C programozási alapok

Wagner György

Általános Informatikai Tanszék

Általános INFORMATIKAI Tanszék

Hirdetmények (2)

- Jegyzet:
  - Ami az előadáson elhangzik
  - Ajánlott irodalom:
- Elérhetőség:
  - Tel: (46)565-111/17-56
  - Tel: (30)9383-669
  - E-mail: wagner@iit.uni-miskolc.hu

Benkő Tiborné, Benkő László, Tóth Bertalan:

Programozzunk C nyelven

(ComputerBooks kiadó, Budapest, ISBN 963-618-090-3

Alapfogalmak (1)

- Adat: implicit jelentéssel rendelkező ismeretek, tények.
- Információ: olyan közlés, amely valamely értelemben fennálló bizonytalanságot szüntet meg.
- Bit: - bináris számjegy
  - memóriarekesz
  - információ egység
- Byte: - 8 bitből álló számjegycsoport
  - memóriarekesz
- Word (szó): CPU függő
- Character (karakter): egy adott karakterkészlet valamely eleme
- Code (kód): egyezmé-nyes jel, szimbólum, kapcsolt jelentéssel

Alapfogalmak (2)

- Pl: ASCII: American Standard Code for Information Interchange
  - 0-127-ig kötött, felette különböző ajánlások
- még: ECMA, EBCDIC, stb.

A számítástechnika tárgya

- A számítástechnikai eszközök
  - tervezésével
  - üzemeltetésével
  - alkalmazásával (!)

  összefüggő ismeretek, törvényszerűségek, tapasztalatok gyűjtése, rendszerezése és fejlesztése.

Csoportosítások (1)

- Működési elv szerint:
  - analóg: az informácókat folytonos értéktartományban hordozzák
  - digitális: az információk és utasítások (!) kettes számrendszerben vannak tárolva.
  - hibrid: analóg és digitális vegyesen

Csoportosítások (2)

- Teljesítmény szerint:
  - home (Commodore, ZX 81, Sony PSX, ...)
  - personal (PC-k)
  - mini (PDP, TPA, VAX, ...)
  - nagy (IBM, Cray, ...)
  - szuper (Hitachi, Cray)

Csoportosítások (3)

- Cél szerint:
  - célszámítógép (blokkolás gátló, motorvezérlő, varrógépbe, …)
  - univerzális számítógép
- Fizikai fejlettség szerint:
  - 1. generációs (1946-)
  - 2. generációs (1955-)
  - 3. generációs (1966-)
  - 4. generációs (1975-), …

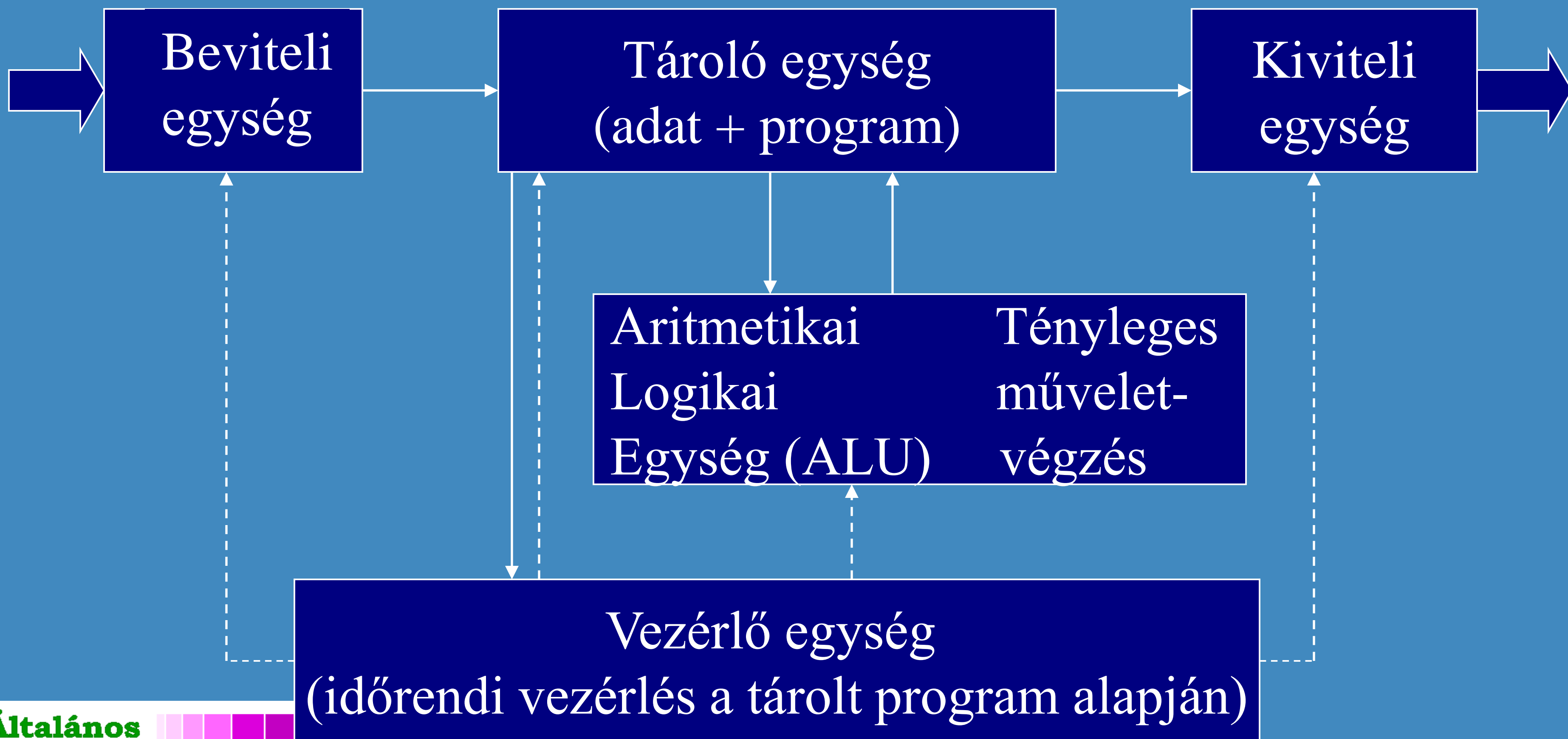
Az ENIAC

- 180 KW áramfelvétel
- 18000 elektroncső
- volt hogy 20 percig (!) hibamentesen számolt
- ballisztikus számítá-sokhoz használták
- 5000 * gyorsabban számolt mint az ember
- huzalozott programo-zású volt
- Neumann Jánost kérték meg, tekintse át
- Javaslata: nem fix huzalozás !

A számítógép jellemzői (1)

- elektronikus (elektronikus eszközökből áll)
- digitális (diszkrét állapotok jellemzik)
- automatikus (külső beavatkozás nélkül működik)
- tárolt programú (eleinte fix huzalozású, majd számvezérlésű)
- programvezérlésű (olyan berendezés, amely végesszámú, különféle műveletfajta végrehajtására alkalmas. Ezen műveletek elemeiből kell egy összetett folyamat végrehajtására szolgáló időrendi vezérlést készíteni, a programot.)

A számítógép klasszikus funkcionális rendszervázlata



Meghatározás

- Számítógép: olyan technikai rendszer, amely adatok, információk feldolgozására képes, közvetlen emberi beavatkozás nélkül a benne letárolt utasítások alapján.
- Erőforrás: a rendeltetésszerű használathoz szükséges komponensek összessége.

Erőforrások (1)

Hardware

A számítógép fizikai megvalósítása

1. Központi egység (CPU)

2. Központi memória (adatok és utasítások tárolására)

3. Áramkörök az adattovábbításra (sin, busz,…)

Software

Programok, utasítások összessége

1. Rendszerszoftverek (a szgéppel együtt megvásárolhatók, a felhasználó munkáját könnyítik)

a) Vezérlő szoftver (operációs rendszer, amely az erőforrások optimális kihasználtságát maximalizálja)

b) Feldolgozó szoftver (szövegszerk., segédprg., ...)

Erőforrások (2)

