	—	A megadott elem az aktív elem
MACTMARKED	—	A megadott elem egy kijelölt elem, és egyben ez az aktív elem is
MMARKED	—	Az adott elem egy kijelölt elem a menüben
MHIDDEN	—	Az adott elem egy rejtett elem a menüben

- | | | |
|----------------|---|--|
| MHIDDEN | — | Az adott elem egy rejtett elem a menüben |
| MNORMAL | — | Az elem nem rejtett, nincs is kijelölve, és nem is aktuális elem. Ez a normális elem |

A "kiválasztó" és a "válogató" menüket a következő két konstans segítségével lehet megkülönböztetni:

- | | | |
|-----------------|---|-------------------|
| M_CHOOSE | — | "kiválasztó" menü |
| M_SELECT | — | "válogató" menü |

A most következő konstans az általános menük definiálásakor a menüelemek magasságának és szélességének megadására használható. Ezzel a konstanssal a menüelem pontosan olyan méretű lesz, mint amekkora helyet az ablakban elfoglal.

- | | | |
|-----------------|---|---------------------------------------|
| DEF_SIZE | — | A menüelem méretét meg kell határozni |
|-----------------|---|---------------------------------------|

13. fejezet

Típusdefiníciók

Mint minden egyéb, a ZMENU által felhasznált típusok definíciói is a ZMENU.H file-ban találhatók. Most ezek ismertetése következik.

MNUM	— A menük azonosítására szolgáló kód. Egy egész szám.
CODE	— Egyes tevékenységek megadásánál a megadott érték egyes bitjeire van szükség. Az ilyen értékek általában CODE típusúak.
VALUE	— A bevezetőben említett menüelemekhez tartozó változó ilyen típusú. Ez is egy szám.
INUM	— Menüelem sorszáma a menüben
HITEM	— Vízszintes menük definiálásakor használatos struktúra. A menüt ilyen elemek tömbjeként lehet megadni. Az elemek két részből állnak. Az egyik egy mutató egy menüelem nevére, mint karakter sorozatra, a másik pedig egy szám, ami a menüelem helyét adja meg (oszlopszám).
VITEM	— Függőleges menük definiálásakor ilyen elemek tömbjét kell megadni. Ez egy karakter sorozatra mutató pointer.
BITEM	— Téglalap alakú (box) menük létrehozásakor ilyen elemek tömbje használható. Ez is egy mutató egy karakter sorozatra.

MITEM

- Általános menük létrehozásakor használható elem. A következő elemekből áll:
`item_str` — a menüelem neve, amit be kell írni az ablakba. Ha több sorból áll, akkor a sorok végét '\n' jelzi.
`item_row` — megadja, hogy a menüelem az ablak mely sorában legyen
`item_col` — megadja, hogy menüelem az ablak mely oszlopában legyen
`item_height` — a menüelem magassága
`item_width` — a menüelem szélessége
`item_lpos` — megadja, hogy a menüelemnek hanyadik betűje az "aktív" betű. Ezzel a karakterrel lehet az elemet kiválasztani.

MACTION

- Ilyen típusú értéket ad vissza a felhasználó által írt olyan eljárás, ami az elemek kiválasztásakor aktivizálódik. Ez adja meg, hogy ki kell-e lépni a menüből, vagy sem.

PFM

- Pointer egy void függvényre, ami egy MNUM típusú paramétert kap. Ilyen függvények adhatók meg a menü megnyitásához és lezárásához.

Pl: void my_begin(MNUM mnum);

PFMI

- Pointer egy void függvényre, ami egy MNUM és egy INUM típusú paramétert kap. Ilyen függvények használhatók a menüelemekre való rálépés és a róluk való lelépés során.

Pl: void my_to(MNUM mnum, INUM inum);

CFMI

- Egy függvényre mutató pointer, ami **MACTION** típusú értéket ad vissza. A függvény három paramétert kap. Rendre **MNUM**, **INUM** és **KEY** típusúakat. Ezek azok a függvények, amelyek a menüelemek kiválasztásakor hívódnak meg.

PI: **MACTION** my_call(**MNUM** mnum, **INUM** inum, **KEY** key);

PFS

- Egy **BOOL** függvényre mutató pointer. Ennek a függvénynek több paramétere is van. A következő típusúak kell hogy legyenek: **MNUM**, **INUM**, **ADDR***, **POS***, **POS***, **LEN***, **LEN***, **POS***. Ez egyébként ritkán használatos függvény, a legelső szintű menüt létrehozó eljárásnak kell megadni.

14. fejezet

Globális változók

A menük kezeléséhez jónéhány globális változó tartozik. Ez azért célszerű, mert ha a programban sok az egyforma menü (szín, ablakkeret stb.), akkor nem kell mindegyiket külön-külön paraméterezni, hanem elég csak az eltérőket.

A most következő globális változók a menüelemek színeit adják meg. Valamennyi ATTR típusú.

NORM_ATTR	—	normál menüelem színe
ACT_ATTR	—	az aktuális menüelem színe
ACT_MARKED_ATTR	—	aktív elem színe, ha ki is van jelölve
HIDDEN_ATTR	—	rejtett menüelem színe
MARKED_ATTR	—	kijelölt menüelem színe
LETTER_ATTR	—	az aktív betű színe

A következő változók a menük ablakára vonatkozó alapfeltételezéseket adják meg. Az ablak, a keret és a cím a színét adják meg. Valamennyi ATTR típusú.

BORDER_ATTR	—	a keret színe
ACT_BORDER_ATTR	—	a keret színe, ha aktív az ablak
TITLE_ATTR	—	az ablak címének a színe
ACT_TITLE_ATTR	—	az ablak címének színe aktív esetben
WINDOW_ATTR	—	az ablak tartalmának színe

Az alábbi két globális változó a menük ablakának keretét adja meg abban az esetben, ha ez az ablak az aktuális, és akkor, ha nem. Mindkettő ADDR típusú.

M_BORDER	— az ablak kerete, ha nem aktív
M_ACT_BORDER	— az ablak kerete, ha aktív

A legtöbb menüben ugyanazokkal a billentyűkkel lehet kiválasztani az elemeket, és ugyanazzal a billentyűvel lehet kilépni a menüből. Két globális változó adja meg, hogy melyek ezek a billentyűk. Mindketten MACRO típusúak.

M_CALL_KEYS	— a menüelemek kiválasztására alkalmas billentyűk tömbje, amelynek utolsó eleme 0. Ennek default értéke úgy van beállítva, hogy csak az ENTER-t tartalmazza.
M_ESC_KEYS	— a menüből ezekkel a billentyűkkel lehet kilépni. Kezdetben ez csak az ESC billentyű lehet.

15. fejezet

Eljárások, függvények

Rátérünk a ZMENU eljárásainak ismertetésére. Az egyes szakaszok az egy funkciócsoportba tartozó eljárásokat ismertetik, ezen belül névsor szerint következnek.

15.1. Inicializálás/lezárás

Ebben a szakaszban a menükezelés során a kezdő és a lezáró eljárás ismertetése következik. Az egyik eljárás állítja be a megfelelő globális változókat, és foglalja le a puffereket, a másik pedig ezen pufferek felszabadítását végzi el.

zm_begin()

Név: zm_begin() – a menükezelés inicializálása

Használat: #include <zkey.h>
#include <zwindow.h>
#include <zmenu.h>
BOOL zm_begin(LEN maxmenu);

Leírás: A menükezelés inicializálása. Paraméterben adható meg, hogy egyszerre hány menüt akar a program nyitva tartani. Ennek alapján lefoglalja a menükezelő eljárások számára a megfelelő puffereket. Néhány globális változó default értékét is beállítja.

Visszatérés: TRUE, ha sikerült az inicializálás. FALSE, ha nem volt elég szabad memória a pufferek lefoglalásához.

Lásd még: `zm_end()`

Példa:

```
ret = zm_begin(8);
if( !ret ) exit(1);
```

`zm_end()`

Név: `zm_end()` – a menükezelés befejezése

Használat:

```
#include <zkey.h>
#include <zwindow.h>
#include <zmenu.h>
void zm_end(void);
```

Leírás: A menükezelés befejezése. A használt puffereket felszabadítja. A menükhöz tartozó ablakokat is lezárja.

Visszatérés: —

Lásd még: `zm_begin()`

Példa:

```
ret = zm_begin(8);
if( !ret )
    exit(1);
...
zm_end();
```


15.2. Menü létrehozása

Ebben a szakaszban a menük létrehozásával kapcsolatos eljárásokat tárgyaljuk. Látni fogjuk, hogy vannak egészen egyszerűen használhatók, de vannak igen bonyolultak is. Mind a két fajtának megvannak a maga előnyei.

bset_menu()

Név: bset_menu() – box jellegű menü létrehozása default értékkel

Használat: #include <zkey.h>
#include <zwindow.h>
#include <zmenu.h>
RET_VALUE bset_menu(POS upr, POS upc, LEN maxh,
LEN width, LEN iwidth, LEN cols,
CODE mtype, BITEMS bitems);

Leírás: Téglalap alakú (box) menük létrehozására szolgál. Megadható a menü ablakának helye a képernyőn (upr, upc), az ablak maximális magassága (maxh) és szélessége (width), a menübe kerülő elem szélessége (iwidth), valamint az, hogy hány elem kerüljön egy sorba (cols). Az mtype paraméter adja meg a menu típusát (M_SELECT, M_CHOOSE), a bitems pedig a menüelemek tömbjére mutat. A tömb utolsó eleme kötelezően NULL.

Visszatérés: A létrehozott menü azonosítószáma, ha negatív, akkor hibakód.

Lásd még: bxset_menu() hset_menu() vset_menu()

Példa: BITEM items[]={"egy", "ketto", "harom", NULL};
mnum = bset_menu(0,0,5,40,20,2,M_SELECT,items);
go_menu(ACTIVE_MENU,mnum,REMOVE_MENU);

build_menu()

Név: build_menu() – menü felépítése

Használat: #include <zkey.h>
#include <zwindow.h>
#include <zmenu.h>
RET_VALUE build_menu(WNUM wnum, LEN max, CODE mtype);

Leírás: Ezzel az eljárással leolvasható egy ablak tartalma, és abból menü készíthető. Minden olyan szöveget, ami nem az ablak saját színével van kiírva, menüelemnek tekint, de legfeljebb max darabot. Mivel az eljárás a menüelemeknek olyan helyet foglal le, amelyhez nem lehet eljárással hozzáférni, ezért ezt fel sem lehet szabadítani. (Az ilyen helyek felszabadítása céljából készült a free_menu() eljárás. Ahhoz azonban, hogy ezt használhassuk, a menü teljes lezárása (remove) előtt meg kell hívni. Tehát a go_menu() eljárást nem szabad REMOVE_MENU paraméterrel meghívni!)

Visszatérés: A definiált menü száma, ha negatív, akkor egy hibakód.

Lásd még: menu_setup() free_menu()

Példa:

```
wprint(wnum,"Ez \1\1elem\2 lesz \1\1itt\2\n"  
      "\1\1Meg\2 egy elem.",DEF_ATTR);  
mnum=build_menu(wnum,10, M_SELECT);  
go_menu(ACTIVE_MENU,mnum,CLOSE_MENU);  
free_menu(mnum);  
remove_menu(mnum);
```

bxset_menu()

- Név:** `bxset_menu()` – box jellegű menü létrehozása
- Használat:** `#include <zkey.h>`
`#include <zwindow.h>`
`#include <zmenu.h>`
`RET_VALUE bxset_menu(POS upr, POS upc,`
`LEN maxh, LEN width, LEN iwidth, LEN cols,`
`ADDR title, ADDR border, ADDR act_border,`
`CODE mtype, BITEMS bitem3);`
- Leírás:** A `bset_menu()` eljáráshoz hasonlóan működik. Megadható a menü fejléce és kerete is.
- Visszatérés:** A létrehozott menü azonosító száma. Ha negatív, akkor hibakód.
- Lásd még:** `bset_menu()`
- Példa:** `char *items[]={ "egy", "ket", "har", NULL};`
`mnum = bxset_menu(10,10,5,40,15,2,"cim",`
`BORDER1,BORDER2,M_SELECT,items);`
`go_menu(ACTIVE_MENU,mnum,CLOSE_MENU);`

hset_menu()

- Név:** `hset_menu()` – vízszintes menü létrehozása
- Használat:** `#include <zkey.h>`
`#include <zwindow.h>`
`#include <zmenu.h>`
`RET_VALUE hset_menu(POS upr, POS upc, LEN width,`
`CODE mtype, HITEMS hitem3);`

Leírás: Egyszerű vízszintes menük létrehozására szolgál. Megadható a menü ablakának helye a képernyőn (upr, upc), a menü szélessége (width), és a menü típusa (M_SELECT, M_CHOOSE). A hitem paraméter egy HITEM típusú elemekből álló tömbre kell hogy mutasson, ahol fel vannak sorolva a menü elemei, és az, hogy hányadik oszlopban lesznek az ablakban. Az utolsó utáni elemnek olyannak kell lennie, amelynek az item_str része a NULL pointert tartalmazza.

Visszatérés: A létrehozott menü azonosító száma. Ha negatív, akkor hibakód.

Lásd még: hxset_menu()

Példa:

```
HITEM hitems[]={ {"Egy",1}, {"Ketto",21},
                  {"Harom",41},{NULL,0}};
mnum = hset_menu(0,0,80,M_SELECT,hitems);
go_menu(ACTIVE_MENU,mnum,REMOVE_MENU);
```

hxset_menu()

Név: hxset_menu() – vízszintes menü létrehozása

Használat:

```
#include <zkey.h>
#include <zwindow.h>
#include <zmenu.h>
RET_VALUE hxset_menu(POS row,POS col,
                     LEN width, ADDR title,
                     ADDR border, ADDR act_border,
                     CODE mtype, MITEMS items);
```

Leírás: A hset_menu() eljáráshoz hasonló, de megadható a menü ablakának címe és kerete is.

Visszatérés: A létrehozott menü azonosító száma. Ha negatív, akkor hibakód.

Lásd még: hset_menu()

Példa:

```
HITEM hitems[]={ {"Egyik",1}, {"Masik",11},
                  {"Utolso elem",31},{NULL,0}};
mnum = hxset_menu(10,10,40," title ",
                 BORDERO,BORDERO,M_SELECT,items);
go_menu(ACTIVE_MENU,mnum,CLOSE_MENU);
```

menu_setup()

Név: menu_setup() – menü létrehozása

Használat: #include <zkey.h>
#include <zwindow.h>
#include <zmenu.h>
RET_VALUE menu_setup(LEN item_num, PFS setup_fv);

Leírás: Menük létrehozására ez a legalacsonyabb szintű eljárás. Meg kell adni, hogy maximum hány eleme lesz a menünek, valamint egy függvényt, amellyel megadhatók a menüelemek adatai. Ezt a függvényt a menüt létrehozó eljárás a következőképpen hívja meg:

```
MNUM setup_fv(mnum,inum,&itemstr,  
               &row,&col,&height,  
               &width,&letter_pos);
```

A setup_fv megkapja, hogy mely menü melyik elemét kell visszaadnia a szintén paraméterként megadott pointerekben. Vissztérési : a menüelem stringjére mutató pointer (itemstr), a menüelem helye az ablakban (row, col), a menüelem mérete (height, width), és a menüelem aktív betűjének pozíciója a menüelemen belül (letter_pos). Erre az a megszorítás, hogy a program csak a menüelem első sorában levő aktív betűket képes kezelni. A TRUE értéket kell visszaadnia, ha kitöltötte a menüelem adatait, FALSE-t, ha nem. Ez azt jelenti, hogy nincs több menüelem. Lehet tehát a menünek kevesebb eleme is, mint item_num.

Az eljárás sok mindent beállít, amit a menü meghívása előtt az mchange....() és mset....() eljárásokkal lehet megváltoztatni. Amit mindenképpen meg kell hívni, az az mchange_pos() eljárás, mert az ablak méretére nincs feltevés.

Visszatérés: A létrehozott menü azonosító száma. Ha negatív, akkor hibakód.

Lásd még: vset_menu() hset_menu() bset_menu()

Példa: a ZMENU.C példaprogramban található.

mset_menu()

Név: mset_menu() – általános menü létrehozása default értékkel

Használat: #include <zkey.h>
#include <zwindow.h>
#include <zmenu.h>
RET_VALUE mset_menu(POS upr, POS upc,
LEN height, LEN width,
CODE mtype, MITEM *mitems);

Leírás: Ezzel az eljárással lehet teljesen általános menüket létrehozni. Meg kell adni a menü ablakának a helyét a képernyőn, az ablak magasságát és szélességét, valamint a menü típusát. A mitems mutatót MITEM elemek tömbjére kell beállítani. Ennek utolsó eleme után egy olyan struktúrát kell tenni, aminek az item_str eleme a NULL pointert tartalmazza.

A menüelem szélessége és magassága helyett megadható az előre definiált DEF_SIZE konstans is. Ekkor a program számolja ki, hogy az adott elem milyen magas és milyen széles helyre fér el.

Visszatérés: A menü azonosító száma, különben egy hibakód.

Lásd még: mxset_menu() vset_menu()

Példa: MITEM mitems[]={
{"Egyik\nelem",0,0,DEF_SIZE,DEF_SIZE,0},
{"\nMasik elem",0,30,3,15,1},
{NULL,0,0,0,0,0}};
mnum = mset_menu(2,2,7,50,mitems);

mxset_menu()

Név: mxset_menu() – általános menü létrehozása

Használat: #include <zkey.h>
#include <zwindow.h>
#include <zmenu.h>
RET_VALUE mxset_menu(POS upr, POS upc,
LEN height, LEN width,
ADDR title, ADDR border, ADDR act_border,
CODE mtype, MITEM *mitems);

Leírás: Az mset_menu() eljáráshoz hasonló. Az a többlet, hogy megadható a menü ablakának címe és kerete is.

Visszatérés: A menü azonosító száma, különben hibakód.

Lásd még: mset_menu()

Példa: MITEM mitems[]={
{"Egyik\nelem",0,0,DEF_SIZE,DEF_SIZE,0},
{"\nMasik elem",0,30,3,15,1},
{NULL,0,0,0,0,0}};
mnum = mxset_menu(2,2,7,50,"title",
BORDER1,BORDER3,M_SELECT,mitems);

vset_menu()

Név: vset_menu() – függőleges menü létrehozása

Használat: #include <zkey.h>
#include <zwindow.h>
#include <zmenu.h>
RET_VALUE vset_menu(POS row, POS col, LEN maxh,
CODE mtype, VITEMS vitems);

Leírás: Egyszerű függőleges menük létrehozására szolgál. Megadható a menü ablakának a helye a képernyőn, a menü maximális magassága, a menü típusa (M_SELECT, M_CHOOSE), és a menüelemek tömbjére mutató pointer. A tömb utolsó eleme a NULL pointer kell legyen.

Visszatérés: A létrehozott menü azonosító száma. Ha negatív, akkor hibakód.

Lásd még: `vxset_menu()` `hset_menu()` `bset_menu()`

Példa:

```
char *items[]={"egy","ketto","harom",NULL};
mnum = vset_menu(10,10,5,M_CHOOSE,items);
go_menu(ACTIVE_MENU,mnum,REMOVE_MENU);
```

`vxset_menu()`

Név: `vxset_menu()` – függőleges menü létrehozása

Használat: `#include <zkey.h>`
`#include <zwindow.h>`
`#include <zmenu.h>`
`RET_VALUE vxset_menu(POS row, POS col, LEN maxh,`
`ADDR title, ADDR border, ADDR act_border,`
`CODE mtype, MITEMS items);`

Leírás: A `vset_menu()` eljárashoz hasonló. Megadható még a menü fejléce és kerete is.

Visszatérés: Ha nem negatív, akkor a létrehozott menü azonosító száma. Ha negatív, akkor hibakód.

Lásd még: `vset_menu()`

Példa:

```
char *items[]={"egy","ketto","harom",NULL};
mnum = vxset_menu(10,10,5,"title",
    BORDER1,BORDER2,M_SELECT,items);
go_menu(ACTIVE_MENU,mnum,CLOSE_MENU);
```


15.3. Menü megnyitás, lezárás, meghívás

Ebben a szakaszban azok az eljárások szerepelnek, amelyek a már korábban létrehozott menüket meg tudják nyitni, és azokban lehetővé teszik a válogatást stb.

`active_menu()`

Név: `active_menu()` – menü aktív megnyitása

Használat: `#include <zkey.h>`
`#include <zwindow.h>`
`#include <zmenu.h>`
`RET_VALUE active_menu(MNUM mcode);`

Leírás: Megnyitja az adott menühöz tartozó ablakot, és egyúttal aktívvá is teszi. Az ablakba kiírja a menüelemeket. Meghívja továbbá a menühöz tartozó `begin()` függvényt, amelyet a menü minden megnyitásakor meg kell hívni.

Visszatérés: GOOD, vagy negatív hibakód.

Lásd még: `open_menu()`

Példa: `active_menu(mnum);`
`...`
`call_menu(mnum);`
`readkey();`

call_menu()

Név: call_menu() – menü hívása

Használat: #include <zkey.h>
#include <zwindow.h>
#include <zmenu.h>
RET_VALUE call_menu(MNUM mcode);

Leírás: Már megnyitott menü-ablakban lehet válogatni, a menü függvényei meghívódnak, egészen a kilépésig. (Tulajdonképpen a go_menu() is ezt az eljárást hívja meg a menü ablakának létrehozása után.) Akkor alkalmazható, amikor valamilyen okból nem akarjuk, hogy a menü az aktuális ablakban legyen.

Igen fontos tulajdonsága, hogy azt a billentyűt, amivel kiléptek a menüből, visszateszi a program belső billentyűpufferébe. Ezt a program használhatja, de mindenképpen ki kell onnan olvasnia. Ez felhasználható arra, hogy egymásba ágyazott menük esetén a legalsó szintű menüben megnyomott ESC akár négy menüből is kiléptessen. Mivel nem kell kiolvasni az ESC-et a megfelelő menük meghívása után, mert így az egy szinttel feljebb levő menü ezt olvassa következőnek, és abból is kilép, miközben újra beteszi a billentyűt a pufferbe.

Visszatérés: Annak az elemnek a sorszáma, ami a kilépés pillanatában az aktuális volt. Ha negatív, akkor hibakód.

Lásd még: open_menu() active_menu() go_menu()

Példa: open_menu(mnum);
call_menu(mnum);
readkey(); /* a bill. kiolvasása */

call_menu_key()

Név: call_menu_key() – menühívás billentyű alapján

Használat: `#include <zkey.h>`
`#include <zwindow.h>`
`#include <zmenu.h>`
`RET_VALUE call_menu_key(MNUM mcode, KEY key);`

Leírás: Az adott menüben azt a tevékenységet hajtja végre, ami a menüben való válogatás során történne, ha ez a billentyű lenne leütve.

Jól használható például arra, hogy egyszerre több menüben lehessen válogatni. Legyen az egyik menü vízszintes, a másik pedig függőleges. Ha a balra vagy jobbra nyilakkal a vízszintes menüben lehet mozogni, a fel és le nyilakkal pedig függőleges menüben. Ilyet lehet látni például a PathMinder, vagy a Quick-Dos programokban is.

Az eljárás meghívása után azt a billentyűt hagyja a billentyűzet pufferében, amivel kiléptek a menüből. Ez vagy az ESC_KEYS-ek valamelyike, vagy a CALL_KEYS-ek közül való. Ez utóbbi akkor lehet, ha a felhasználó által megadott függvény az MEXIT értékkel jelezte, hogy ki kell lépni a menüből.

Visszatérés: GOOD, vagy negatív hibakód.

Lásd még: call_menu()

Példa: `call_menu_key(mnum, LEFT);`
`call_menu_key(mnum2, UP);`

close_menu()

Név: close_menu() – menü ablakának lezárása

Használat: #include <zkey.h>
#include <zwindow.h>
#include <zmenu.h>
RET_VALUE close_menu(MNUM mcode);

Leírás: A menühöz tartozó ablakot lezárja, de a menü maga nem változik, tehát újra használható. A menüben minden megmarad, például a kijelölt elemek, az aktuális elem stb.

Visszatérés: GOOD, vagy negatív hibakód.

Lásd még: hide_menu() remove_menu()

Példa:
open_menu(mnum);
call_menu(mnum);
readkey();
close_menu(mnum);
...
active_menu(mnum);
remove_menu(mnum);

free_menu()

Név: free_menu() – menüelemek felszabadítása

Használat: #include <zkey.h>
#include <zwindow.h>
#include <zmenu.h>
RET_VALUE free_menu(MNUM mcode);

Leírás: Csak és kizárólag a build_menu() által létrehozott menükre használható. A menüelemek tárolásához lefoglalt területet szabadítja fel. Arra is vigyázni kell, hogy nem szabad előtte meghívni a remove_menu() eljárást, mert az törli a menüt, és ezáltal elérhetlenné válnak az elemei. (Ekkor persze fel sem lehet szabadítani a helyüket.)

Visszatérés: GOOD, vagy negatív hibakód.

Lásd még: build_menu() remove_menu()

Példa:

```
mnum = build_menu(wnum,20);  
go_menu(ACTIVE_MENU,mnum,CLOSE_MENU);  
readkey();  
...  
free_menu(mnum);
```

go_menu()

Név: go_menu() – menü végrehajtása

Használat:

```
#include <zkey.h>  
#include <zwindow.h>  
#include <zmenu.h>  
RET_VALUE go_menu(CODE begin, MNUM mcode, CODE  
end);
```

Leírás: Az adott menühöz tartozó ablakot nyitja meg és meghívja a menüt, majd bezárja az ablakot. A begin paraméter adja meg, hogyan kell az ablakot megnyitni (ACTIVE_MENU, OPEN_MENU), az end paraméter pedig azt, hogyan kell lezárni (KEEP_MENU, HIDE_MENU, CLOSE_MENU, REMOVE_MENU). A nyitás és a zárás között lehet a menüben válogatni, és ezzel a menühöz tartozó megfelelő függvények meghívhatók. A menüből végül a megadott kilépő billentyűk lenyomásával lehet kilépni. A call_menu() eljárásnál leírtak itt is érvényesek. A legfontosabb közülük, hogy itt is marad billentyűkód a pufferben.

Visszatérés: A menüből való kilépés pillanatában aktuális menüelem sorszáma, ha negatív, hibakód.

Lásd még: open_menu() active_menu() call_menu()

Példa:

```
go_menu(ACTIVE_MENU,mnum,HIDE_MENU);  
readkey(); /* a bill. kiolvasása */
```

hide_menu()

Név: `hide_menu()` – menü ablakának elrejtése

Használat: `#include <zkey.h>`
`#include <zwindow.h>`
`#include <zmenu.h>`
`RET_VALUE hide_menu(MNUM mcode);`

Leírás: A menühöz tartozó ablakot láthatatlanná teszi, a menü maga pedig nem változik. Így tehát a menü a későbbiekben használható. Ha a menü ablakát nem zárjuk le, akkor helyet foglal a memóriában, de menü meghívásakor gyorsabb az ablak megjelenítése, mert nem kell a menükezelőnek létrehoznia egy új ablakot, és abba kiírnia az összes menüelemet.

Visszatérés: GOOD, vagy negatív hibakód.