4. Perifériák:

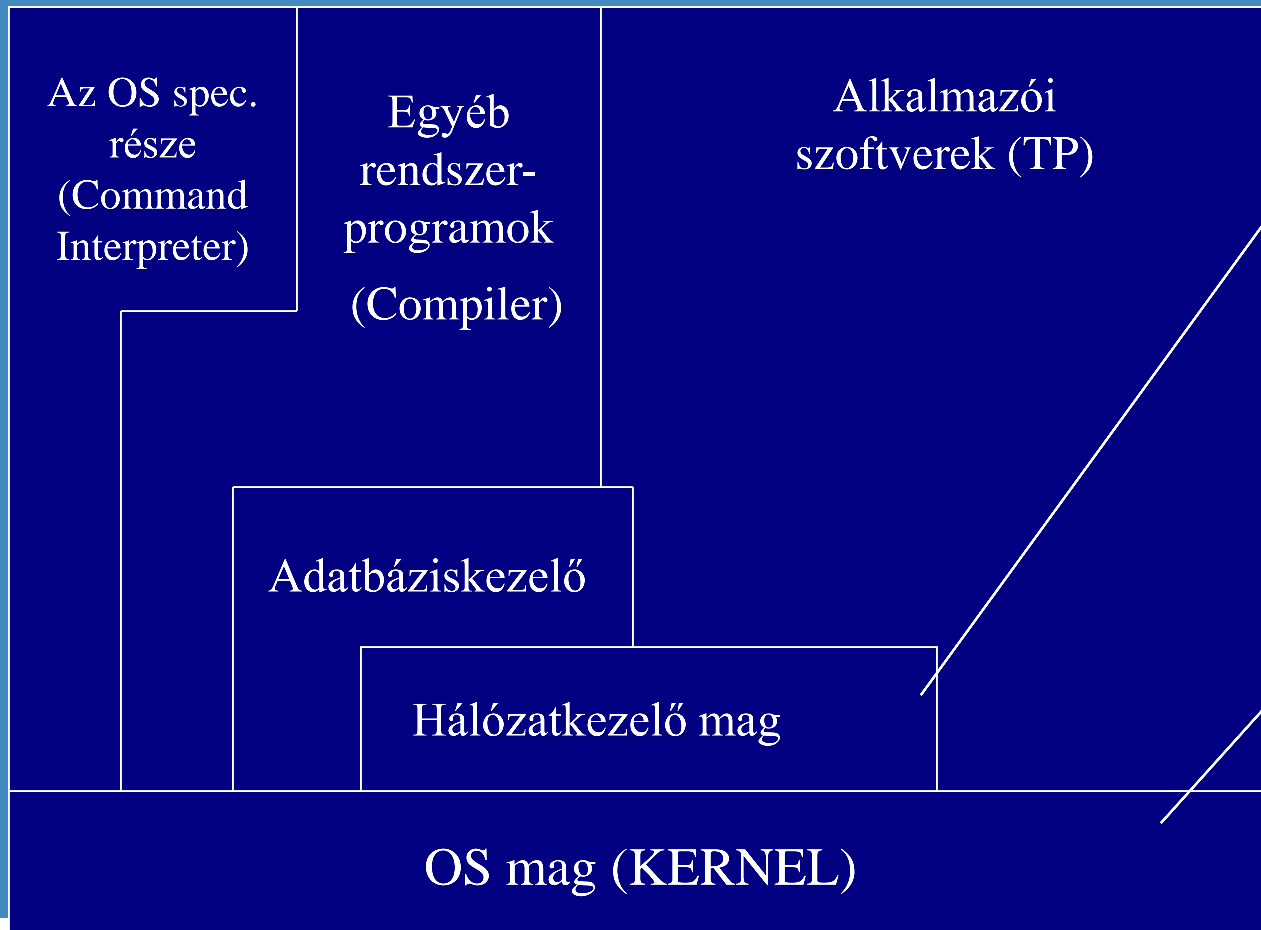
a). Tömegtárolók (mágneslemez,.)

b). Kapcsolattartás a felhasználóval (keyboard, display, mouse, tablet, plotter, printer,…)

c). Más rendszerekkel való kapcsolattartás eszközei (hálózati kártya, modem, ...)

2. Alkalmazói szoftverek (egyedi célra írottak, pl.: Word, Photoshop, …)

Erőforrások (3)



Kis programok, az OS-sel tudnak kapcsolatot tartani

Programok együttese, a hardware-t kezelik

A számítógépes feladatmegoldás eszközei

Adatok

(Amiken utasításokat hajtunk végre)

Utasítások

(Amiket végrehajtunk)

Program struktúra

Általános INFORMATIKAI Tanszék

Adatok

- Konstans (a programon belül végig állandó)
- Változó (a program során más és más értéket vehet fel)

- Adattípusok (felvehető értékük szerint)
  - numerikus
    - egész
    - valós
  - szöveges
    - karakter
    - sztring
  - logikai

Algoritmus (definíció)

Egy meghatározott cél elérésére irányuló, egymástól elkülönített tevékenységek alkalmas sorozata, melynek segítségével bizonyos kiindulási helyzetből, végesszámú közbenső állapoton keresztül, előírt feltételeknek eleget tevő véghelyzethez jutunk.

Példa
egyszerű numerikus algoritmusra

$c = a + b$

- vedd a-t
- vedd b-t
- képezd a+b -t
- legyen c = a+b -vel
- tedd c-t
- vége

Numerikus algoritmus jellemzői

- Diszkrét jellegű: Véges sok elemi lépés. (A leírása!!)
- Determinált (meghatározott): Mindig tudjuk, a következő lépésben mi lesz a teendő.
- „Elemiek” a lépések: Már nem függ külső paraméterektől.
- Célra irányított (teljes): Van egy értékrendszer, ami ezt a célt kifejezi. Ezt akarjuk elérni.
- Véges: Véges számú lépésben eléri célját. (A megoldása!)
- Egész feladatosztályra érvényes: A bemenő adatok értelmezési tartománya: minden lehetséges adat.

Algoritmus leírási módok (1)

- természetes nyelv (vagy más néven verbális.) Gyors, mindenki által könnyen érthető, de pontatlan.

Pl.: kerékcsere:

Labilis mozgást érzékelek; jelzek jobbra;leállok; kulcs kikapcs; kézifék be.

Amíg nem szállhatok ki, hátranézek.

Kiszállok, …, szükséges szerszámokat előveszem;

Dísztárcsa le, rögzítőcsavart lazít, …, emelek, kereket le.

Ha van pótkerék, akkor:

kiveszem pótkereket, kereket be, pótkereket fel,…,légnyomást ellenőriz. Ha megfelel semmit, különben

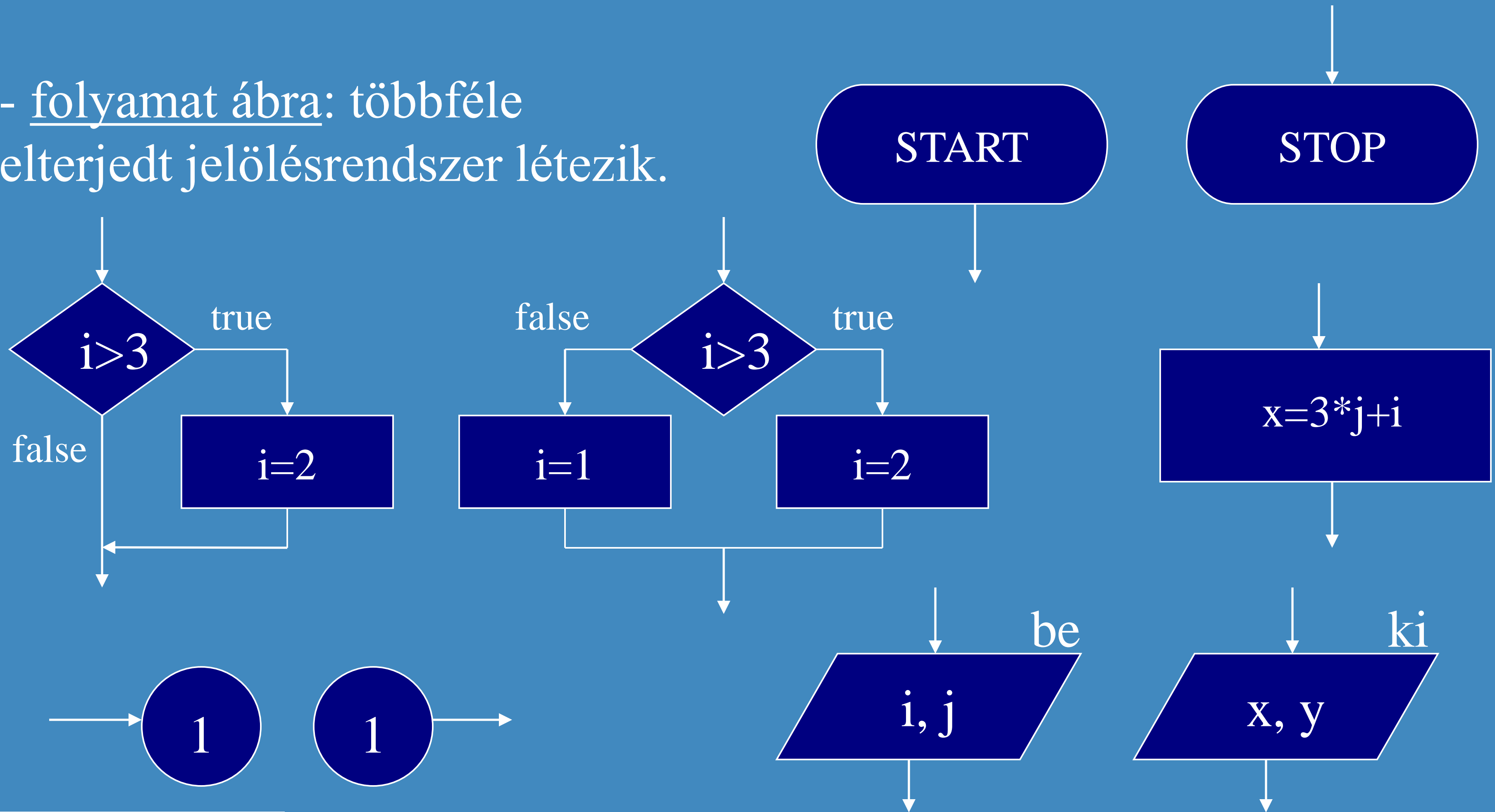
míg meg nem felel pumpál, ellenőriz….

Ha nincs, ragaszt, majd pumpál, stb.. , majd elindul

Általános INFORMATIKAI Tanszék

Algoritmus leírási módok (2)

- folyamat ábra: többféle elterjedt jelölésrendszer létezik.



Algoritmus leírási módok (3)

Pszeudonyelv: elemi utasítások összessége. Olyan vezérlő vagy keret utasítások, amelyek az algoritmus logikai vázát adják. (Pszeudo azért, mert nem programnyelv, hanem viszonylag kötetlen, programnyelvszerű jelölésrendszer. Gyakran emlegetik **PDL**, azaz **P**roblem **D**efinition **L**anguage - probléma leíró nyelv - néven is.)