Lásd még: `open_menu()` `remove_menu()` `close_menu()`

Példa: `open_menu(mnum);`
`call_menu(mnum);`
`hide_menu(mnum);`
`...`
`open_menu(mnum);`
`remove_menu(mnum);`

open_menu()

Név: `open_menu()` – menü megnyitása

Használat: `#include <zkey.h>`
`#include <zwindow.h>`
`#include <zmenu.h>`
`RET_VALUE open_menu(MNUM mcode);`

Leírás: Megnyitja az adott menühöz tartozó ablakot, és kiírja a menüelemeket. Az ablakot azonban nem teszi aktuálissá. Meghívja egyúttal azt a függvényt, ami a menü megnyitáshoz szükséges.

Visszatérés: GOOD, vagy negatív hibakód.

Lásd még: `active_menu()`

Példa:
`open_menu(mnum);`
`call_menu(mnum);`
`readkey();`

`remove_menu()`

Név: `remove_menu()` – menü törlése

Használat: `#include <zkey.h>`
`#include <zwindow.h>`
`#include <zmenu.h>`
`RET_VALUE remove_menu(MNUM mcode);`

Leírás: Teljesen törli a megadott menüt. A menü ablaka bezáródik és a menühöz tartozó valamennyi puffer felszabadul. A menü ezek után nem használható.

Visszatérés: GOOD, vagy negatív hibakód.

Lásd még: `open_menu()` `go_menu()`

Példa:
`open_menu(mnum);`
`call_menu(mnum);`
`readkey();`
`remove_menu(mnum);`

15.4. Menü adatainak lekérdezése

Most azok az eljárások következnek amelyek segítségével le lehet kérdezni, hogy egy menüben mely elemek vannak kijelölve, egy adott elem milyen állapotú (rejtett, aktív, kijelölt), egy adott menüben, egy adott elemhez milyen szöveg tartozik (ez látható az ablakban), valamint a bevezetőben említett — a menüelemhez tartozó — változó értékét.

`mget_first_marked()`

Név: `mget_first_marked()` – első megjelölt menüelem lekérdezése

Használat: `#include <zkey.h>`
`#include <zwindow.h>`
`#include <zmenu.h>`
`RET_VALUE mget_first_marked(MNUM mnum);`

Lefrás: Megadja a menüben a legkisebb sorszámú megjelölt menüelemet, ha van ilyen elem.

Visszatérés: A menüelem sorszáma, ha negatív, akkor hibakód.

Lásd még: `mget_next_marked()`

Példa:

```
inum = mget_first_marked(mnum);  
if( inum >= 0 )  
    inum2 = mget_next_marked();
```

mget_item_state()

Név: mget_item_state() – menüelem állapotának lekérdezése

Használat: #include <zkey.h>
#include <zwindow.h>
#include <zmenu.h>
RET_VALUE mget_item_state(MNUM mnum, INUM inum);

Leírás: A menüelem állapotát adja meg. Ez a következő konstansok valamelyike: MACTIVE, MACTMARKED, MMARKED, MHIDDEN, MNORMAL.

Visszatérés: Az állapot maga, vagy negatív, ha hiba volt.

Lásd még: mchange....() mset....()

Példa: state = mget_item_state(mnum, inum);

mget_item_str()

Név: mget_item_str() – menüelem stringjének lekérdezése

Használat: #include <zkey.h>
#include <zwindow.h>
#include <zmenu.h>
ADDR mget_item_str(MNUM mnum, INUM inum);

Leírás: Megadja az adott menü, adott elemének szövegére mutató pointert.

Visszatérés: A pointer, ha volt ilyen menü és ilyen elem, különben NULL.

Lásd még: mchange....() mset....()

Példa: printf("%s", mget_item_str(mnum, inum));

mget_item_value()

Név: `mget_item_value()` – a menüelem értékének lekérdezése

Használat: `#include <zkey.h>`
`#include <zwindow.h>`
`#include <zmenu.h>`
`RET_VALUE mget_item_value(MNUM mnum, INUM inum);`

Leírás: A menüelemek belső változójának lekérdezésére szolgál.

Visszatérés: Az érték maga, vagy negatív, ha hiba volt.

Lásd még: `mchange....()` `mset....()`

Példa: `val = mget_item_value(mnum, inum);`

mget_next_marked()

Név: `mget_next_marked()` – a következő megjelölt elem lekérdezése

Használat: `#include <zkey.h>`
`#include <zwindow.h>`
`#include <zmenu.h>`
`RET_VALUE mget_next_marked(void);`

Leírás: Megadja az utolsó `mget_first_marked()` eljárás hívásánál használt menüben a soron következő megjelölt menüelemet.

Visszatérés: A menüelem sorszáma, ha negatív, akkor hibakód.

Lásd még: `mget_first_marked()`

Példa: `inum = mget_first_marked(mnum);`
`inum2= mget_next_marked();`

15.5. Menüelemek adatainak beállítása

Ebben a szakasban a menüelemek adatait beállító eljárásokat ismertetjük. Az eljárások az elemek kijelölését, rejtettségét és a hozzájuk tartozó változó értékét állítják be.

`mset_act_mark...()`

Név: `mset_act_mark...()` – aktív elem megjelölése

Használat: `#include <zkey.h>`
`#include <zwindow.h>`
`#include <zmenu.h>`
`RET_VALUE mset_act_marked(MNUM mnum);`
`RET_VALUE mset_act_nomarked(MNUM mnum);`
`RET_VALUE mset_act_xmarked(MNUM mnum);`

Leírás: Megjelöli az aktuális menüelemeket, vagy visszaállítja, vagy az ellenkezőjére változtatja.

Visszatérés: GOOD, vagy negatív hibakód.

Lásd még: `mset_item_marked()`

Példa: `mset_act_marked(mnum);`
`mset_act_xmarked(mnum);`

mset_all_mark...()

Név: mset_all_mark...() – az összes elem megjelölése

Használat: #include <zkey.h>
#include <zwindow.h>
#include <zmenu.h>
RET_VALUE mset_all_marked(MNUM mnum);
RET_VALUE mset_all_nomarked(MNUM mnum);
RET_VALUE mset_all_xmarked(MNUM mnum);

Leírás: Az adott menüben az összes elemet meg lehet jelölni, nem megjelöltté tenni, vagy az elemek megjelöltségét az ellenkezőjére változtatni.

Visszatérés: GOOD, vagy negatív hibakód.

Lásd még: mset_item_marked()

Példa: ret = mset_all_xmarked(mnum);

mset_item_hidden...()

Név: mset_item_hidden...() – a menüelem elrejtése

Használat: #include <zkey.h>
#include <zwindow.h>
#include <zmenu.h>
RET_VALUE mset_item_hidden(MNUM mnum, INUM inum);
RET_VALUE mset_item_nohidden(MNUM mnum, INUM inum);
RET_VALUE mset_item_xhidden(MNUM mnum, INUM inum);

Leírás: Az adott menüelem rejtettségét beállítja, megszünteti megváltoztatja, tehát ami nem volt rejtett rejtett lesz, ill. ami rejtett volt, annak a rejtettsége megszűnik.

Visszatérés: GOOD, vagy negatív hibakód.

Lásd még: mset_item_marked() mset_act_marked()

Példa: ret = mset_item_xhidden(mnum, inum);

mset_item_mark..()

Név: mset_item_mark..() – menüelem megjelölése

Használat: #include <zkey.h>
#include <zwindow.h>
#include <zmenu.h>
RET_VALUE mset_item_marked(MNUM mnum, INUM inum);
RET_VALUE mset_item_nomarked(MNUM mnum, INUM inum);
RET_VALUE mset_item_xmarked(MNUM mnum, INUM inum);

Leírás: A menüelemek megjelöltségét lehet ezekkel az eljárásokkal állítani: megjelöltté tenni, nem megjelöltté tenni, vagy a megjelöltséget ellenkezőjére változtatni.

Visszatérés: GOOD, vagy negatív hibakód.

Lásd még: mset_item_hidden() mset_act_marked()

Példa: ret = mset_item_marked(mnum, inum);

mset_item_value()

Név: mset_item_value() – menüelem értéke

Használat: #include <zkey.h>
#include <zwindow.h>
#include <zmenu.h>
RET_VALUE mset_item_value(MNUM mnum, INUM inum, VALUE value);

Leírás: A menüelemekhez tartozó változókat lehet ezzel az eljárással beállítani. Célszerű ezt nem negatív értékre állítani, mert az érték lekérdezésekor csak így tudjuk megkülönböztetni a hibakódtól.

Visszatérés: GOOD, vagy negatív hibakód.

Lásd még: mget_item_value()

Példa: mset_item_end(mnum, modell_item, "Large");
mset_item_value(mnum, modell_item, 4);

15.6. Menü adatainak változtatása

A menünek és a hozzá tartozó ablaknak több tulajdonságát meg lehet változtatni a program futása közben is. Lehetőség van a menühöz tartozó függvények beállítására, a menüelemek megváltoztatására, és a menühöz tartozó ablak szinte minden adatának a lecserélésére.

mchange....()

Név: mchange....() – a menü ablakának adatai

Használat:

```
#include <zkey.h>
#include <zwindow.h>
#include <zmenu.h>
RET_VALUE mchange_title(MNUM mnum, ADDR title);
RET_VALUE mchange_btitle(MNUM mnum, ADDR btitle);
RET_VALUE mchange_border(MNUM mnum, ADDR border,
                          ADDR act_border);
RET_VALUE mchange_battr(MNUM mnum,
                        ATTR battr, ATTR act_battr);
RET_VALUE mchange_tattr(MNUM mnum,
                        ATTR tattr, ATTR act_tattr);
RET_VALUE mchange_sattr(MNUM mnum, ATTR sattr);
RET_VALUE mchange_attrs(MNUM mnum,
                        ATTR norm_attr, ATTR hidden_attr,
                        ATTR marked_attr, ATTR act_attr,
                        ATTR act_marked_attr,
                        ATTR letter);
```

Leírás: A menühöz tartozó ablak különböző adatait lehet megváltoztatni. Rendre az ablak címét, alsó címét, keretét, keretszínét, címének színét, az ablak belsejének színét, és a menüelemek színét.

Visszatérés: GOOD, vagy negatív hibakód.

Példa: `ret = mchange_title(mnum,"uj title");`
 `ret = mchange_sattr(mnum,BLUE);`

mchange_begin_fv()

Név: `mchange_begin_fv()` – menü kezdő függvénye

Használat: `#include <zkey.h>`
 `#include <zwindow.h>`
 `#include <zmenu.h>`
 `RET_VALUE mchange_begin_fv(MNUM mnum, PFM fv);`

Leírás: Kicseréli a menühöz tartozó kezdő függvényt. Ez akkor hívódik meg, amikor a menü ablak megjelenik a képernyőn. Ekkor például létre lehet hozni egy másik ablakot, amibe a menü help-je kerül majd. A függvény meghívásakor megkapja a menü azonosító számát.

Visszatérés: `GOOD`, vagy negatív hibakód.

Lásd még: `mchange_end_fv()`

Példa: `ret = mchange_begin_fv(mnum,my_begin_fv);`

mchange_call_fv()

Név: `mchange_call_fv()` – a menü hívási függvénye

Használat: `#include <zkey.h>`
 `#include <zwindow.h>`
 `#include <zmenu.h>`
 `RET_VALUE mchange_call_fv(MNUM mnum, CFMI fv);`

Leírás: Ha a menüben való válogatás során a felhasználó a később ismertetendő módon megadott billentyűk közül üti le valamelyiket (`CALL_KEYS`), akkor a menühöz tartozó hívási függvény kerül végrehajtásra. Megkapja a menü számát, a menüelem sorszámát, és a billentyűt is, amellyel meghívták. Ha a függvény az `MEXIT` értékkel tér vissza, akkor a menüből kilép a program, ha `MNOEXIT` értékkel, akkor tovább lehet válogatni a menüben.

Visszatérés: GOOD, vagy negatív hibakód.

Lásd még: `mchange_begin_fv()` `mchange_to_fv()`

Példa: `ret = mchange_call_fv(mnum, my_call_fv);`

`mchange_call_keys()`

Név: `mchange_call_keys()` – a menü hívási billentyűi

Használat: `#include <zkey.h>`
`#include <zwindow.h>`
`#include <zmenu.h>`
`RET_VALUE mchange_call_keys(MNUM mnum, MACRO keys);`

Leírás: Meg lehet adni azokat a billentyűket, amelyek lenyomása esetén a menühöz tartozó hívási függvényt kell meghívni. A `CALL_KEYS` változó default értéke szerint ez csak az `ENTER`.
A billentyűket egy tömbbe kell tenni, amelynek utolsó eleme 0, és az ide mutató pointert kell az eljárásnak megadni.

Visszatérés: GOOD, vagy negatív hibakód.

Lásd még: `mchange_esc_keys()`

Példa: `KEY my_keys[]={F1,F2,F3,ENTER,0};`
`ret = mchange_call_keys(mnum, my_keys);`

`mchange_esc_keys()`

Név: `mchange_esc_keys()` – a menü kilépő billentyűi

Használat: `#include <zkey.h>`
`#include <zwindow.h>`
`#include <zmenu.h>`
`RET_VALUE mchange_esc_keys(MNUM mnum, MACRO keys);`

Leírás: Meg lehet adni azokat a billentyűket, amelyek lenyomása esetén a menüből elemválasztás nélkül lehet kilépni. Az előre definiált `ESC_KEYS` változó egyedül az `ESC` billentyűt adja meg.
A billentyűknek egy tömbbe kell kerülniük, amelynek végét 0 jelzi.

Visszatérés: GOOD, vagy negatív hibakód.

Lásd még: `mchange_call_keys()`

Példa:
`KEY my_esc[]={F1,F2,F3,ENTER,0};`
`ret = mchange_esc_keys(mnum,my_esc);`

`mchange_end_fv()`

Név: `mchange_end_fv()` – menü befejező függvénye

Használat: `#include <zkey.h>`
`#include <zwindow.h>`
`#include <zmenu.h>`
`RET_VALUE mchange_end_fv(MNUM mnum, PFM fv);`

Leírás: Kicseréli a menühöz tartozó befejező függvényt. Ez akkor hívódik meg, amikor a menü ablaka eltűnik a képernyőről.