Pl.:

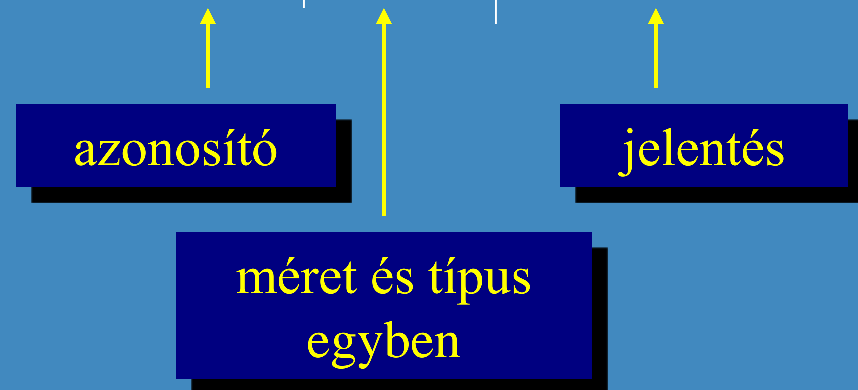
```
program átvált
olvas dollár
forint = dollár * árfolyam
kiír forint
vége átvált
```

Algoritmus leírási módok (4)

- Hiánya: kódoláskor nem derül ki az adatokról semmi (méret, cél, típus), ezért a PDL leíráshoz mindig hozzátartozik egy **ADATLEÍRÁS**. Felépítése:
  - azonosító
  - méret
  - típus
  - jelentés

- Pl.: spoz = $\sum x_i$ (i=1...n), ha $x_i > 0$

n	integer	az elemek száma
spoz	real	a pozitív elemek összege
x(n)	real	az adatok tömbje
i	integer	ciklusváltozó



Algoritmus leírási módok (5)

Metanyelv (a „nyelv nyelve”):

> Sajátos szókészlettel és nyelvtannal rendelkező nyelv, melynek segítségével egy nyelv formai és tartalmi elemzése elvégezhető.

Pl.:

Backus-Naur Forma (BNF)

Egy szimbólumrendszer a PASCAL nyelv leírásához

Szimbólumok:

- ::= szabály (definíció szerint)
- | választás (vagy)
- {…} ismétlés nullaszor vagy többször
- <…> szintaktikai egység

Algoritmus leírási módok (6)

Bármely magasszintű programozási nyelv: PL/1, ADA, FORTRAN, APL, C, C++, JAVA, PASCAL, LISP, stb…

```
PROGRAM  CNSUME
   CHARACTER  * 50  BUF
   DATA  NEXT   /1/,  BUF /' '/
   BUF(NEXT:NEXT)  =  C
   IF  (C  .NE.  ´ ´)
         WRITE(*,´(A)´) BUF
         SAVE  BUFFER,  NEXT
   END IF
END
```

A program

Számítógépi program:

- kezelt adatok leírása
- elvégzendő tevékenységek

Programozási nyelv:

szimbólumrendszer, melynek segítségével programot írhatunk.

Progr. nyelv fő összetevői:

- karakterkészlet
- nyelvi elemek
  - kulcsszavak, operátorok
  - azonosítók, konstansok, utasítások
- szintaktika (nyelvtan)
- szemantika (jelentéstan) (**A repülőgép ezután hajszálait tépdesve kockásan ráugrott a levegőre**.)

A C nyelv története

- 1963: BCPL (Basic Combined Programming Language).
- Kifejlesztője: Martin Richards
- 1970: B nyelv (B – Bell)
- Kifejlesztője: Ken Thompson (AT&T Bell Labs)
- Célja: a Unix OS kernelének megírására
- Nem volt elég hatékony
- 1971: C nyelv
- Kifejlesztője: Dennis Ritchie

A C nyelv története (2)

- Nem volt pontos leírása a nyelvnek.
- Használni csak az AT&T által készített UNIX-okban lehetett, tanulása autodidakta módon
- Feladat: lehessen más környezetbe is implementálni
- Megoldás:

  Kernighan – Ritchie: The C Programming Reference
  (1978, magyarul: Műszaki Könyvkiadó, 1985)

A C nyelv története (3)

- Nem volt teljes körű a definíció, ezért elég eltérő változatok készültek el.
- Probléma: a különböző gépeken készült C programok máshol nem voltak „futtathatók”
- Cél: egységesítés, a portabilitás érdekében
- Megoldás:
  1983: ANSI → X3J11 bizottság (tagjai különböző számítógépes cégek alkalmazottjai)
- 6 évvel később: **ANSI C X3.159-1989**

ANSI C

Kidolgozásának fontosabb szempontjai:

- A forráskód legyen hordozható
- De nem baj, ha nem hordozható
- Tartsa meg a C szellemiségét:
  - Bízz a programozóban!
  - Ne akadályozd a programozót a munkájában, hiszen ő tudja mi az, amit tennie kell!
  - A nyelv legyen kicsi és egyszerű!
  - Valamely művelet elvégzésére csak egy utat biztosíts!
  - Tedd gyorssá a programot, még akkor is, ha ez nem biztosítja a kód hordozhatóságát!

A C nyelv

Érvényes karakterkészlet:

A…z ! ” # % . & 0…9 `

( ) + - * / , ; < >

= ? [ ] \ { } | ~ ^

Azonosítók

Mire: konstansok, típusok, változók, függvények jelölésére.

Szintaxisuk: betűvel kezdődő alfanumerikus karaktersorozat.

Az _ (aláhúzásjel) betűnek számít, de azzal NEM kezdődhet! (…)

Hatáskörük: használatuk hatáskörükön belül egyértelmű és kizárólagos. (később pontosítjuk)

A kis és nagy betűk megkülönböztetődnek!

pl.: x, y, x2, valtozo, Valtozo, var5, Osszeg, …

(Standard függvény azonosítók

pl.: printf, scanf, exp, cos, acos, fopen, fgetc,…)

Fontos! Vannak foglalt azonosítók → kulcsszavak

pl.: auto, break, const, do, if, int, static, void, while, …

Megjegyzés (komment)

```
//  Ez egy megjegyzés, és nem lehet több soros.
//  Ha több soros kell, akkor minden sor
//  elejére ki kell tenni

/* ez is egy megjegyzés,
   de ez lehet több soros is,
   mert van lezárója */
```

Operátorok

! != % %= & && &= ~ * *=

+ ++ += - -- -= -> . / ,

/= < << <= <<= = == ^ ^=

> >> >= >>= [ ] | |= ||

sizeof

C program szerkezete (1)

- Blokkokat használunk
- A C program függvényekből áll
- Minden programban lennie kell 1 db main nevű függvénynek
- A függvények meghívhatják egymást
- Meghívhatják önmagukat is (rekurzió)
- Mindenről nyilatkozni kell (definiálni illetve deklarálni kell)

C program szerkezete (2)

Jellemző sorrend:

- deklarációk
- main() függvény
- függvénydeklarációk

Általános INFORMATIKAI Tanszék

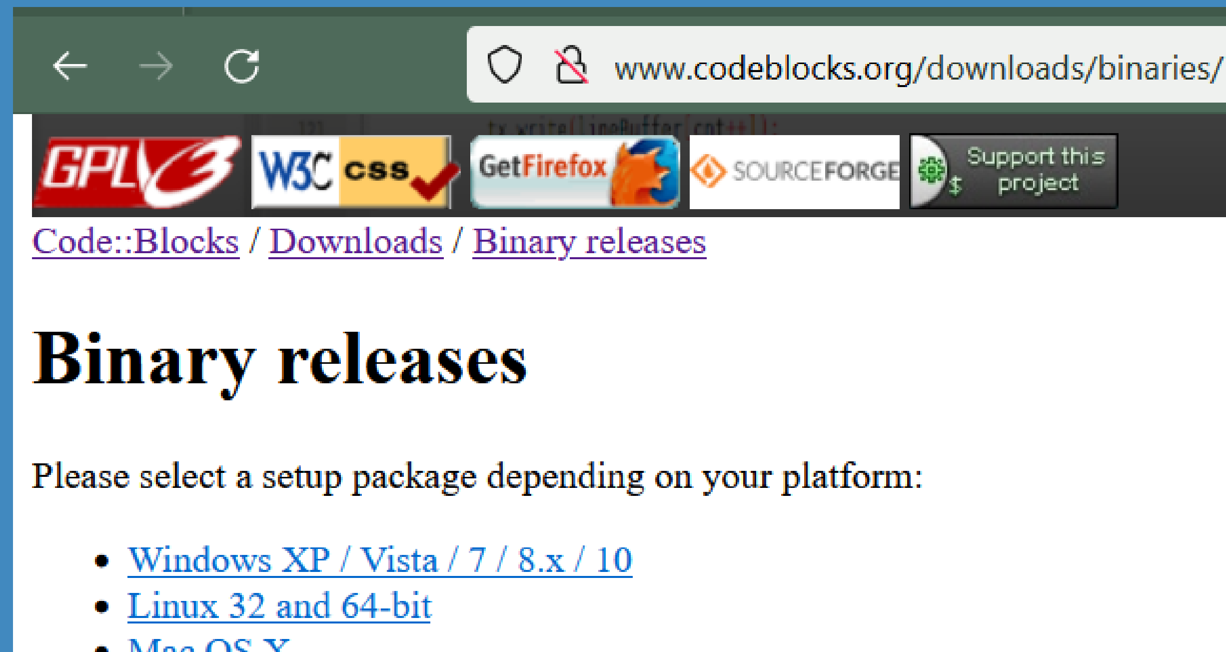
Példa program

```
#include <stdio.h>

int main (void)
{
    printf ("indulunk...");
    return 0;
}
```

Javasolt PC-s fejlesztőrendszer

- DEV C++ 4.9.9.2 (ingyenes)
  - Hosszabb ideje nem fejlesztik
- Code Blocks (ingyenes) (codeblocks.org)
  - Folyamatosan javítják,
  - Létezik több különböző platformon is