Ekkor például le lehet törölni a már nem szükséges ablakokat, vagy lezárni a feleslegessé vált file-okat. A függvény meghívási paramétere a menü azonosító száma.

Visszatérés: GOOD, vagy negatív hibakód.

Lásd még: `mchange_begin_fv()`

Példa: `ret = mchange_end_fv(mnum,my_end_fv);`

`mchange_item_end()`

Név: `mchange_item_end()` – a menüelem vége

Használat: `#include <zkey.h>`
`#include <zwindow.h>`
`#include <zmenu.h>`
`RET_VALUE mchange_item_end(MNUM mnum,INUM inum, ADDR end);`

Leírás: Az adott menüelemnek megváltoztatja a végét. Így lehet például az elem végére odaírni az elem állapotával kapcsolatos információkat. Példának igen jó a Turbo C memória modellt kiválasztó menüje. Ennek a végén mindig ott van az aktuális modell. A menüelemek belső változójával együtt igen jól használható.

Visszatérés: GOOD, vagy negatív hibakód.

Lásd még: `mget_item_str()`

Példa: `mchange_item_end(mnum,modell_inum," Large");`

`mchange_from_fv()`

Név: `mchange_from_fv()` – a menü lelépés függvénye

Használat: `#include <zkey.h>`
`#include <zwindow.h>`
`#include <zmenu.h>`
`RET_VALUE mchange_from_fv(MNUM mnum, PFMI fv);`

Leírás: A menühöz tartozó kilépés függvényt cseréli ki. Ez a függvény minden alkalommal meghívódik, amikor a menüben való válogatás során egy menüelemről kilépünk.

A függvény paraméterként megkapja a menü számát és az elem a sorszámát.

Visszatérés: GOOD, vagy negatív hibakód.

Lásd még: `mchange_to_fv()`

Példa: `ret = mchange_from_fv(mnum,futty_fv);`

mchange_item_front()

- Név:** `mchange_item_front()` – a menüelem eleje
- Használat:** `#include <zkey.h>`
`#include <zwindow.h>`
`#include <zmenu.h>`
`RET_VALUE mchange_item_front(MNUM mnum, INUM inum, ADDR front);`
- Leírás:** Az adott menüelemnek megváltoztatja a kezdetét.
Így lehet például az elem elé odatenni egy kis jelet, ami jelzi, hogy ez a menüelem be van állítva. A menüelemek belső változójával együtt igen jól használható.
- Visszatérés:** GOOD, vagy negatív hibakód.
- Lásd még:** `mget_item_str()`
- Példa:** `mchange_item_front(mnum, inum, "+");`

mchange_item_str()

- Név:** `mchange_item_str()` – a menüelem neve
- Használat:** `#include <zkey.h>`
`#include <zwindow.h>`
`#include <zmenu.h>`
`RET_VALUE mchange_item_str(MNUM mnum, INUM inum, ADDR item_str, POS letter_pos);`
- Leírás:** A megadott menüelem stringjét kicseréli, azaz a menüelem más nevet kap. Ez a képenyőn is azonnal látszik. Ezzel együtt is jól használható a menüelemek belső változója.
- Visszatérés:** GOOD, vagy negatív hibakód.
- Lásd még:** `mget_item_str()`
- Példa:** `ret = mchange_item_str(mnum, inum, "Uj szoveg", 0);`

mchange_pos()

Név: mchange_pos() – a menü pozíciója

Használat: #include <zkey.h>
#include <zwindow.h>
#include <zmenu.h>
RET_VALUE mchange_pos(MNUM mnum,
POS upr, POS upc, POS lor, POS loc);

Leírás: Meg lehet változtatni a menü helyét a képernyőn. Igen érdekes lehet például, amint egy menüben való válogatás során maga a menü is elmozdul az eredeti helyéről.

Visszatérés: GOOD, vagy negatív hibakód.

Lásd még: mchange_type() mchange_str()

Példa: ret = mchange_pos(mnum, 2, 2, 10, 30);

mchange_str()

Név: mchange_str() – a menü ablakának tartalma

Használat: #include <zkey.h>
#include <zwindow.h>
#include <zmenu.h>
RET_VALUE mchange_str(MNUM mnum, ADDR str);

Leírás: A menük ablakában a menüelemeken kívül más információk is lehetnek. Ezzel az eljárással lehet beállítani, hogy a menü ablakának a létrehozásakor mit kell kiírni az ablakba az elemek kiírása előtt. Így is megoldható például az olyan menü, amiben a menüelemeken kívül szöveg is van.

Visszatérés: GOOD, vagy negatív hibakód.

Lásd még: mchange_type()

Példa: ret = mchange_str(mnum, window_str);

mchange_to_fv()

Név: mchange_to_fv() – a menü rálépés függvénye

Használat: #include <zkey.h>
#include <zwindow.h>
#include <zmenu.h>
RET_VALUE mchange_to_fv(MNUM mnum, PFMI fv);

Leírás: A menühöz tartozó rálépés függvényt cseréli ki. Ez a függvény minden alkalommal meghívódik, amikor a menüben való válogatás során egy menüelemre rálépünk. A függvény paramétere a menü számát és az elem sorszámát is megkapja.

Jól használható például arra, hogy a menüelemhez tartozó help szöveget kiirassuk.

Visszatérés: GOOD, vagy negatív hibakód.

Lásd még: mchange_from_fv()

Példa: ret = mchange_to_fv(mnum, help_fv);

mchange_type()

Név: mchange_type() – a menü típusa

Használat: #include <zkey.h>
#include <zwindow.h>
#include <zmenu.h>
RET_VALUE mchange_type(MNUM mnum, CODE type);

Leírás: A menü típusát változtatja meg. Egyelőre az M_SELECT és az M_CHOOSE típusok használata lehetséges. Az elsőben ki lehet jelölni több elemet a SPACE, .+, -, * billentyűkkel, míg a másik típusú menüben ez nem lehetséges.

Visszatérés: GOOD, vagy negatív hibakód.

Lásd még: mchange_str()

Példa: ret = mchange_type(mnum, M_SELECT);

15.7. Aktív elem kijelölése

mselect()

- Név:** mselect() – aktív elem kijelölése
- Használat:** #include <zkey.h>
#include <zwindow.h>
#include <zmenu.h>
RET.VALUE mselect(MNUM mnum, INUM inum);
- Leírás:** Az adott menüben kijelöli az új aktív elemet. Ez a megfelelő függvények meghívását is maga után vonja. A régi aktuális elemről az új aktuális elemre lép.
- Visszatérés:** GOOD, vagy negatív hibakód.
- Példa:** ret = mselect(mnum, inum);

15.8. Következő elem meghatározása

A alábbi eljárások akkor használhatók, ha valaki nem akarja a beépített menüket kezelő eljárást használni, hanem maga akar ilyet írni. Ezek az eljárások megmondják, hogy a menüben az egyes billentyűk lenyomására mely elem lenne a következő aktuális.

next_item....()

Név: next_item....() – következő elem

Használat: #include <zkey.h>
#include <zwindow.h>
#include <zmenu.h>
RET_VALUE next_item_home(MNUM mnum);
RET_VALUE next_item_end(MNUM mnum);
RET_VALUE next_item_left(MNUM mnum);
RET_VALUE next_item_right(MNUM mnum);
RET_VALUE next_item_up(MNUM mnum);
RET_VALUE next_item_down(MNUM mnum);
RET_VALUE next_item_pgup(MNUM mnum);
RET_VALUE next_item_pgdn(MNUM mnum);
RET_VALUE next_item_char(MNUM mnum, KEY key);

Leírás: Megadja, hogy az adott menüben mi lenne a következő aktuális elem, ha az eljárás nevében szereplő billentyűt nyomnák le. Ha valaki saját maga akarja a menüket kezelni (mert például egyszerre két menüben kell mozognia), akkor ezeket, és az mselect() eljárást használhatja.

Ezeknek az eljárásoknak a segítségével valósítható meg, hogy az előre definiált kurzormozgató billentyűk helyett másokat használjunk. Erre is akkor lehet szükség, ha egyszerre több menüben lehet válogatni. Az egyikben, mondjuk a nyilakkal lehet mozogni, a másiban a funkció billentyűkkel.

Visszatérés: A következő elem sorszáma, vagy negatív, ha valami hiba volt.

Lásd még: `mchange_type()`

Példa:

```
if( key==UP )
    mselect(mnum,next_item_up(mnum));
if( key=='8') /* SHIFT+UP */
    mselect(mnum2,next_item_up(mnum2));
```


16. fejezet

Adatbevitel

16.1. Bevezetés

ZINPUT

- A ZINPUT program lehetővé teszi, hogy bármilyen adatbeviteli lehetőséget kihasználjunk. Mivel lehet az a legmegfelelőbb adatbeviteli mód, amikor is a cím megadható, illetve a program is nem foglalkozik az adat bevitelével, hanem csak az adatot fogadja, és a program a megfelelő adatot adja ki. Az adatbeviteli módok közül az egyik az, hogy az adatbevitelt a programozó maga kezeli, az adatot a programozó adja meg, és vannak olyan módok is, például a szöveges adat bevitel, ahol az adatot a programozó adja meg.

Adatbeviteli eljárások

- Több adatbeviteli mód van, és ezek közül az egyik az, hogy az adatbevitelt a programozó maga kezeli, az adatot a programozó adja meg, és vannak olyan módok is, például a szöveges adat bevitel, ahol az adatot a programozó adja meg.
- Vannak azonban más lehetőségek is. Ezek már nem a ZINPUT program, hanem a programozó maga kezeli az adatbevitelt, és a programozó maga adja meg az adatot. Ezek a lehetőségek az adatbeviteli módok, és ezek közül az egyik az, hogy az adatbevitelt a programozó maga kezeli, az adatot a programozó adja meg, és vannak olyan módok is, például a szöveges adat bevitel, ahol az adatot a programozó adja meg.

16. fejezet

Adatbevitel

16.1. Bevezetés

- A ZINPUT program legfontosabb szolgáltatása, hogy kényelmes adatbeviteli lehetőséget biztosít. Megvalósítható segítségével az ellenőrzött adatbevitel, amikor is a nem megfelelő értéket a program el sem fogadja. Az input mezőben pedig — ha ez lehetséges — arra a pozícióra állítja a kurzort, ahol az érvénytelen adat szerepel. Az ellenőrző eljárásokat a programozó maga készítheti el saját igényei szerint, de vannak előre megírtak is, például a személyi szám, dátum, stb ellenőrzésére.
- Több inputmezőből összeállítható egy ún. inputtábla. Ez azt jelenti, hogy először kijelöljük azokat a mezőket, ahova adatokat kell a felhasználónak bevennie, mindegyikhez megadjuk a megfelelő paramétereket (formátum, ellenőrző függvény, stb), majd ha ezzel készen vagyunk, a megfelelő eljárás meghívásával lehetővé tesszük az adatbavitelt. A DBASE ismerőinek ez talán ismerős egy kicsit.
- Vannak azonban más szolgáltatások is. Akik már ismerik a ZMENU programot, nem fognak meglepődni, hogy az inputtáblában való lépkedés során meghívódnak olyan függvények, amelyeket a felhasználó adott meg. Például az a függvény, amit akkor hívunk meg, ha a tábla egy eleméről lelépés történik. Ennek párja természetesen egy másik, amit az elemre való rálépéskor kell hívni. Ezek a függvények például arra használhatók, hogy a szerkesztett mezőről információt adjanak.

- Megadható egy függvény az inputtábla megnyitásához, valamint definiálható egy olyan függvény is, amely akkor hívódik meg, amikor a felhasználó az inputtáblából akar kilépni. Ez a függvény határozhatja meg, hogy ki lehet-e lépni, vagy folytatni kell-e a szerkesztést. A választ például az egyes mezők közötti összefüggések ismeretében határozhatjuk meg.
- A legfontosabb lehetőség azonban az, hogy a megfelelő eljárások használatával lehetőségünk van az inputtábla menet közbeni változtatására. Az inputtáblában egyes mezőket nem kiválaszthatóvá tudunk tenni (természetesen ezt később vissza is lehet állítani). Arra is van lehetőség, hogy egy mezőt teljes egészében kitöröljünk, de újakat is felvehetünk.
- Hosszabb szövegek bevitelére egy egyszerű, de jól használható editort tartalmaz, amivel egy ablak tartalma full screen módon kényelmesen módosítható.

16.2. A ZINPUT lemez tartalma

README.ZI	—	A legújabb információk
ZINPUT.H	—	A ZINPUT header file-ja
ZI_TC.S.LIB	—	Turbo C könyvtár — small
ZI_TC.L.LIB	—	Turbo C könyvtár — large
ZI_MSC.S.LIB	—	Microsoft C könyvtár — small
ZI_MSC.M.LIB	—	Microsoft C könyvtár — medium
ZI_MSC.L.LIB	—	Microsoft C könyvtár — large
ZINPUT.C	—	A példaprogram forrásnyelven

16.3. Használat

A ZINPUT program használatához elengedhetetlenül szükséges a ZKEY és ZWINDOW programok megléte, mert igen nagy mértékben ezekre épül. Ennek a két programnak a használatára itt nem térek ki, azok az előző programleírásokban megtalálhatók.

A ZINPUT használatához az elkészítendő programba be kell hívni a ZINPUT.H header file-t (a ZKEY.H és a ZWINDOW.H után). A következő könyvtárak közül pedig a megfelelőt hozzá kell szerkeszteni a programhoz.