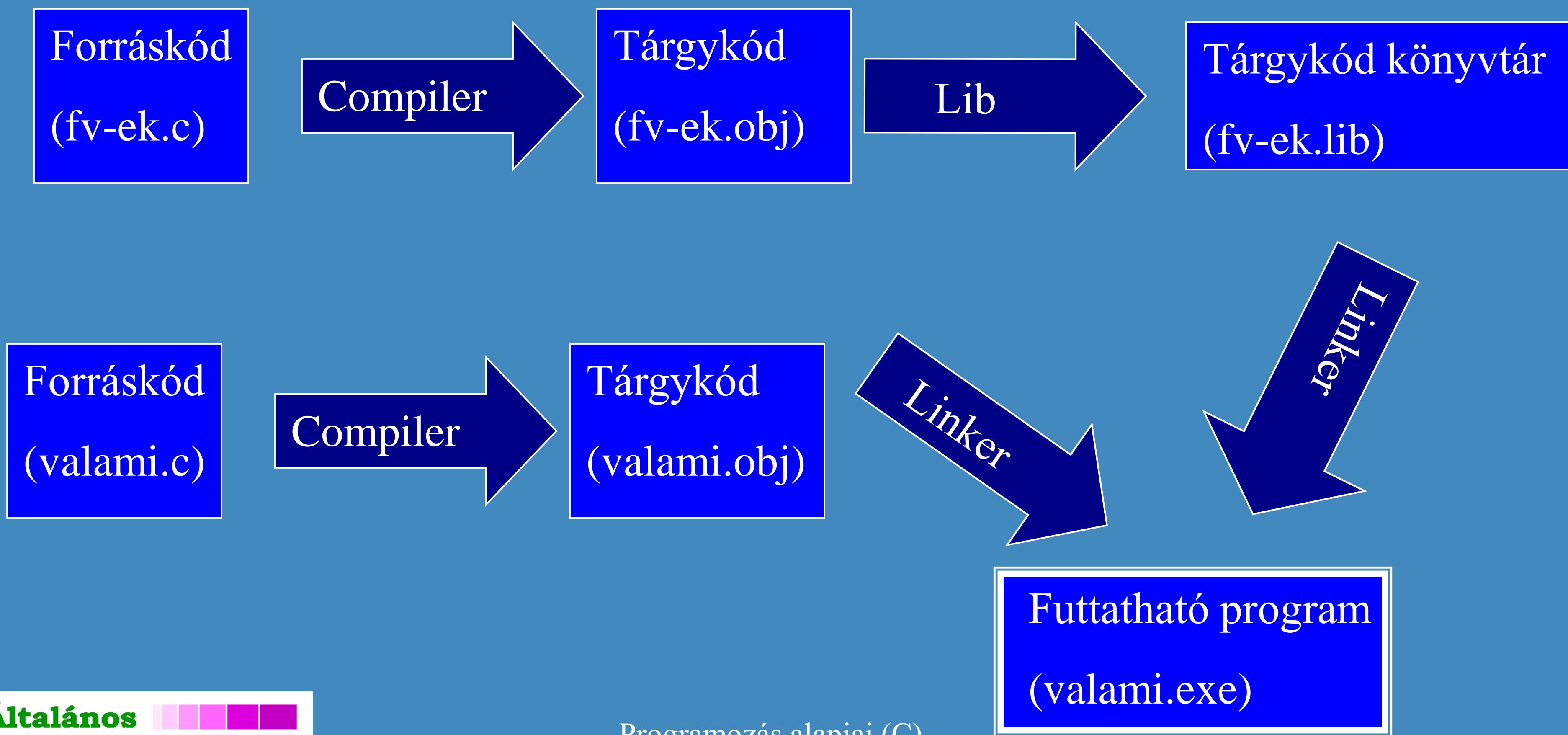
Általános INFORMATIKAI Tanszék

PC-s memóriamodellek

- Tiny
- Small
- Medium
- Compact
- Large
- Huge

(Windows alatt ezek már nem használatosak)

Egy futtatható program előállításának menete



Általános INFORMATIKAI Tanszék

Konstans definíció

- Azonosítót rendelünk konstans értékhez, kifejezéshez. Ez az azonosító - a konstans neve - szolgáltatja a fordítás alatt a konstans értéket.

A program futása alatt nem változtatható!

Konstans lehet

- Szám:
  - decimális egészek (int): 325, +325, -400, 0 (nem nullával kezdődik)
  - Hexadecimális egészek: 0x5F
  - Oktális egészek: 037 (nullával kezdődik, utána 0..7 közötti érték lehet)
  - valósok (real):
    - fixpontos: 0.1, +0.1, -0.012
    - lebegőpontos: 0.1E0, +0.1e-1, 0.0012E+1
- Karakter: ’a ’ ’ 1 ’ ’ ab ’ (aposztrófok közé írt 1 vagy több karakter) a több karakteres konstans nem szabványos!!
- Karakterlánc (sztringkonstans): (idézőjelek közé írt karakterek) ”ez egy meglehetősen hosszú sztring” ”ez rövid”

Utasítás

- <u>Függvényhívó utasítás</u>: (pl.: scanf(), printf(),…)

- <u>Értékadó utasítás</u>: (az a tevékenység, melynek során egy változó értékét módosítjuk)

  <változó> = <kifejezés>; ← **Nem BNF !!**

  kifejezés: (konstans vagy változó) operandusokból és operátorokból álló kiértékelhető szerkezet.

  (pl.: a = 30.0 * 3.1415 / 180.0;)

Változó deklaráció

- A **memória egy része**, ahol a program futása közben más - és más (változó) értékek jelenhetnek meg.

Minden változónak van:

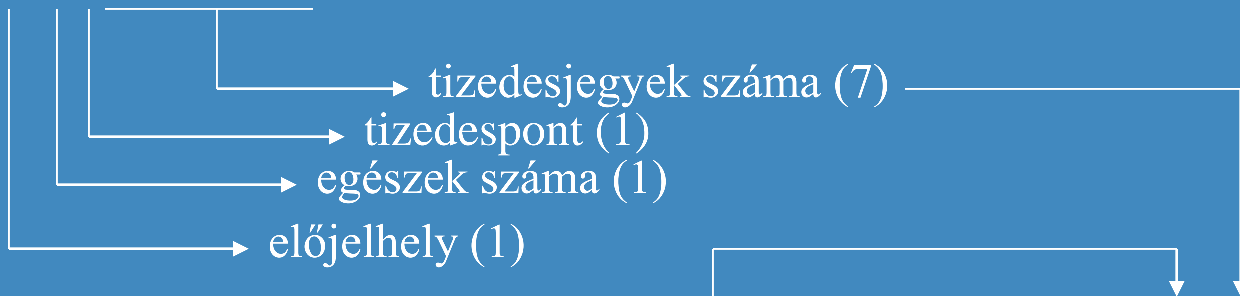
- **Neve**: a tárolóhely címére utal. Ez a cím egyelőre elérhetetlen.
- **Címe**: a memória területet azonosítja.
- **Típusa**: a névvel címzett memóriarész értelmezése (a programozó szemszögéből!)
- **Aktuális értéke** (tartalma): a névvel címzett memóriarész (tároló) tartalma egy adott időpillanatban.
- **Érvényességi köre**: a program azon része, amelyben a változóra a nevével érvényesen hivatkozni lehet.

Valós számok kiíratása (1)

<u>A mezőszélesség fogalma</u>:

Azoknak a karaktereknek a száma, amelyek egy szám adott tizedesjeggyel történő kiíratásához szükségesek.

Pl.: **3.1415927**



<u>Mezőszélesség</u>: 1+1+1+7=**10**

```
printf("Pi = %10.7f", a);
```

Valós számok kiíratása (2)

A tizedesjegyek számának maximális értéke a választott típustól függ.

Ha a tizedesjegyek számának megadása elmarad, akkor még duplapontos típus esetén is csak 6 tizedesjeggyel történik a kiírás.

Pl.: 0,1234567890123456 ⟶ 0,123457

[|-] <egész> . <tizedesjegyek> e [+|-] <kitevő>

1 1 1 min. 1 1 1 3 = min ∑9

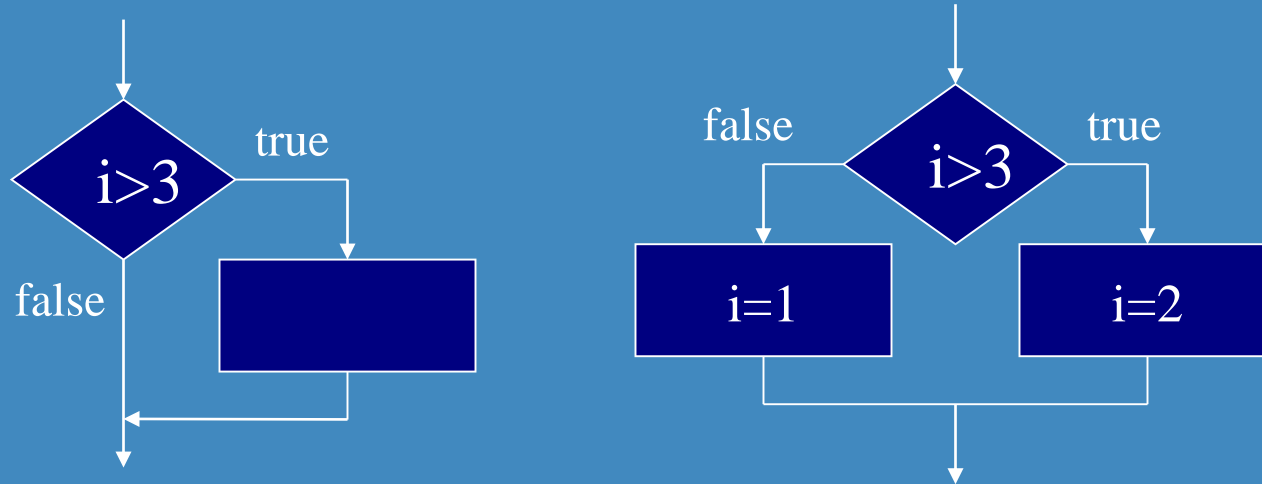
Lebegőpontos ábrázolási forma esetén a mezőszélesség minimum 9 karakter, és a tizedesjegyek számának növekedésének megfelelően nő.

A mezőszélességet célszerű kicsit megnövelni. Így az egymás melletti számok nem olvadnak össze.