ZI.TC.S.LIB	—	Turbo C small modell
ZI.TC.L.LIB	—	Turbo C large modell
ZI.MSC.S.LIB	—	Microsoft C small modell
ZI.MSC.M.LIB	—	Microsoft C medium modell
ZI.MSC.L.LIB	—	Microsoft C large modell

Célszerű a header file-t a C fordító INCLUDE alkönyvtárába tenni, a .LIB kiterjesztésű file-okat pedig a LIB alkönyvtárba. Így használhatók a legegyszerűbben.

17. fejezet

Konstansok

A ZINPUT program által használt konstansok definíciói a ZINPUT.H file-ban találhatók. Most ezek rövid ismertetése következik.

A ZWINDOW programból már ismert RET_VALUE típus további lehetséges értékei következnek. Az eljárások a ZWINDOW (és ZMENU) leírásban felsoroltakon kívül ezeket is visszaadhatják. Ezekkel a konstansokkal tudnak tájékoztatást adni arról, hogy az eljárás rendben lefutott-e. Ha nem, akkor a hiba okára is magyarázatot adnak.

- | | | |
|--------------------|---|--|
| NOMORETABLE | — | Nem lehet több inputtáblát létrehozni, mert már most is annyi van nyitva, amennyit a zi_begin() eljárás hívásával beállítottunk. |
| MEMORYFULL | — | Nincs elég szabad memória az inputtábla létrehozásához. |
| BADTNUM | — | Az eljárás hívásakor rossz inputtábla azonosítót kapott. |
| TNOTOPEN | — | A hivatkozott inputtábla nincs megnyitva. |
| NOMOREGET | — | Az inputtáblába nem lehet több elemet felvenni (get eljárások). |
| BADGNUM | — | Az inputtáblában nincs olyan elem, ami az adott sorszámmal rendelkezne. |

Az a függvény, amely az inputtáblából való kilépés előtt hívódik meg, a következő konstansok valamelyikét kapja. Ebből állapíthatja meg, hogy a kilépéskérés miért történt.

- | | | |
|-----------------|---|--|
| ESC_EXIT | — | Az inputtáblából a felhasználó az ESC billentyűvel akar kilépni. |
|-----------------|---|--|

- ENTER_EXIT** — Az inputtábla utolsó eleménél a felhasználó az ENTER-t nyomta meg.
- C_ENTER_EXIT** — Az inputtáblából a felhasználó az CTRL/ENTER billentyű kombinációval akar kilépni.
- UP_EXIT** — Az inputtábla első elemének szerkesztésekor a felhasználó felfelé akar lépni.
- DOWN_EXIT** — Az inputtábla utolsó elemének szerkesztésekor a felhasználó lefelé akar lépni.

A következő konstansok is az inputtáblából való kilépés előtt hívott függvénnyel kapcsolatosak. Ezek valamelyikét adja vissza a függvény. Ezzel szabályozza, hogy a szerkesztés folytatódjon-e, vagy engedélyezi a kilépést.

- G_CURRENT** — A szerkesztést folytatni kell az inputtáblában, az éppen aktuális elemnél.
- G_FIRST** — A szerkesztést folytatni kell az inputtábla első eleménél.
- G_LAST** — A szerkesztést folytatni kell az inputtábla utolsó eleménél.
- G_EXIT** — Az inputtáblából ki kell lépni, és jelezni kell, hogy az adatok érvényesek.
- G_ESC** — Az inputtáblából ki kell lépni, az adatok érvénytelenségének jelzésével.

A következő konstansok az egyes inputeljárások vezérlésére alkalmasak. Ezek mindegyike egy olyan egész szám, amelyben egy-egy bit van csak egyre állítva. Mindegyik bit egy tulajdonságot képvisel. Az eljárásnak ilyen konstansok összegét kell megadni.

- EDIT** — Az inputmező szerkeszthető.
- SHOW_ONLY** — Az inputmező tartalma nem változtatható, csak meg kell mutatni a felhasználónak.
- LEFT_JUST** — Az inputmező balra igazítva szerkeszthető. (Ettől eltérni csak számok bevitelekor célszerű.)
- RIGHT_JUST** — Az inputmező jobbra igazítva szerkeszthető. (Számok bevitelekor érdemes használni.)

FIRST_CHAR — Ha a mező szerkesztésekor nem speciális az első leütött billentyű, akkor a mező tartalmát törölni kell, mielőtt ez a karakter megjelenne. Ez azt jelenti, hogy ha a mezőbe való belépés után a felhasználó azonnal elkezd beírni egy konkrét értéket, akkor a default érték eltűnik. Ha azonban előbb a kurzor mozgató billentyűket használja, akkor a default értéket tudja szerkeszteni.

A következő konstansok az előre definiált ellenőrző függvényekhez tartozó formátumok megadására szolgálnak (picture).

DATE_PIC — dátum formátum.
TIME_PIC — idő formátum.
PN_PIC — személyi szám formátum.

A következő konstansok az előre definiált ellenőrző függvények paramétereinek megadásához használhatók.

HUN_DATE — magyar dátum (év-hó-nap)
EUR_DATE — európai dátum (nap-hó-év)
USA_DATE — USA dátum (hó-nap-év)
HOURL2 — idő, 12 órás formában (1..12 óra)
HOURL24 — idő, 24 órás formában (0..23 óra)
PN_NUMBER — személyi szám
EMPTY — az inputmező üres is lehet

18. fejezet

Típusdefiníciók

A ZINPUT programban használt típusok definícióit is a ZINPUT.H header file tartalmazza. Most ezek rövid magyarázata következik.

PAR

- Az ellenőrző függvények paramétere ilyen egész típusú. Ez mutatja meg, hogy a függvény számára mely értékek fogadhatók el. A beépített ellenőrző függvények például olyanok, hogy ha ebben a paraméterben az EMPTY bit egyre van állítva, akkor elfogadják azt a mezőt is, ami üres.

TACTION

- Az inputtáblából való kilépés előtt meghívódik a pxopen eljárással megadott tcall_fv függvény. Ez a függvény ellenőrizheti az egyes mezők összefüggéseit, és végül megadhatja, hogy mit kell tenni. Ennek a függvénynek a visszatérési értéke ilyen egész típusú.

TNUM

- Az input tábla azonosító száma. Az inputtáblákat létrehozó eljárások adják meg. A későbbiekben ezzel az azonosítóval lehet rájuk hivatkozni (egész).

GNUM

- Az inputtáblában az elemek sorszáma, amit a get() eljárások adnak meg. Ezzel lehet az elemekre hivatkozni (egész).

- PFP** — Egy függvényre mutató pointer. A függvény egy PAR és egy ADDR típusú paraméteret kap, és POS típusú értéket ad vissza. Az ilyen függvényeket használjuk ellenőrző függvénynek.
PI: POS fv(PAR par, ADDR field);
- PFTG** — Függvényre mutató pointer. A függvény három paraméterrel rendelkezik, amelyek rendre TNUM, GNUM, ADDR típusúak. A függvény pedig nem ad vissza értéket (void). Az ilyen függvényeket az inputtábla elemeihez rendelt rálépő és lelépő függvényként használhatjuk.
PI: void fv(TNUM t, GNUM g, ADDR field);
- PFT** — Egy olyan függvényre mutató pointer, ami egy TNUM típusú paraméterrel rendelkezik, ő maga pedig void. Az ilyen függvény alkalmas arra, hogy egy inputtáblához rendeljük hozzá, mint megnyitáskor meghívandó függvényt.
- PFTR** — Ez egy függvényre mutató pointer. A függvény paraméterei TNUM és RET_VALUE típusúak. Az általa visszaadott érték pedig TACTION típusú. Ilyen függvényeket lehet használni az inputtáblák kilépő függvényének.
PI: TACTION fv(TNUM t, RET_VALUE exit);

19. fejezet

Globális változók

Szintén a ZINPUT.H file-ban vannak definiálva a program által használt globális változók. Ezek közül először azok következnek, amelyek az inputtáblákban az elemek színét határozzák meg abban az esetben, ha nincs megadva azokra konkrét érték. Valamennyi ATTR típusú.

- | | | |
|----------------------|---|---|
| G_NORM_ATTR | — | A nem kiválasztott elemek színe |
| G_ACT_ATTR | — | Az éppen kiválasztott, azaz szerkesztett elem színe |
| G_HIDDEN_ATTR | — | A láthatatlanná, azaz nem kiválaszthatóvá tett elemek színe |
| G_PROMPT_ATTR | — | Az inputmező előtt megjelenő szöveg (prompt) színe |

A következő néhány konstans a mezők szerkesztése során használt kilépő billentyűket adja meg. Ezeket használják azok az eljárások, amelyeknek nem lehet megadni ilyen paramétert. Ezek MACRO típusúak.

- | | | |
|--------------------|---|--|
| I_EXIT_KEYS | — | Megadja, hogy a mezők szerkesztése során mely billentyűk lenyomása jelzi, hogy kész az adat bevitele. (kezdetben csak az ENTER) |
| I_ESC_KEYS | — | Azok a billentyűk, amelyeknek a lenyomásával jelezheti a felhasználó, hogy adat megadása nélkül akar kilépni a szerkesztésből. Kezdetben ez csak az ESC. |

E_EXIT_KEYS

—

Az ablakok tartalmának szerkesztésére alkalmas eljárások számára adja meg, hogy mely billentyűk lenyomása jelenti a kilépést. Kezdetben az ENTER és az F10 billentyűket tartalmazza.

20. fejezet

Eljárások, függvények

20.1. Inicializálás/lezárás

A ZKEY, ZWINDOW és ZMENU programoktól eltérően a következő két eljárást csak akkor kell használni, ha inputtáblával akarunk dolgozni a programban, különben nem kötelező.

zi_begin()

Név: zi_begin() – táblakezelés kezdődik

Használat: #include <zkey.h>
#include <zwindow.h>
#include <zinput.h>
BOOL zi_begin(LEN maxtable);

Leírás: Az inputtáblák kezelése előtt meghívandó inicializáló eljárás. Megfelelő méretű puffereket foglal le a táblák kezeléséhez. A paraméterben adható meg, hogy a programban hány táblát akarunk egyidőben használni. Tehát ha egy táblát használat után törölünk, és csak ezután készítünk újat, akkor elég a maxtable = 1.

Visszatérés: TRUE, ha sikerült lefoglalnia a puffereket, FALSE, ha nem volt elég memória.

Lásd még: zi_end()

Példa: `ret = zi.begin(1);`
 `if( !ret ) exit(1);`

zi_end()

Név: `zi_end()` – táblakezelés lezárása

Használat: `#include <zkey.h>`
 `#include <zwindow.h>`
 `#include <zinput.h>`
 `void zi_end(void);`

Leírás: A táblák kezelése során használt puffereket szabadítja fel. Akkor célszerű meghívni, ha már nem akarjuk többet az inputtáblát használni.

Visszatérés: —

Lásd még: `zi_begin()`

Példa: `zi_begin(1);`
 `...`
 `zi_end();`

20.2. Editor

Ez a szakasz ismerteti azt a két eljárást, amelyek segítségével módosítani lehet az aktuális ablak tartalmát. Egy WORDSTAR-hoz hasonló full screen editor áll tehát rendelkezésünkre.

edit_wind()

Név: edit_wind() – ablak szerkesztése

Használat: #include <zkey.h>
#include <zwindow.h>
#include <zinput.h>
RET_VALUE edit_wind(ADDR str);

Leírás: Az aktuális ablak tartalmát kitörli, majd beírja az adott stringet. Ezután az ablak bal felső sarkába állítja a kurzort, és lehetővé teszi az ablak tartalmának a módosítását. A következő billentyűk használhatók, a WORDSTAR-ból ismert módon:

CTRL/PGUP
CTRL/PGDN
CTRL/HOME
CTRL/END
CTRL/Y
CTRL/Q—CTRL/Y
CTRL/T

Az editorból az E_EXIT_KEYS változó által megadott billentyűkkel lehet kilépni. Ezután az ablak tartalma a ZWINDOW modul leírásában ismertetett módon leolvasható.

Visszatérés: Nem negatív érték esetén az a billentyű, amivel kiléptek az editorból. Negatív érték esetén pedig a hiba kódja.

Lásd még: xedit_wind()

Példa: wselect(wnum);
ret = edit_wind(my_str);

xedit_wind()

Név: xedit_wind() – ablak szerkesztése

Használat: #include <zkey.h>
#include <zwindow.h>
#include <zinput.h>
RET_VALUE xedit_wind(POS row, POS col,
MACRO exitkeys, CODE control);

Leírás: Az aktuális ablak aktuális tartalma módosítható ezzel az eljárással. Nagyon hasonló az edit_wind() eljáráshoz, de annál többet tud. Megadható ugyanis, hogy az ablak mely pozíciójára kell állni az editálás megkezdése előtt, továbbá hogy milyen billentyűkkel lehet kilépni az editálásból. A control paraméterrel pedig szabályozni lehet, hogy valóban editálni, vagy csak megnézni lehet az ablak tartalmát. A control paramétert ettől függően SHOW_ONLY vagy EDIT értékre kell beállítani.

Visszatérés: Nem negatív érték esetén az a billentyű, amivel kiléptek az editorból. Negatív érték esetén pedig a hibakód.

Lásd még: edit_wind()

Példa: KEY my_exit_keys[] = { F9, F10, CTRL('q'), 0 };
wselect(wnum);
wprint(wnum, my_str, DEF_ATTR);
ret = xedit_wind(2, 2, my_exit_keys, SHOW_ONLY);

20.3. Formátumos input

Ez a szakasz a beolvasó eljárások egyik fajtáját ismerteti, nevezetesen azt, amikor meg lehet adni egy ún. picture-t, ami leírja, hogy az adatmező egyes pozícióira milyen karakterek kerülhetnek. Oda nem illőt nem lehet beírni.