Gyakorlás

- Egyelőre papíron, folyamatábra jelekkel oldjuk meg a feladatokat…

Vezérlési szerkezetek (1)



Működése:

if (logkif) … szerkezet esetén ha az if utáni logikai kifejezés értéke igaz, akkor a () utáni utasítás végrehajtódik. Egyébként az if UTÁNI első utasítás hajtódik végre.

if (logkif) … else … szerkezet esetén a logikai kifejezés hamis értékénél nem a következő utasítás, hanem az else utáni utasítás hajtódik végre.

Vezérlési szerkezetek (2)

if utasítás példa:

```
if (n<0)
        printf(”az érték negatív”);
```

if else utasítás példa:

```
if (n<0)
        printf(”az érték negatív”);
else
        printf(”az érték 0 vagy pozitív”);
```

Vezérlési szerkezetek (3)

Megjegyzések:

- Else ág esetében utasítás1 után VAN pontosvessző!
- Egymásba ágyazott If utasításnál vigyázni, hogy az Else ág hova tartozik!
- utasítás és utasítás1 csak 1(!) utasítást jelent.

Összetett utasítás

Összetett utasítás vagy utasítás zárójel:

```
{ utasítás; }
```

Az **összetett utasítás** 1 utasításnak számít, így többnyire olyan helyen használatos, ahol egyébként csak 1 utasítás lenne használható.

A **{ }** párban van, emiatt célszerű egymás alá írni.

If problémák (1)

if (a < b) **if** (a < d) utasítás1; else utasítás2;

Hova tartozik az else ? (Első, vagy második if ?)

if (a < b) **if** (a < d) utasítás1; **else** utasítás2;

1 utasítás, így a második if-hez tartozik

Így olvashatóbb:

```
if (a < b)
    if (a < d)
        utasítás1;
    else
        utasítás2;
```

Nem az számít, hova írom az else-t:

```
if (a < b)
    if (a < d)
        utasítás1;
else
        utasítás2;
```

If problémák (2)

```
if (a < b) {if (a < d) utasítás1;} else utasítás2;
```

Hova tartozik az else ? (Első, vagy második if ?)

```
if (a < b) { if (a < d) utasítás1; } else utasítás2;
```

Utasítás zárójel! Záródik az utasítás, tehát az elsőhöz tartozik.

Így olvashatóbb:

```
if (a < b)
{
    if (a < d)
        utasítás1;
}
else
    utasítás2;
```

Nem a tagolástól kerül jó helyre az else ág. Az csak segít a hibakereséskor!

Logikai kifejezés

Mik képeznek logikai kifejezést?

- Relációs operátorok bármilyen kifejezésekkel
  (a >= 5)
- logikai operátorok logikai kifejezésekkel.

Logikai értéket képviselhetnek:

- Konstansok (pl.: #define TRUE 1)
- Változók: (pl.: int Logikai;)

Ezek kiértékeléskor logikai értéket eredményezhetnek.

Relációs operátorok

== < > <= >= !=

2 operandusú

Precedenciájuk az aritmetikai operátorok után.

Precedencia

(A műveletek végrehajtásának sorrendje)

Alapszabályok:

1. Zárójelek közötti kifejezések kiértékelése, mintha a zárójelen belül egy operandus lenne. Mindig a legbelső zárójelen belül kezdődik meg a kiértékelés. **Pl.: (x + 1) < ((x - abs(x)) * 1.6)**
2. Azonos precedenciájú operátorok kiértékelése balról-jobbra történik. (Általában, bár a compiler optimalizációs célból átrendezheti a kiszámítást.) **Pl.: 7.0 + a - 1.0**
3. Különböző precedenciájú operátorok esetén a magasabb precedenciájú értékelődik ki előbb. (Precedencia táblázat!)

Pl.: 7 + a * 9 / abs (i)

Precedencia táblázat

( ) [ ] . ->

! ~ - ++ -- & * (type cast) sizeof — Jobbról balra

* / %

+ -

<< >>

< <= > >=

== !=

&

^

|

&&

||

?: — Jobbról balra

= += -= *= /= %= <<= >>= &= |= ^= — Jobbról balra

Logikai operátorok

&&: (ÉS, logikai szorzás)

||: (VAGY, logikai összeadás)

!: (NEM, logikai negáció)

A logikai operátorokat háromféleképpen szokás tárgyalni:

- igazságtáblával
- kapcsolóáramkörrel
- Venn-diagrammal (halmazzal)

Igazságtáblával

&&	F	T
F	**F**	**F**
T	**F**	**T**

||	F	T
F	**F**	**T**
T	**T**	**T**

!	F	T
	T	**F**

Vagy:

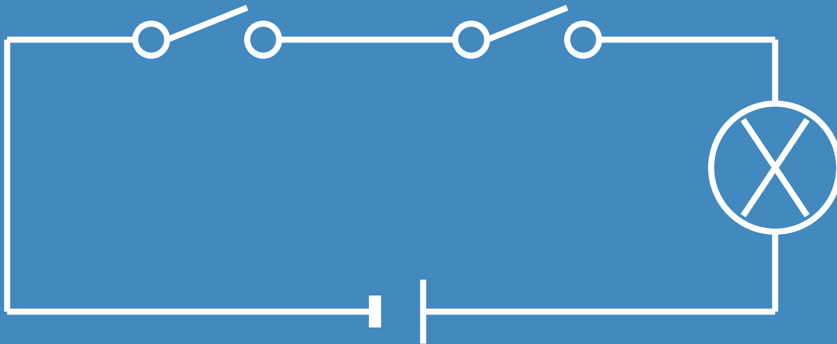
&&		
F	F	**F**
F	T	**F**
T	F	**F**
T	T	**T**

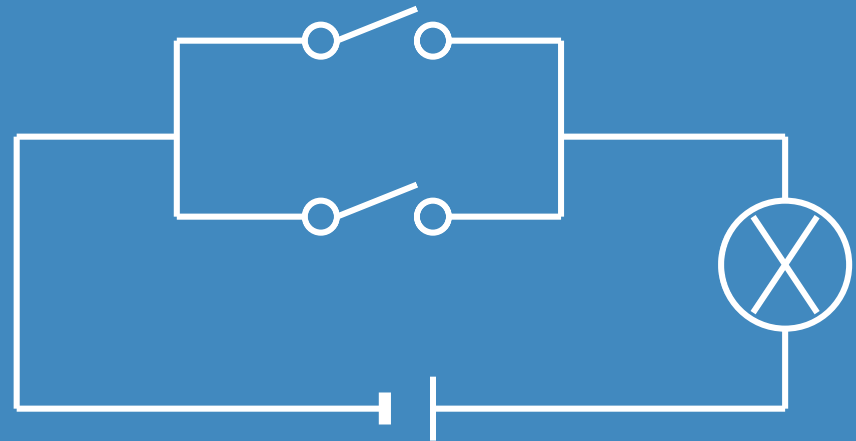
||		
F	F	**F**
F	T	**T**
T	F	**T**
T	T	**T**

!	
F	**T**
T	**F**

Általános INFORMATIKAI Tanszék

Kapcsolóáramkörrel





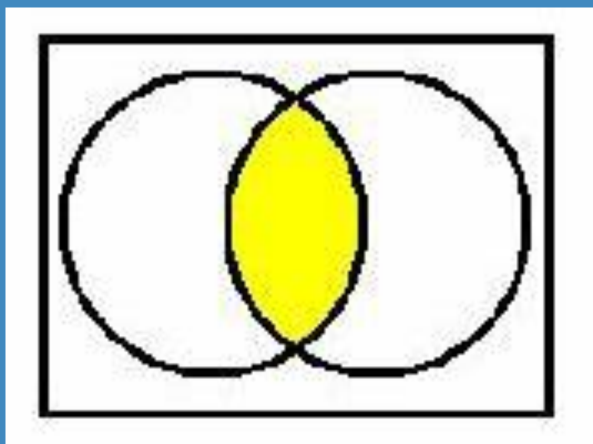
&&

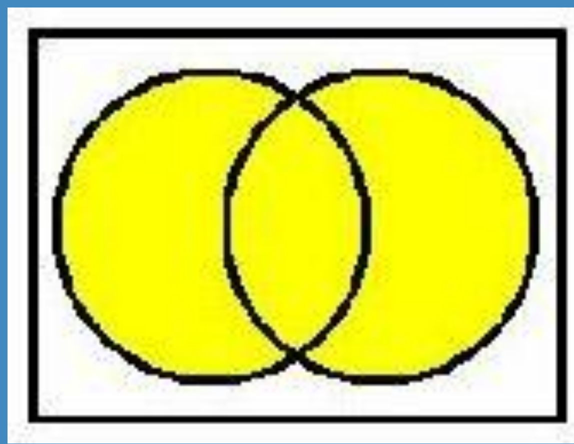
A lámpa ég, ha **mindkettő** be van kapcsolva

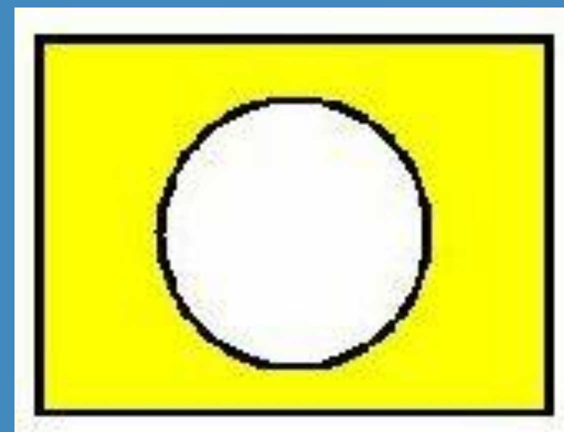
||

A lámpa ég, ha **legalább az egyik** be van kapcsolva

Venn-diagrammal







&& || !

Vezérlési szerkezetek (3)

while utasítás

```
while (logikai kifejezés)
        utasítás;
```

Működése: **mindaddig** ismétli a while utáni utasítást, amíg a zárójelben levő logikai kifejezés igaz.

Vezérlési szerkezetek (4)

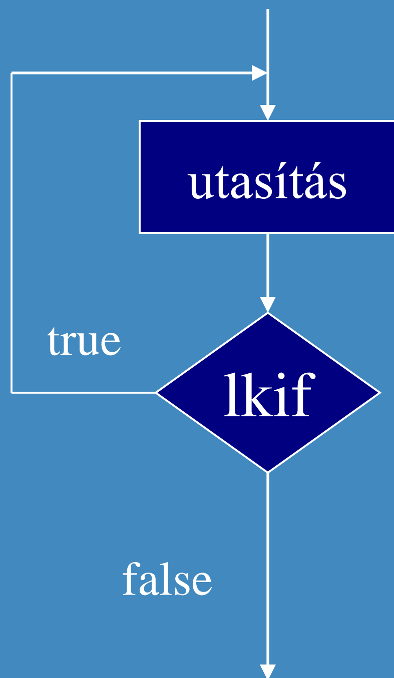
do while utasítás:

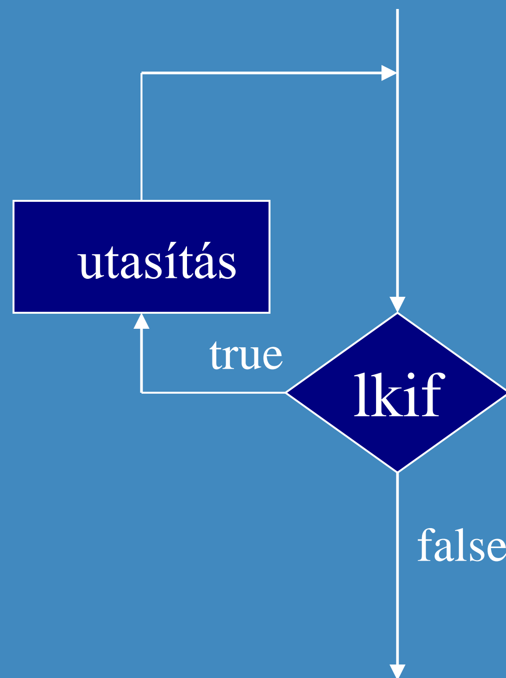
```
do
    utasítás
while (logikai kifejezés);
```

Működése: **Amíg** a while utáni logikai kifejezés értéke **igaz**, **addig** ismétli a do utáni utasítást.

Általános INFORMATIKAI Tanszék

Folyamatábra jelekkel





Példák

```
#include <stdio.h>
int main (void)
{
	char valasz;
	printf("Érthető?\n");
	do
		scanf("%c",&valasz);
	while (valasz != 'i');
	printf("Rendben\n");
	return 0;
}
```

```
#include <stdio.h>
int main (void)
{
	char valasz;
	printf("Érthető?\n");
	while (valasz != 'i')
		scanf("%c",&valasz);
	printf("Rendben\n");
	return 0;
}
```

Hibás!!! Úgy vizsgálja meg a valasz azonosítójú változó értékét, hogy az még nem kapott értéket!

Általános INFORMATIKAI Tanszék

Vezérlési szerkezetek

- Gyakorlás, írjunk kis programokat

Ciklusok (1)

Meghatározás: egy adott tevékenységsor többszöri, egymás utáni végrehajtása.

Fajtái:

- taxativ
- iterativ

Fajtái:

- elöltesztelő
- hátultesztelő

Ciklusok (2)

Részei:

- előkészítő rész
- ciklus mag
- végértékkel történő összehasonlítás
- döntés
- módosítás
- helyreállítás

Ciklusszámláló esetében **nem elfelejteni** a kezdőértékadást!

Még a logikai operátorokról

De Morgan-szabály:

Not (A And B) (Not A) Or (Not B)

Not (A Or B) (Not A) And (Not B)

Pl.: 5 és 10 közötti szám
beolvasása üzenettel:

Általános INFORMATIKAI Tanszék

Vezérlési szerkezetek (5)

A FOR utasítás

```
for ( init utasítás; feltétel; léptető utasítás)
    utasítás;
```

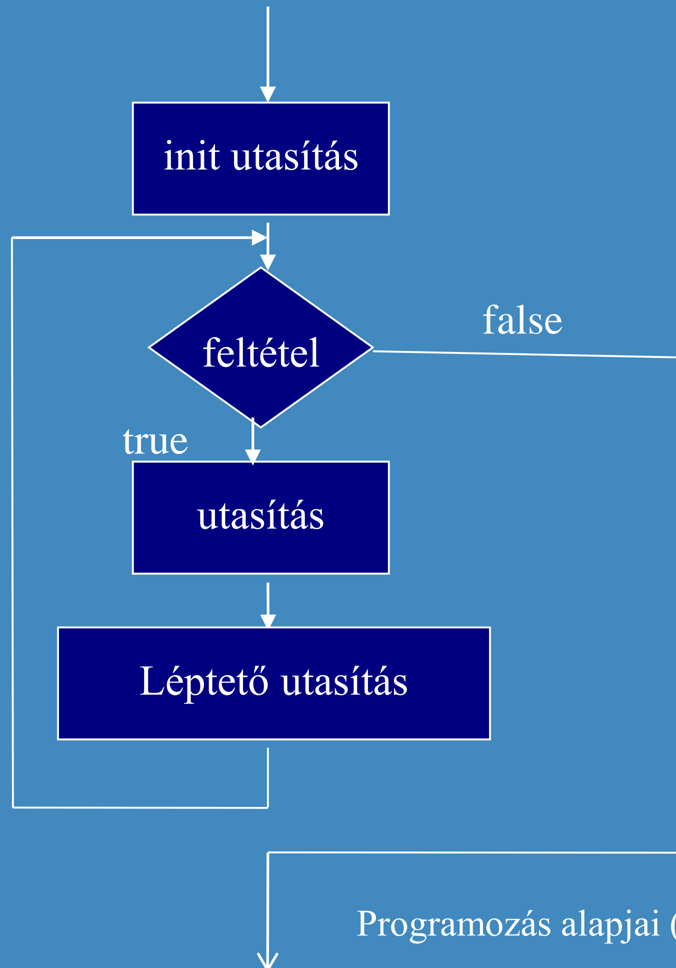
Bármelyik rész (akár mindhárom egyszerre) elhagyható.

Működése:

1. Végrehajtódik az init utasítás
2. Leellenőrződik a feltétel.
3. Ha igaz, vagy ha nem lett megadva, akkor először végrehajtódik az utasítás, majd azután a léptető utasítás. Ha hamis, akkor a program következő utasítása fog végrehajtódni.
4. Ezután ismétlés a 2. ponttól.

Vezérlési szerkezetek (6)

- Alkalmazási területe: elsősorban adatsorozatok feldolgozása esetén
- Folyamatábrával:



Példa

```
for (i=1; i<= 10; i++)
        printf(”i= %d \n”, i);
```

```
s=0;
for(i=1; i<= 100; i++)
        s=s+i;
```

```
for(s=0, i=1; i<=100; s=s+i, i++);
```

Tömbök (1)

Összetett adatobjektum, mely meghatározott számú komponensből áll.

A tömb bármilyen típusú lehet. (kivéve void)

Az elemek azonos típusúak.

Az összetett objektum neve: a tömb neve

Egyetlen komponense: a tömb egy eleme

A tömb elemeire indexek segítségével hivatkozhatunk.

Az index-ek típusa: egész.

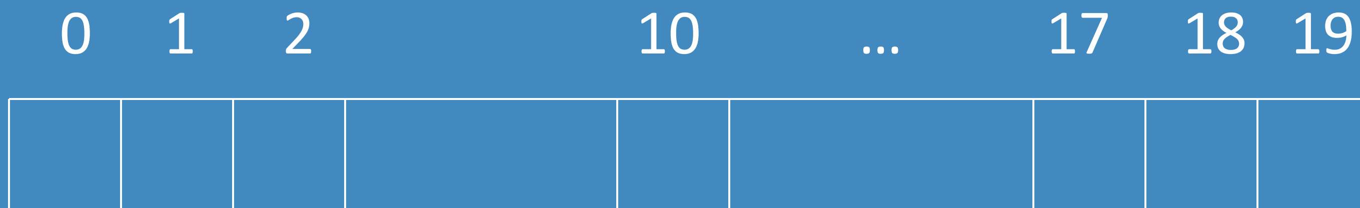
Az első elem sorszáma: 0

Deklarálása a változó deklarációs részben.

Tömbök (2)

Pl.: t : int t [10];

Pl.: double t [20];



Tömbök (3)

Hivatkozás a tömb elemeire:

tömbnév [indexkifejezés]

Sem az index-re, sem az index kifejezésre nincs ellenőrzés!

Pl.: t [2] = 3.1415;

t [i + 1] = t [i] - 5;

i értékét a rendszer előtte kiértékeli.

Ciklus esetén vigyázni! for (i = 0; i <= n -1; ...)

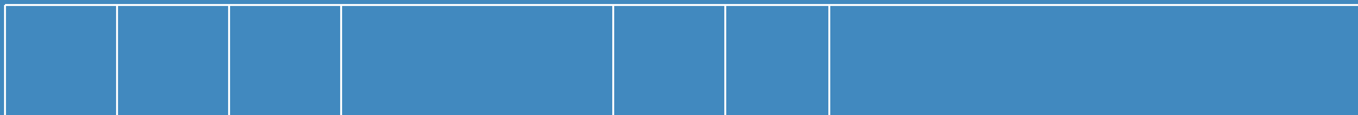
i = n-1 esetén az index n lesz!