A következő vezérlőkaraktereket lehet a picture-ben használni:

9	—	egy számjegy 0 és 9 között.
0	—	számjegy, vagy SPACE
8	—	számjegy, SPACE vagy előjel (+,-).
7	—	számjegy, SPACE, előjel, 'e' vagy 'E' betű
a,A	—	kis és nagy betű vagy SPACE
b,B	—	kis és nagy betű, SPACE, számjegy
c,C	—	32 és 126 közé eső karakterkódok
x,X	—	bármely 1 és 255 kód között

A nagy- és kisbetűs vezérlőkarakterek között az a különbség, hogy nagybetűk esetén az adott pozícióra csak nagybetű kerülhet, tehát a betű magától konvertálódik, a SHIFT használata nélkül is.

A "98888" picture esetén például csak egy olyan adatot adhatunk meg, amelynek a nulladik pozícióján számjegy van, a többin pedig szám vagy SPACE.

Ha a picture tartalma nem vezérlő karakter, akkor az megjelenik az inputmezőben, de nem lehet módosítani, mert a kurzor mindig átugorja. Például a személyi szám beolvasásához az előre definiált a következő picture:

"0 000000 0000"

Egy ilyen picture-rel rendelkező mező szerkesztése során a két SPACE karaktert a kurzor mindig átugorja.

edit_picture()

Név: edit_picture() – formátumos mező szerkesztése

Használat: #include <zkey.h>
#include <zwindow.h>
#include <zinput.h>
RET_VALUE edit_picture(WNUM wnum, POS row, POS col,
ADDR prompt, ADDR picture, ADDR field,
ATTR pattr, ATTR attr,
PFP val_fv, PAR val_par);

Lefrás: Az adott ablakban az adott helyen megjelenik a prompt és a szerkeszthető mező értéke, egészen addig, amíg az ESC vagy az ENTER billentyűt nem nyomjuk meg. Az ENTER-rel való kilépés előtt azonban meghívódik a val_fv, és megkapja a val_par és a field paramétereket.

Az előbbit azért, hogy ne kelljen minden egyes inputmezőhöz külön ellenőrző függvényt készíteni, hanem egy paraméter segítségével lehessen kiválasztani, hogy mit is akarunk ellenőrizni (Erre látható egy példa a date_val() függvény ismertetésénél). A field pedig arra a stringre mutató pointer, amit a felhasználó megadott. Ezt ellenőrizni kell, hogy helyes-e. Ha ez a függvény -1 értéket ad vissza, azt jelzi, hogy megfelelő értéket kapott. Ha nem negatív értéket ad, akkor az annak a karakternek a helye az inputmezőben, ami a hibát okozta. Ekkor a kurzor rááll a mezőben a megadott karakterre, meghívódik a bell() hanggeneráló függvény, és folytatódik a szerkesztés egy érvényes adat megadásáig, vagy az ESC billentyű lenyomásáig.

Visszatérés: Ha pozitív, akkor annak a billentyűnek a kódja, ami miatt kiléptünk (ESC, ENTER), ha negatív, akkor pedig egy hibakód.

Lásd még: put_picture() xedit_picture() date_val()

Példa:

```
#define MY_PAR 1
POS my_val(MY_PAR par, ADDR field)
{ if( par!=MY_PAR || !strcmp(field," ") )
    return(-1);
else
    return(0);}
edit_picture(w,0,10,"- >","Aaa",field,23,23,
my_val,URES);
```

put_picture()

Név: put_picture() – formátumos mező kiírása

Használat:

```
#include <zkey.h>
#include <zwindow.h>
#include <zinput.h>
RET_VALUE xedit_wind(WNUM wnum, POS row, POS col,
    ADDR prompt, ADDR picture, ADDR field,
    ATTR pattr, ATTR attr);
```

Leírás: Az adott ablakban az adott helyre kiírja a megadott promptot a pattr színnel, majd mögé írja a field értékét, amire még rárakja a picture-ből azokat a karaktereket, amelyek nem vezérlőkarakterek. Például a személyi számnál a két SPACE-t. Ezek bekerülnek a field tartalmába is.

Visszatérés: GOOD, ha minden rendben volt. Negatív érték esetén pedig a hibakód.

Lásd még: edit_picture() xedit_picture()

Példa:

```
w = wopen(wdata_ptr);
strcpy(szem-szam,"1 621216 7506");
put_picture(w,0,5,"Szem szam:",PN_PIC,szem-szam,
    BG*BLUE+WHITE,BG*BLUE+YELLOW);
```

xedit_picture()

Név: xedit_picture() – formátumos mező szerkesztése

Használat: #include <zkey.h>
#include <zwindow.h>
#include <zinput.h>
RET_VALUE xedit_picture(WNUM wnum,
POS row, POS col,
ADDR prompt, ADDR picture, ADDR field,
ATTR pattr, ATTR attr,
PFP val_fv, PAR val_par,
MACRO exit_keys, MACRO esc_keys,
CODE control);

Leírás: Az edit_picture() eljárás bővített változata. Az ott ismertett paramétereken kívül megadhatók még a kilépő billentyűk, valamint egy control érték is, ami bizonyos konstansok összege: LEFT_JUST, SHOW_ONLY, EDIT, FIRST_CHAR stb.

Visszatérés: Ha pozitív, akkor annak a billentyűnek a kódja, ami miatt kiléptünk, ha negatív, akkor hibakód.

Lásd még: put_picture() edit_picture()

Példa: xedit_picture(w, 0, 10, "– >", PN_PIC, field, 23, 23,
pn_val, EMPTY, my_exit, my_esc, FIRST_CHAR);

20.4. Mező input

Ebben a szakaszban azokat a beolvasó eljárásokat ismertetjük, amelyek formátum nélküli inputot valósítanak meg. Ellenőrző függvény azonban ezekhez is megadható. Előnyös tulajdonságuk, hogy az ablakban látható mezőnél hosszabb mezőt is szerkeszthetnek. Ekkor a mező tartalma mindig a megfelelő irányba mozog.

`edit_field()`

Név: `edit_field()` – mező szerkesztése default értékekkel

Használat: `#include <zkey.h>`
`#include <zwindow.h>`
`#include <zinput.h>`
`RET_VALUE edit_field(WNUM wnum, POS row, POS col,`
`LEN field_len, ADDR prompt, ADDR field,`
`ATTR attr, ATTR attr);`

Leírás: Kiírja a promptot és a field aktuális értékét, majd lehetővé teszi a mező szerkesztését. Az ENTER és az ESC billentyűvel lehet befejezni. Ennek az eljárásnak nem adható meg ellenőrző függvény, és a szerkeszthető mező hossza megegyezik az ablakban is látható hosszal.

Visszatérés: a kilépő billentyű, vagy negatív hibakód.

Lásd még: `xedit_field()`

Példa: `strcpy(field, "Ez a default");`
`edit_field(w, 0, 0, 15, "-->", field, 23, 30);`

put_field()

Név: put_field() – mező kiírása

Használat: #include <zkey.h>
#include <zwindow.h>
#include <zinput.h>
RET_VALUE put_field(WNUM wnum, POS row, POS col,
LEN field_len, ADDR prompt, ADDR field,
ATTR pattr, ATTR attr);

Leírás: Egy mezőt ír ki az ablak adott helyére. A prompt a mező elé kerül. Ha a field hossza nagyobb, mint a field_len paraméterben megadott érték, akkor csak az eleje jelenik meg.

Visszatérés: GOOD, vagy negatív hibakód.

Lásd még: edit_field()

Példa: strcpy(field, "Itt van egy szöveg");
put_field(w, 0, 0, 15, "-->", field, 23, 23);

xedit_field()

Név: xedit_field() – mező szerkesztése

Használat: #include <zkey.h>
#include <zwindow.h>
#include <zinput.h>
RET_VALUE edit_field(WNUM wnum, POS row, POS col,
LEN field_len, LEN virt_len,
ADDR prompt, ADDR field,
ATTR pattr, ATTR attr,
PFP val_fv, PAR val_par,
MACRO exit_keys, MACRO esc_keys,
CODE control);

Leírás: Az `edit_field()` eljárásnál bővebben paraméterezhető beolvasást valósít meg. Megadható az ellenőrző függvény, a kilépő billentyűk, és az ablakban látható mezőnél hosszabb adatot is lehet segítségével bevinni. A valódi hosszat a `virt_len` paraméter adja meg. Az ellenőrző függvény csak az `exit_keys` billentyűk valamelyikével történő kilépéskor hívódik meg. Funkciója az `edit_picture()` eljárásnál leírtakkal azonos.

Visszatérés: a kilépő billentyű, vagy negatív hibakód.

Lásd még: `edit_field()`

Példa:

```
strcpy(field,"Ez a default szoveg itt most");
xedit_field(w,0,0,15,40,"-->",field,23,30,
            my_exit,my_esc,EDIT|FIRST_CHAR);
```

20.5. Beépített ellenőrző függvények

Ebben a szakaszban azok az előre definiált ellenőrző függvények szerepelnek, amelyek a dátum, az időpont és a személyi szám vizsgálatára alkalmasak.

`date_val()`

Név: `date_val()` – dátumellenőrző

Használat:

```
#include <zkey.h>
#include <zwindow.h>
#include <zinput.h>
TACTION date_val(PAR par, ADDR field);
```

Leírás: A `par` paraméter szerint megvizsgálja, hogy a `field` érvényes dátumot ad-e meg. Ha a `par` paraméter `EMPTY` bitje egyes, akkor az üres dátum is megfelelő. Ezen kívül a `par` paraméternek a `HUN_DATE`, `EUR_DATE` vagy `USA_DATE` bitjétől függően megvizsgálja a dátum formátumát. Természetesen a szökőéveket is figyelembe veszi.

Visszatérés: Ha hibát talált, akkor a hibás karakter pozíciója a mezőn belül, ami 0 és (mezőhossz-1) közé eshet. Ha nem volt hiba, akkor -1.

Lásd még: time_val() pn_val()

Példa:

```
strcpy(field,"62-12-16");
edit_picture(w,0,10,"-->",DATE_PIC,field,23,23,
            date_val,HUN_DATE|EMPTY);
```

pn_val()

Név: pn_val() – személyi szám ellenőrző

Használat:

```
#include <zkey.h>
#include <zwindow.h>
#include <zinput.h>
TACTION pn_val(PAR par, ADDR field);
```

Leírás: A par paraméter szerint megvizsgálja, hogy a field érvényes személyi számot ad-e meg. Ha a par paraméter EMPTY bitje egyes, akkor az üres személyi szám is megfelelő.

Visszatérés: Ha hibát talált, akkor a hibás karakter pozíciója a mezőn belül. Ha nem volt hiba, akkor a -1 érték.

Lásd még: time_val() date_val()

Példa:

```
strcpy(field,"1 621216 7506");
edit_picture(w,0,10,"->",PN_PIC,field,23,23,
            date_val,PN_NUMBER|EMPTY);
```

time_val()

Név: time_val() – időpont ellenőrző

Használat: #include <zkey.h>
#include <zwindow.h>
#include <zinput.h>
TACTION time_val(PAR par, ADDR field);

Leírás: A par paraméter szerint megvizsgálja, hogy a field érvényes időpontot ad-e meg. Ha a par paraméter EMPTY bitje egyes, akkor az üres mezőt is elfogadja. Ezen kívül a par paraméternek a HOUR_12 vagy HOUR_24 bitjétől függően vizsgálja az időpont formátumát.

Visszatérés: Ha hibát talált, akkor a hibás karakter pozíciója a mezőn belül. Ha nem volt hiba, akkor -1.

Lásd még: date_val() pn_val()

Példa: strcpy(field, "22:23:12");
edit_picture(w, 0, 10, "—>", TIME_PIC, field, 23, 23,
time_val, HOUR_24|EMPTY);

20.6. Input táblák kezelése

A következő eljárások segítségével inputtáblákat lehet megnyitni, lezárni, és használni. Ezek az eljárások az egész táblára vonatkoznak, a táblák egyes elemeit kezelő eljárásokat a következő szakaszban ismertetjük.

tclose()

Név: tclose() – tábla lezárása

Használat: `#include <zkey.h>`
`#include <zwindow.h>`
`#include <zinput.h>`
`RET_VALUE tclose(TNUM tnum);`

Leírás: Az adott inputtáblát lezárja, azaz felszabadítja a hozzá tartozó puffereket. A tábla ettől kezdve nem használható.

Visszatérés: GOOD, vagy negatív hibakód.

Lásd még: `topen()`

Példa: `tnum = topen(5);`
`...`
`tclose(tnum);`

tdraw()

Név: tdraw() – tábla kirajzolása

Használat: #include <zkey.h>
#include <zwindow.h>
#include <zinput.h>
RET_VALUE tdraw(TNUM tnum);

Leírás: Az adott inputtábla valamennyi elemét kiírja a hozzá tartozó ablakba (nem szükséges minden elemnek egy ablakban lennie).

Visszatérés: GOOD, vagy negatív hibakód.

Lásd még: topen()

Példa: tnum = topen(5);
...
tdraw(tnum);

terase()

Név: terase() – tábla törlése

Használat: #include <zkey.h>
#include <zwindow.h>
#include <zinput.h>
RET_VALUE terase(TNUM tnum);

Leírás: Az adott inputtábla valamennyi elemét feltölti SPACE karakterekkel. Ez csak egy tdraw() hívás után látszik az ablakban.

Visszatérés: GOOD, vagy negatív hibakód.

Lásd még: topen() tdraw()

Példa: tnum = topen(3); ...
tread(tnum);
terase(tnum);
tread(tnum);

topen()

- Név:** topen() – tábla megnyitása default értékekkel
- Használat:** `#include <zkey.h>`
`#include <zwindow.h>`
`#include <zinput.h>`
`TNUM topen(LEN maxget);`
- Leírás:** Megnyit egy inputtáblát, amelyben maximum maxget elem számára van hely.
- Visszatérés:** Ha nem negatív, akkor a frissen megnyitott tábla azonosítója.
Ha negatív, akkor a hiba kódja.
- Lásd még:** txopen() tclose()
- Példa:** `tnum = topen(10);`
`if( tnum<0 ) exit(1);`

tread()

- Név:** tread() – tábla szerkesztése az első elemtől
- Használat:** `#include <zkey.h>`
`#include <zwindow.h>`
`#include <zinput.h>`
`RET_VALUE tread(TNUM tnum);`

Leírás:

Az adott tábla szerkesztését teszi lehetővé. Ez azt jelenti, hogy a programozó által megadott függvények a megfelelő alkalmakkor meghívódnak. (A tread elején a tbegin_fv, a végén pedig a tend_fv.) Az inputtábla mezői között a felfelé és lefelé nyilakkal lehet mozogni, a mezőn belül pedig a mező szerkesztésénél leírt billentyűk segítségével lehet az adatokat szerkeszteni.