Tömbök (4)

```
int i, t[10], j;
j=5;
for(i=1; i<=10; i++)   //Hibás, mert az első elem indexe valójában 0
    t[i] = 0;
```

Amikor i = 10, akkor valójában j értéke kerül módosításra!!

i t[0] t[1] … t[9] j

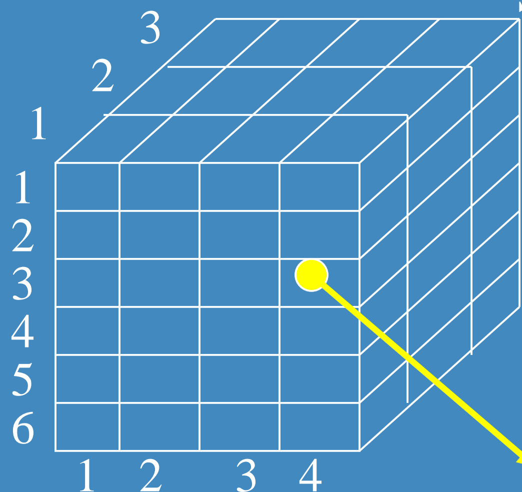


Több dimenziós tömbök (1)

double t[5];

double t[3][5];

Sor Oszlop



i = 0 … 5

j = 0 … 3

k = 0 … 2

t [2][3][0]

A több dimenziós tömbök a memóriában sorfolytonosan kerülnek tárolásra.

Több dimenziós tömbök (2)

Több dimenziós tömb kezeléséhez annyi egymásba ágyazott ciklusra van szükség, ahány dimenziós a tömb.

```
#include <stdio.h>

int main (void)
{
    int i, j;
    double t[3][4];
    printf("Kérem az adatokat \n");
    for (i=0; i<3; i++)
        for (j=0; j< 4; j++)
            scanf("%lf",&t[i][j]);
    return 0;
}
```


Általános INFORMATIKAI Tanszék

Alap algoritmusok

- Számlálás
- Összegzés
- Átlagolás
- Osztályba sorolás
- Minimum-maximum keresés

Számlálás

```
#include <stdio.h>
#define NMAX 100
int main (void)
{
     int i, n, db;
     double t[ NMAX];
     printf (”Kérem az elemek számát (Max 100): ”);
     do
             scanf(”%d”, &n);
     while ((n < 1) || (n > NMAX));
     db = 0;
     for (i = 0;  i < n; i++)
     {
             scanf (”%lf”, &t [ i ]);
             if (t [ i ] >= 0)
                  db = db + 1;
     }
     printf(”Összesen %d elem volt pozitív\n ”, db);
     return 0;
}
```

Általános INFORMATIKAI Tanszék

Összegzés

```
#include <stdio.h>
#define NMAX 100
int main (void)
{
    int i, n;
    double szumma, t[ NMAX];
    printf (”Kérem az elemek számát (Max 100): ”);
    do
        scanf(”%d”, &n);
    while ((n < 1) || (n > NMAX));
    szumma = 0;
    for (i = 0;  i < n; i++)
    {
        scanf (”%lf”, &t [ i ]);
        if (t [ i ] >= 0)
            szumma = szumma + t [ i ];
    }
    printf(”A pozitív elemek összege: %lf \n ”, szumma);
    return 0;
}
```

Átlagolás

```
#include <stdio.h>
#define NMAX 100
int main (void)
{
      int i, n, db;
      double szumma, t[ NMAX];
      printf (”Kérem az elemek számát (Max 100): ”);
      do
            scanf(”%d”, &n);
      while ((n < 1) || (n > NMAX));
      db = 0; szumma = 0;
      for (i = 0;  i < n; i++)
      {
            scanf (”%lf”, &t [ i ]);
            if (t [ i ] >= 0) {
                  szumma = szumma + t [ i ];
                  db++;
            }
      }
      printf(”A pozitív elemek átlaga: %lf \n ”, szumma / db);
      return 0;
}
```

Osztályba sorolás

Tipikus példa, év végi átlagok:

AH	FH	Osztály
2.00	2.50	elégséges
2.51	3.50	közepes
3.51	4.50	jó
4.51	5.00	jeles

Megvalósítása egyszerű esetben IF-ekkel, de ügyesebb, ha tömbben tároljuk az osztályok határait...

Minimum-maximum keresés

```
#include <stdio.h>

int main (void)
{
    int i, mini, t[100];
    for (i = 0;  i < 100; i++)
            scanf (”%d”, &t [ i ]);
    mini = 0;
    for (i = 1;  i < 100; i++)
            if  (t [ i ] < t [ mini ])
                 mini = i;
    printf(”A legkisebb elem indexe:  %d ,és értéke: %d \n ”, mini, t [ mini ]);
    return 0;
}
```

Cserélve kiválasztásos rendezés (1)

A minimum-maximum keresés elvére épül.

Ismétlés: minimum keresés

A halmazból egy tetszőleges elemet kinevezünk legkisebb elemnek (általában az elsőt),

majd az összes többit sorban összehasonlítjuk azzal.

Minden alkalommal, amikor kisebbet találunk, megjegyezzük annak a sorszámát, és a továbbiakban már azt tekintjük a legkisebbnek.

A keresés legvégén megkaptuk a legkisebb elem sorszámát.

Cserélve kiválasztásos rendezés (2)

A rendezés elve:

- A rendezett halmazban első elem a legkisebb elem.
- A második elem a második legkisebb, vagyis az elsőtől jobbra levő elemek közül a legkisebb.
- A harmadik a másodiktól jobbra levő elemek közül a legkisebb
- …
- Minden elem a tőle jobbra levő elemek közül (beleértve saját magát) a legkisebb.

Ügyelni! Az utolsó elemtől jobbra már nincs elem!!

Cserélve kiválasztásos rendezés (3)

Ciklus az 1. elemtől az utolsó előttiig

```
for (i = 0; i <= n – 2; i++)
{
```

Állítás

```
        mini = i;
```

Minimum keresés

```
        for (j = i + 1; j <= n - 1; j++)
                if  (t [ j ] < t [ mini ])
                        mini = j;
```

Csere

```
        s = t [ i ];
        t [ i ] = t [ mini ];
        t [ mini ] = s;
```

```
}
```

Pointerek (1)

Sok esetben (például beolvasásnál) szükség van egy változó címének megadására.

Emiatt szükség van olyan változóra, ami képes ezt a címet tárolni.

Az ilyen változókat pointereknek nevezzük.

Pointerek (2)

A változó címének képzése ismert:

&sum a sum nevű változó címét képzi.

Ha sum például double (double sum;) akkor annak a pointernek a típusa, ami képes ezt a címet tárolni:

```
double *p;
```

Ekkor: p = ∑ (erre főként a függvényeknél lesz szükség)

Általános INFORMATIKAI Tanszék

Pointerek (3)

Hivatkozás a pointer-ben tárolt címen levő értékre:

```
*p
```

Vagyis például:

```
c = 5;
p = &c;
c = *p + 1;  // c értéke 6 lesz
```

Pointerek (4)

Fontos!!

```
c = *p + 1;
c = *(p + 1);  // egészen mást jelent!!
```

Függvények (1)

- **Jelenség**: ha ugyanazt a műveletsort kell elvégezni más és más adathalmazon, akkor jelenleg többször ismétlődő programrészletet kell írnunk a programba.

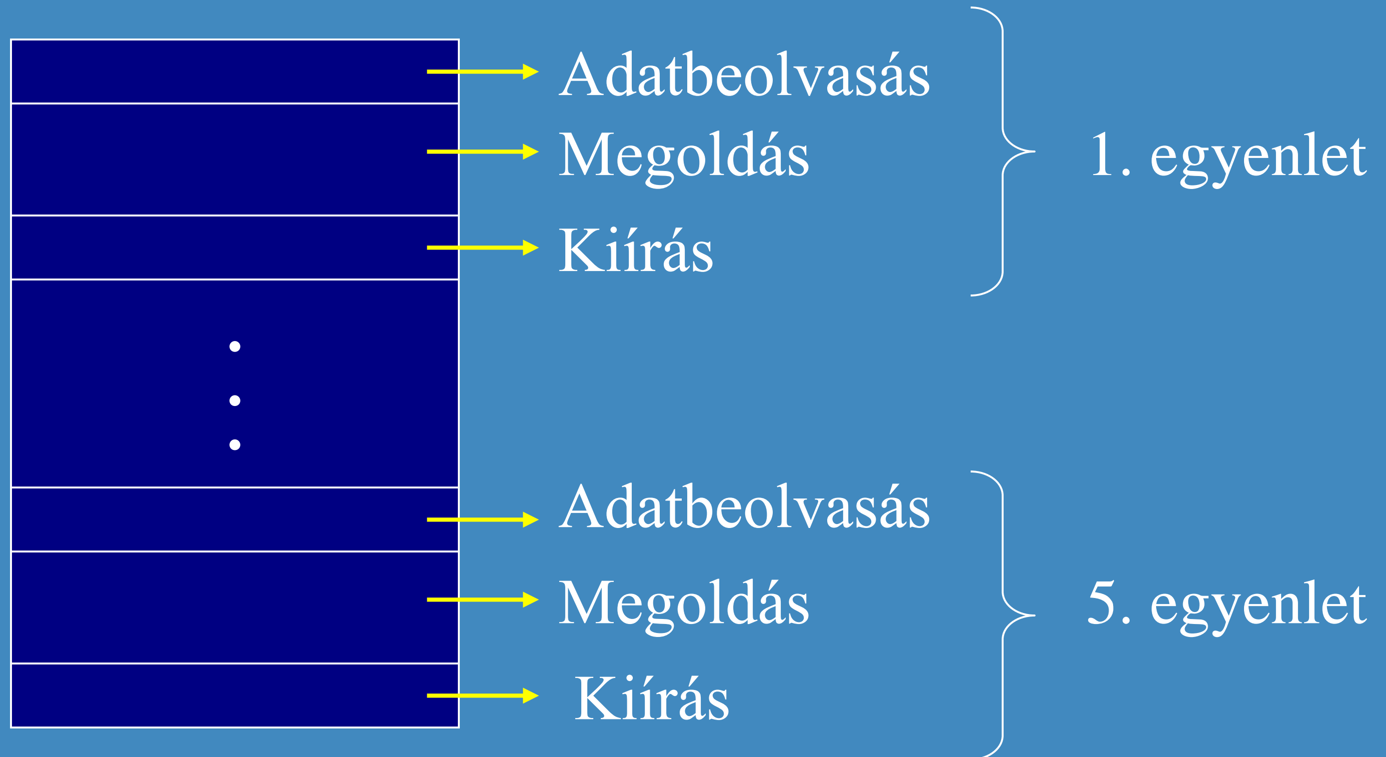
  Következmény:
  - sokat kell írni *(majd másolom az ísmétlődő utasításokat)*
  - hosszú lesz a program *(legalább látják mekkora munkám van)*
  - sok helyet foglal a file a HDD-n *(na bumm, elfér, nem?)*
  - tovább tart a program lefordítása *(kibírom.)*