A következő billentyűk használhatók az inputtábla elemei között való lépkedésre:

- Felfelé és lefelé nyilak az előző illetve a következő elem kiválasztásához.

- PGUP vagy CTRL/HOME az első elemre való ugráshoz.

- PDGN vagy CTRL/END az utolsó elemre való ugráshoz.

A következő billentyűk valamelyikével lehet a tábla szerkesztéséből kilépni, azaz engedélyt kérni a kilépést engedélyező függvénytől:

- Felfelé nyíl, ha az inputtábla legelső eleménél ütöttük le.

- Lefelé nyíl, ha a legalsó elemnél nyomtuk meg.

- ENTER, ha a legasló elemnél ütöttük le.

- CTRL/ENTER, bármely elem szerkesztése közben.

- ESC, bármely elem szerkesztése közben.

Ha nincs megadva kilépést ellenőrző függvény, akkor a le és fel nyilak úgy működnek, mintha az első elem előtt az utolsó állna, és viszont. Az ENTER és a CTRL/ENTER olyan kilépést jelent a táblából, amelynél a beírt adatok érvényesek. Az ESC pedig olyan kilépés, ahol a beírt adatok helyességének ellenőrzése nem történik meg.

Visszatérés: A kilépést okozó billentyű, vagy negatív hibakód.

Lásd még: txread() tdraw()

Példa:

```
field = malloc(51);
strcpy(field, "");
tnum = topen(1);
get_field(tnum, 0, 10, 20, 50, "-->", field, NULL, 0);
tread(tnum);
```

txopen()

Név: txopen() – tábla megnyitása

Használat: #include <zkey.h>
#include <zwindow.h>
#include <zinput.h>
TNUM txopen(LEN maxget, PFTR tcall_fv,
PFT tbegin_fv, PFT tend_fv,
PFTG gto_fv, PFTG gfrom_fv);

Leírás: Megnyit egy inputtáblát, amely maxget számú elemnek biztosít helyet, és emellett a felsorolt függvények adhatók meg a táblához. Ezek bármelyike lehet NULL, ha nem akarunk eltérni az alapfeltételezésektől.

A tcall_fv-t az inputtáblából való kilépés előtt hívja meg a táblakezelő eljárás. A tbegin_fv és a tend_fv függvények az inputtáblába való be illetve kilépéskor hívódnak meg. A gto_fv és a gfrom_fv pedig az táblában való lépkedés során az elemekre való rálépéskor, illetve a róluk való lelépéskor aktivizálódnak.

Visszatérés: Ha nem negatív, akkor a frissen megnyitott tábla azonosítója. Ha negatív, akkor a hiba kódja.

Lásd még: topen() tclose()

Példa: tnum = txopen(10, NULL, NULL, NULL, NULL, my_to, NULL);

txread()

Név: txread() – tábla szerkesztése adott elemtől

Használat: #include <zkey.h>
#include <zwindow.h>
#include <zinput.h>
RET_VALUE txread(TNUM tnum, GNUM gnum);

- Leírás:** Az inputtábla szerkesztését teszi lehetővé, a `tread()` eljárásnál ismertetett módon. Azzal a különbséggel, hogy a szerkesztést nem a legelső, hanem a megadott elemmel lehet kezdeni.
- Visszatérés:** A kilépést okozó billentyű, vagy negatív hibakód.
- Lásd még:** `tread()` `tdraw()`
- Példa:** `txread(tnum,3);`

20.7. Az input tábla elemei

A következő eljárások arra szolgálnak, hogy az inputtáblába új elemeket vegyenek fel, onnan elemet töröljenek ki, vagy a már korábban felvett elemeket változtassák meg — akár a tábla használata közben is.

`gdel()`

- Név:** `gdel()` – elem törlése
- Használat:** `#include <zkey.h>`
`#include <zwindow.h>`
`#include <zinput.h>`
`RET_VALUE gdel(TNUM tnum, GNUM gnum);`
- Leírás:** A megadott elemet törli a táblából, és egyúttal az ablakból is. Erre az elemre többé nem lehet hivatkozni. Ügyelni kell arra, hogy ha egy elem törlése után egy új elemet veszünk fel az inputtáblába, akkor az a törölt elem helyére kerül, tehát annak a sorszámát kapja.
- Visszatérés:** `GOOD`, vagy negatív hibakód.
- Lásd még:** `ghide()` `get_picture()`
- Példa:** `gdel(tnum, LEANY_NEV);`

get_field()

Név: `get_field()` – elem felvétele a táblába default értékkel

Használat: `#include <zkey.h>`
`#include <zwindow.h>`
`#include <zinput.h>`
`GNUM get_field(TNUM tnum, POS row, POS col,`
`LEN flen, LEN vlen,`
`ADDR prompt, ADDR field,`
`PFP val_fv, PAR val_par);`

Leírás: Az `edit_field()` eljárás paramétereinek megfelelően felvesz egy elemet az inputtáblába. Ez az elem az éppen aktuális ablakba fog tartozni. Az elemet nem írja ki az ablakba, az csak a `tdraw()` eljárással tehető meg.

Lásd még: `get_picture()` `xget_field()`

Példa: `get_field(tnum, 0, 10, 20, 50, "-->", field, val_fv, 0);`

get_picture()

Név: `get_picture()` – elem felvétele a táblába

Használat: `#include <zkey.h>`
`#include <zwindow.h>`
`#include <zinput.h>`
`GNUM get_picture(TNUM tnum, POS row, POS col,`
`ADDR prompt, ADDR picture, ADDR field,`
`PFP val_fv, PAR val_par);`

Leírás: Az `edit_picture()` eljárás paramétereinek megfelelően felvesz egy elemet az inputtáblába. Ez az elem az éppen aktuális ablakba fog tartozni. Ugyanis nem feltétlenül szükséges, hogy egy inputtábla minden eleme ugyanabban az ablakban legyen. Az elemet nem írja ki az ablakba, ez csak a `tdraw()` eljárással tehető meg.

Lásd még: `xget_picture()` `get_field()`

Példa: `get_picture(tnum, 0, 10, "-->", "Aaa", field, my_val, MY_PAR);`

get_picture_right()

Név: `get_picture_right()` – elem felvétele a táblába (default)

Használat: `#include <zkey.h>`
`#include <zwindow.h>`
`#include <zinput.h>`
`GNUM get_picture_right(TNUM tnum, POS row, POS col,`
`ADDR prompt, ADDR picture, ADDR field,`
`PFP val_fv, PAR val_par);`

Leírás: Felvesz egy olyan elemet az inputtáblába, amely jobbra igazítva szerkeszthető. Ezt a lehetőséget csak számok bevitelére célszerű használni.

Lásd még: `get_picture()`

Példa: `get_picture_right(tnum, 0, 10, "-->",`
`"Aaa", field, my_val, MY_PAR);`

ghide()

Név: `ghide()` – elem elrejtése

Használat: `#include <zkey.h>`
`#include <zwindow.h>`
`#include <zinput.h>`
`RET_VALUE ghide(TNUM tnum, GNUM gnum);`

Leírás: Az adott tábla, adott sorszámú elemét nem kiválasztható elemmé teszi. Erre olyan esetben lehet szükség, amikor az input-tábla eddigi adatai alapján nem lehet kitölteni az adott mezőt. Ilyen például az az eset, amikor a "neme" mező "férfi" értéke esetén letiltjuk a leánykori név beolvasását, de a "nő" érték esetén újra engedélyezzük.

Az elem rejtettségét a színváltozás is jelzi.

Visszatérés: GOOD, vagy negatív hibakód.

Lásd még: `gshow()` `gdel()`

Példa: `if( !strcmp(neme, "ferfi"))`
`ghide(tnum, LEANY_NEV);`

gput_get()

Név: gput_get() – elem kiírása

Használat: #include <zkey.h>
#include <zwindow.h>
#include <zinput.h>
RET_VALUE gput_get(TNUM tnum, GNUM gnum);

Leírás: A tábla adott elemét felfrissíti, azaz újra kiírja az ablakba. Erre akkor lehet szükség, ha a táblakezelő által hívott valamelyik függvény megváltoztatta az elem értékét. Érdemes ezt meghívni, hogy a változás ne csak akkor váljon láthatóvá, amikor a mezőt szerkesztésre választják ki.

Visszatérés: GOOD, vagy negatív hibakód.

Lásd még: clear_field()

Példa: strcpy(person.name, "Kovacs Jozsef");
gput_get(tnum, PERS_NAME);

gshow()

Név: gshow() – elem megjelenítése

Használat: #include <zkey.h>
#include <zwindow.h>
#include <zinput.h>
RET_VALUE gshow(TNUM tnum, GNUM gnum);

Leírás: A tábla megadott elemét újra kiválaszthatóvá teszi. Ezzel a ghide() eljárás hatását visszaállítja.

Visszatérés: GOOD, vagy negatív hibakód.

Lásd még: ghide() gdel()

Példa: if(!strcmp(person.name, "nő"))
gshow(tnum, LEANY_NEV);

xget_field()

Név: xget_field() – elem felvétele a táblába

Használat: `#include <zkey.h>`
`#include <zwindow.h>`
`#include <zinput.h>`
GNUM xget_field(TNUM tnum, WNUM wnum,
POS row, POS col,
ADDR prompt, ADDR picture, ADDR field,
ATTR prompt_attr, ATTR norm_attr,
ATTR act_attr, ATTR hidden_attr,
PFP val_fv, PAR val_par, CODE code);

Leírás: Felvesz egy elemet az inputtáblába. A get_field() eljárásához hasonló, de ezzel az eljárással meg lehet adni, hogy az elem mely ablakban legyen, és lehetőséget ad arra, hogy az elem színét a megfelelő globális változókkal megadott értéktől eltérően definiáljuk.

Ha egy újonnan létrehozott inputtáblában veszünk fel elemeket, akkor a sorszámozás 0-val kezdődik.

Visszatérés: Ha nem negatív, akkor az újonnan felvett elem sorszáma, ha negatív, akkor a hiba kódja.

Lásd még: xget_picture() get_field()

Példa: `get_picture(tnum, wnum, 0, 10, "-->", "Aaa", field,
23, 23, my_val, MY_PAR, FIRST LETTER);`

xget_picture()

Név: xget_picture() – elem felvétele a táblába

Használat: #include <zkey.h>
#include <zwindow.h>
#include <zinput.h>
GNUM xget_picture(TNUM tnum, WNUM wnum,
POS row, POS col,
ADDR prompt, ADDR picture, ADDR field,
ATTR prompt_attr, ATTR norm_attr,
ATTR act_attr, ATTR hidden_attr,
PFP val_fv, PAR val_par, CODE code);

Leírás: A get_picture() eljáráshoz hasonlóan felvesz egy elemet az inputtáblába. Megadható, hogy az elem mely ablakba kerüljön (ez nemcsak az aktuális ablak lehet), sőt az is, hogy milyen színű legyen.

Az inputtáblán belül az új elem mindig a legkisebb olyan sorszámot kapja, ami még nem foglalt. Mivel új tábla esetén nullától kezdve kapjuk a sorszámokat, tehát legnagyobb értéke a lehetséges elemszámnál eggyel kisebb.

Visszatérés: Ha nem negatív, akkor az újonnan felvett elem sorszáma, ha negatív, akkor a hiba kódja.

Lásd még: get_picture() xget_field()

Példa: xget_picture(tnum, wnum, 0, 10, "-->", "Aaa",
field, my_val, MY_PAR, SHOW_ONLY);

20.8. Egyéb eljárások

clear_field()

Név: clear_field() – mező törlése

Használat: #include <zkey.h>
#include <zwindow.h>
#include <zinput.h>
void clear_field(ADDR field, LEN len);

Leírás: Az adott című mezőt feltölti SPACE-ekkel, és a végére kiteszi a string végét jelző nullát. Akkor használható jól, ha egy frissen allokált helyet törölni kell. Figyeljünk azonban arra, hogy a záró nullának is legyen helye. A len paraméter legyen eggyel kisebb, mint a lefoglalt hely mérete.

Visszatérés: —

Lásd még: terase() tdraw()

Példa: ptr = malloc(21);
clear_field(ptr, 20);

Tárgymutató

ACTIVE_MENU 112
ACT_ATTR 117
ACT_BORDER_ATTR 117
ACT_MARKED_ATTR 117
ACT_TITLE_ATTR 117
ADDR 48
ATTR 48
A_1 ... A_0 14
A_A ... A_Z 14
A_EQ 14
A_F1 ... A_F10 14
A_MINUS 14
a,A 169
active_menu() 128
act_battr 50
act_border 48
act_tattr 50
add_hot_key() 27
add_macro() 30
adjust_window() 100
ask_hot_key() 27
ask_macro() 30
BADCHAR 111
BADGNUM 158
BADITEM 111
BADMNUM 111
BADNUM 45
BADPAR 45
BADPOS 45
BADTNUM 158
BG 47
BITEM 114
BKSP 14
BLINK 47
BOOL 16, 48
BORDER0 46
BORDER1 46
BORDER2 46
BORDER3 46
BORDER4 46
BORDER_ATTR 117
BTAB 14
b,B 169
battr 48
bell() 35
bell_on 18
billyentyűzet-puffer 21, 23, 25, 31
border 48
bset_menu() 120
btitle 48
build_menu() 121
bw() 102

- bxset_menu() 122
- CFMI 115
- CLOSE_ALL 47
- CLOSE_MENU 112
- CLR_SCR 47
- CODE 114
- CTRL(char) 15
- CURRENT 45
- CUR_ITEM 111
- CUR_ATTR 47
- C.BKSP 15
- C.END 15
- C.ENTER 14
- C.ENTER_EXIT 158
- C.F1 ... C.F10 14
- C.HOME 15
- C.LEFT 15
- C.PGDN 15
- C.PGUP 15
- C.RIGHT 15
- c.C 169
- call_hot_key() 28
- call_macro() 31
- call_menu() 129
- call_menu_key() 130
- change_state() 93
- change_state_end() 93
- clear_field() 188
- clear_hot_keys() 28
- clear_keys() 22
- clear_macros() 31
- close_menu() 131
- control 48
- DATE_PIC 160
- DEF_ATTR 47
- DEF_SIZE 113, 126
- DEL 15
- DOWN 15
- DOWN_EXIT 159
- date_val() 175
- del_hot_key() 29
- del_macro() 32
- depth 50
- EDIT 159
- EMPTY 160
- END 14
- ENTER 14
- ENTER_EXIT 158
- ESC 14
- ESC_EXIT 158
- ESC_PRESSED 111
- EUR_DATE 160
- E_EXIT_KEYS 163
- edit_field() 172
- edit_picture() 169
- edit_wind() 166
- F1 ... F10 14
- FIRST_CHAR 159
- FREQ 16
- FULLMEM 111
- FULLWINDOW 45
- fill_ch 48
- free_menu() 132
- GNUM 161
- GOOD 45
- G_ACT_ATTR 163
- G_CURRENT 159
- G_ESC 159
- G_EXIT 159
- G_FIRST 159
- G_HIDDEN_ATTR 163
- G_LAST 159
- G_NORM_ATTR 163
- G_PROMPT_ATTR 163

- gdel() 183
- get_field() 183
- get_picture() 184
- get_picture_right() 184
- ghide() 185
- good_key() 22
- go_menu() 133
- gput_get() 185
- gshow() 186
- HIDDEN_ATTR 117
- HIDDEN_ITEM 111
- HIDE_MENU 112
- HITEM 114
- HOME 14
- HOT-KEY 11, 21, 26, 29
- HOUR_12 160
- HOUR_24 160
- HUN_DATE 160
- hertz 36
- hide_cursor() 99
- hide_menu() 133
- hide_state() 94
- hot_keys_on 19
- hset_menu() 123
- hxset_menu() 124
- INS 15
- INUM 114
- I_ESC_KEYS 163
- I_EXIT_KEYS 163
- inicializálás 21, 46
- input-tábla 155
- is_color_mon 54
- is_direct 55
- is_ega 53
- is_snow 54
- is_wzoom() 91
- item_col 114
- item_height 114
- item_lpos 114
- item_row 114
- item_str 114
- item_width 114
- KEEP_MENU 112
- KEY 16
- key_pressed() 23
- LEFT 14
- LEFT_JUST 159
- LEN 16, 48
- LETTER_ATTR 117
- LF 14
- loc 48
- lor 48
- MACRO 11, 16, 21, 30
- MACTION 114
- MACTIVE 112
- MACTMARKED 112
- MARKED_ATTR 117
- MAXSIZE 46
- MAX_COL 52
- MAX_ROW 52
- MEMORYFULL 158
- MEXIT 112
- MHIDDEN 112
- MIN_COL 51
- MIN_ROW 51
- MITEM 114
- MMARKED 112
- MNOEXIT 112
- MNORMAL 112
- MNUM 114
- M_ACT_BORDER 117
- M_BORDER 117
- M_CALL_KEYS 118
- M_CHOOSE 113

- M_ESC_KEYS 118
- M_SELECT 113
- macro_keys_on 19
- mchange....() 142
- mchange_attrs() 142
- mchange_battr() 142
- mchange_begin_fv() 143
- mchange_border() 142
- mchange_btitle() 142
- mchange_call_keys() 144
- mchange_call_fv() 143
- mchange_end_fv() 145
- mchange_esc_keys() 144
- mchange_from_fv() 146
- mchange_item_end() 145
- mchange_item_front() 146
- mchange_item_str() 147
- mchange_pos() 147
- mchange_sattr() 142
- mchange_str() 148
- mchange_tattr() 142
- mchange_title() 142
- mchange_to_fv() 148
- mchange_type() 149
- menu_setup() 124
- mget_first_marked() 135
- mget_item_state() 136
- mget_item_str() 137
- mget_item_value() 137
- mget_next_marked() 138
- move_window() 96
- mselect() 150
- mset_act_mark...() 138
- mset_act_marked() 139
- mset_act_nomarked() 139
- mset_act_xmarked() 139
- mset_all_marked() 139
- mset_all_mark...() 139
- mset_all_nomarked() 139
- mset_all_xmarked() 139
- mset_item_hidden...() 140
- mset_item_hidden() 140
- mset_item_mark..() 140
- mset_item_marked() 141
- mset_item_nomarked() 141
- mset_item_nohidden() 140
- mset_item_value() 141
- mset_item_xhidden() 140
- mset_item_xmarked() 141
- mset_menu() 125
- mxset_menu() 126
- NOMOREGET 158
- NOMORETABLE 158
- NOMOVE 45
- NORM_ATTR 117
- NORM_ITEM 111
- NOROOM 45
- NOTACTIVE 45
- NOTFOUND 111
- NOTHIDDEN 45
- NOTOPEN 45
- NOWDEF 45
- NOZOOM 45
- NOZOOMED 45
- NULLFVPTR 111
- NULLSIZE 112
- next_item....() 150
- next_item_char() 150
- next_item_down() 150
- next_item_end() 150
- next_item_home() 150
- next_item_left() 150
- next_item_pgup() 150
- next_item_pgdn() 150

- next_item_right() 150
- next_item_up() 150
- no_sound() 37
- OPEN_MENU 112
- open_menu() 134
- PAR 161
- PFA 48
- PFE 17
- PFK 17
- PFM 115
- PFMI 115
- PFP 161
- PFS 115
- PFT 162
- PFTG 161
- PFTR 162
- PFV 16
- PGDN 15
- PGUP 15
- PN_NUMBER 160
- PN_PIC 160
- POS 48
- pn_val() 176
- pop_state() 93
- push_state() 91
- put_field() 173
- put_picture() 171
- README.ZI 156
- README.ZK 12, 109
- README.ZW 42
- REMOVE_MENU 112
- RETURN 14
- RET_VALUE 48
- RIGHT 14
- RIGHT_JUST 159
- readkey() 23
- readkey_no_hots() 24
- read_macro() 32
- read_matchkey() 24
- refresh_screen() 101
- remove_menu() 135
- SAVE_SCR 47
- SHOW_ONLY 159
- SPACE 14
- S.F1 ... S.F10 14
- sattr 48
- set_bell_func() 35
- set_sound() 36
- set_wait_func() 26
- show_cursor() 100
- show_state() 94
- show_window() 97
- színek 47
- TAB 14
- TACTION 161
- TIME 16
- TIME_PIC 160
- TITLE0 46
- TITLE_ATTR 117
- TNOTOPEN 158
- TNUM 161
- tattr 48
- tclose() 177
- tdraw() 178
- terase() 179
- time_val() 176
- title 48
- topen() 179
- tread() 180
- txopen() 181
- txread() 182
- UP 15
- UP_EXIT 159
- USA_DATE 160

- unreadkey() 25
- upc 48
- upr 48
- VALUE 114
- VITEM 114
- vset_menu() 127
- vxset_menu() 128
- WACTIVE 46
- WADJUST 46
- WAIT függvény 11, 18, 24, 26
- WBLEFT 46
- WBRIGHT 46
- WDATA 48
- WINDOW_ATTR 117
- WMOVE 46
- WNOCUR 46
- WNOZOOM 46
- WNUM 48
- WOPEN 46
- WSIZE 46
- WTLEFT 46
- wait_func_on 18
- watprint() 69
- watput_attr() 71
- watput_attrstr() 72
- watput_char() 73
- watput_charattr() 73
- watput_str() 71
- wchange_battr() 78
- wchange_border() 78
- wchange_btitle() 77
- wchange_control_reset() 81
- wchange_control_set() 81
- wchange_cur() 76
- wchange_mar() 76
- wchange_pos() 82
- wchange_sattr() 79
- wchange_tattr() 79
- wchange_title() 77
- wchange_wattr() 80
- wclose() 61
- wcurrent 53
- weraeol() 67
- weraeos() 66
- werase() 66
- wget_attr() 85
- wget_attrscreen() 88
- wget_char() 85
- wget_line() 86
- wget_screen() 87
- wget_str() 86
- wget_wdata() 88
- whide() 63
- width 50
- wmove() 95
- wopen() 58
- wprint() 68
- wprint_words() 70
- wput_attr() 74
- wput_attrscreen() 75
- wput_char() 74
- wput_charattr() 74
- wselect() 62
- wshow() 64
- wsiz_attrscreen() 84
- wsiz_line() 83
- wsiz_screen() 83
- wxhide() 65
- wxopen() 59
- wxxopen() 60
- wzoom_in() 90
- wzoom_out() 89
- w_active() 98
- w_cur() 99

- w_open() 97
- x,X 169
- xedit_field() 174
- xedit_picture() 171
- xedit_wind() 167
- xget_field() 186
- xget_picture() 187
- ZINPUT.C 156
- ZINPUT.H 156
- ZI.MSC.L.LIB 156–157
- ZI.MSC.M.LIB 156–157
- ZI.MSC.S.LIB 156–1576
- ZI.TC.L.LIB 156–157
- ZI.TC.S.LIB 156
- ZKEY.C 12
- ZKEY.H 12
- ZK.MSC.L.LIB 12
- ZK.MSC.M.LIB 12
- ZK.MSC.S.LIB 12
- ZK.TC.L.LIB 12
- ZK.TC.S.LIB 12
- ZMENU.C 109
- ZMENU.H 109
- ZM.MSC.L.LIB 109–110
- ZM.MSC.M.LIB 109–110
- ZM.MSC.S.LIB 109–110
- ZM.TC.L.LIB 109–110
- ZM.TC.S.LIB 109–110
- ZWINDOW.C 42
- ZWINDOW.H 42
- ZW.MSC.L.LIB 42–43
- ZW.MSC.M.LIB 42–43
- ZW.MSC.S.LIB 42–43
- ZW.TC.L.LIB 42–43
- ZW.TC.S.LIB 42–43
- zi_begin() 165
- zi_end() 166
- zk_begin() 21
- zk_delay() 33
- zk_end() 21
- zk_sound() 36
- zk_wait() 34
- zm_begin() 119
- zm_end() 120
- zw_begin() 56
- zw_end() 57

AKADÉMIAI KIADÓ ÉS NYOMDA VÁLLALAT

Felelős kiadó: Hazai György főigazgató

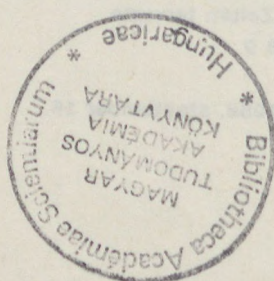
Felelős szerkesztő: Juhász Lehel

A borítót Szabadi Zoltán tervezte

ISBN 963 05 5246 9

ISSN 0238-3896

A kézirat lezárva: 1988. szeptember 15.



ZTOOLS programcsomag

Billentyűzet, Ablak, Menü és Input-tábla

kezelő C könyvtárak

A ZTOOLS könyvtárak az IBM PC/XT/AT kompatibilis számítógépeken C nyelven programozók segédeszköze. A Turbo C 1.5 és a Microsoft C 5.xx verziójával egyaránt használható. A ZTOOLS kifejezetten professzionális alkalmazásokhoz, tapasztalt programozóknak készült. Kifejlesztésekor ezért a legfontosabb szempont a sokrétű paraméterezhetőség volt.

ZKEY — Billentyűzet kezelése

- Speciális billentyűk kezelése (ALT/F1, CTRL-Y, stb.)
- HOT KEY-ek egyszerű használata (definálás, törlés, módosítás, letiltás, engedélyezés)
- Makrók használata (beépített és felhasználó által definiált)
- Hóltidő kihasználása (definálható tevékenységek a várakozási időre)

ZWINDOW — Ablakok kezelése

- Virtuális képernyők kezelése
- Az ablakok mozgathatók, zoom-olhatók, átszínezhetők, keretük és fejlécük tetszőlegesen változtathatók, stb.
- Az ablakok között tetszőlegesen lehet mozogni, ablakokat láthatóvá és láthatatlanná tenni
- Szín váltogatási lehetőségek a szövegen belül
- Az ablakok minden jellemzője, valamint tartalma lekérdezhető
- Beépített eljárások a státusz-sorok kezelésére

ZMENU — Menük kezelése

- Szokásos menük (függőleges, vízszintes, téglalap, pull-down, egymásba ágyazott)
- Speciális menük (több soros menüelemek, tetszőleges menüelem elhelyezés, stb.)
- Menüelemek elrejtése és láthatóvá tétele
- Menük paramétereinek változtatása (hely,szín,méret,fejléc,stb.)
- Több elem kiválasztása

ZINPUT — Adatbeviteli eljárások

- Ellenőrzött adatbeviteli eljárások (picture, ellenőrző függvény)
- Beépített ellenőrző függvények (Idő, dátum, személyi szám)
- A látható beviteli mezőnél hosszabb adatok kezelése
- Input-táblázatok használata
- Full screen editor az ablakok tartalmának szerkesztéséhez

HARDVER: IBM PC/XT, AT vagy velük kompatibilis mikroszámítógép.
Garanciális szolgáltatások. Szoftverkövetés.