  - véletlenül hibásan írtam. **Hány helyen is kell javítanom?**

Függvények (2)

Például: 5 db másodfokú egyenletet kell megoldani. Variációk:

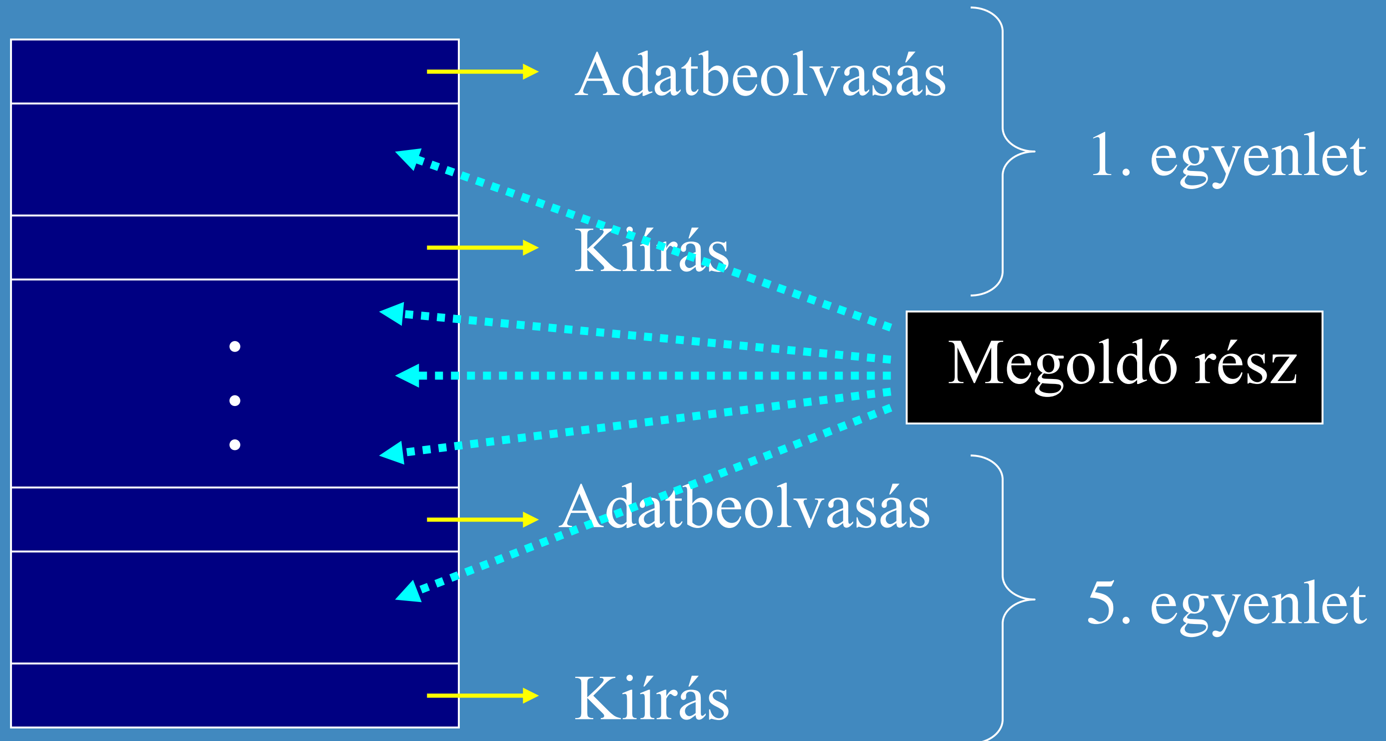
1. Nincs szubrutin



Függvények (3)

2. Nyílt szubrutin (makrók)

Szimbólummal jelölik, ez alapján hívják. A forrás programban csak egyszer van, de a lefordított programban annyiszor fordul elő, ahányszor meghívták.

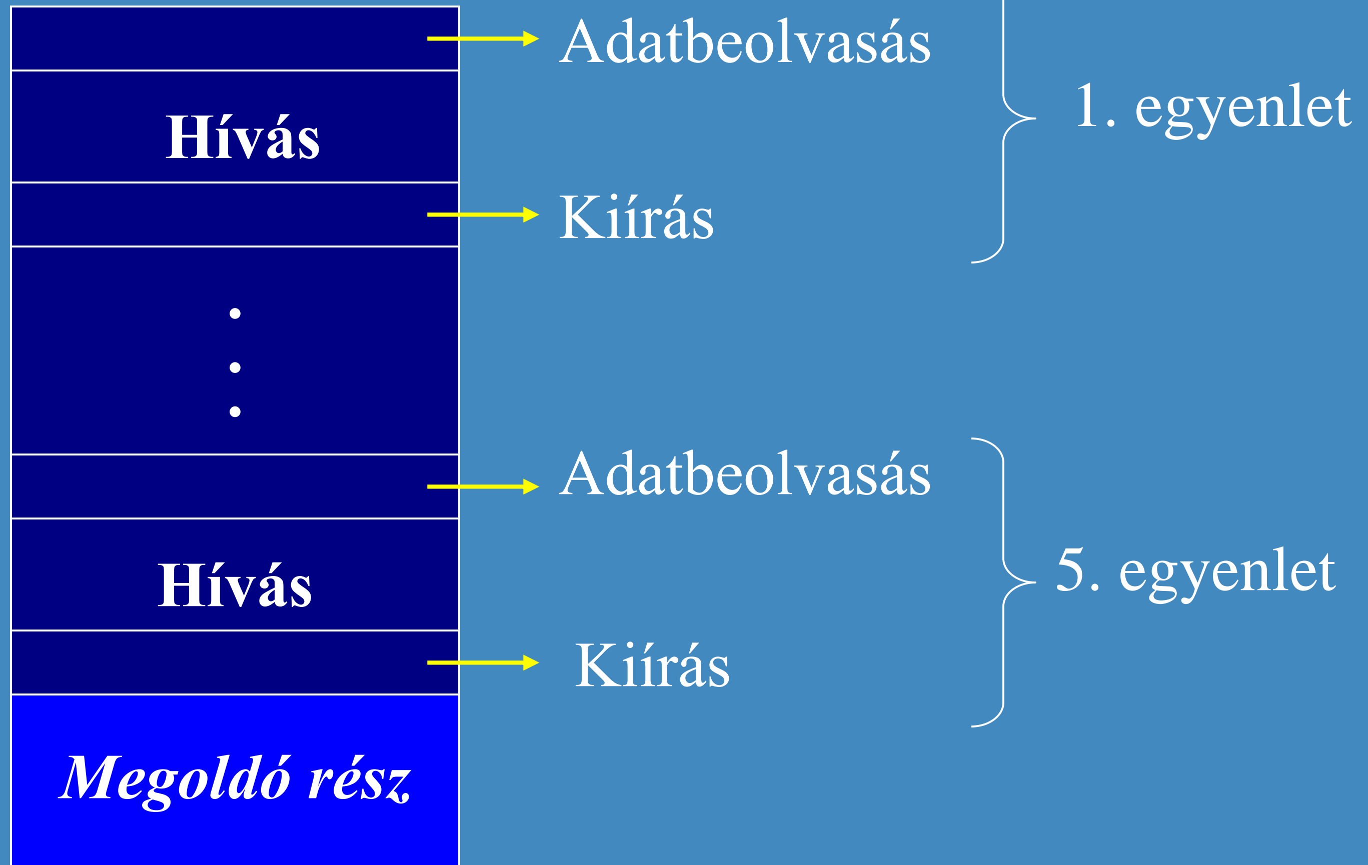


Függvények (4)

3. Zárt szubrutin

A hívó rutin területén kívül tárolódik. A forrásban, és a lefordított programban is csak egyszer fordul elő.

Kezdetben a tár fix helyén volt, később relokálható lett.



Általános INFORMATIKAI Tanszék

Függvények (5)

A függvénynek mindig van visszatérési értéke.

- vagy void (tudjuk, mit jelent),
- vagy egy adott típus.

Pl.:

int main (void)	Visszatérési érték típusa: int
double min (double, double)	Visszatérési érték típusa: double

Általános INFORMATIKAI Tanszék

Függvények (6)

- A függvények tetszőleges sorrendben követhetik egymást, az egyetlen kikötés: **első felhasználásuk előtt deklarálni kell őket**. (Ez egyedül csak olyan esetben probléma, amikor szubrutinok egymást hívják).
- A szubrutinok a program önálló részei, melyeknek a futás során át lehet adni a vezérlést. Ezt úgy nevezzük (zsargon): **meghívjuk a szubrutint**. Ekkor a szubrutin utasításai hajtódnak végre, majd a végrehajtás a hívó program **hívást <u>követő</u> első utasításán** folytatódik.

Függvények (7)

- Paraméterek:

  A szubrutinok paraméterek segítségével kommunikálhatnak környezetükkel. Az eljárások eredményüket is e paraméterek segítségével adják vissza a hívónak. **Nem globális változókkal dolgozni!!!**

```
típus FvNév (típus paraméter_név)
{    saját típusok;
     változók;
     utasítások;
     return …;
}
```

Formális és aktuális paraméterek

A függvények fejlécében felsorolt paramétereket FORMÁLIS paramétereknek nevezzük.

Azokat a paramétereket pedig, amelyekkel a függvényt meghívjuk, AKTUÁLIS paramétereknek nevezzük.

A formális és aktuális paraméterek darabszámának, és páronként típusának meg kell egyeznie!

Paraméter átadási módok

Kétféle paraméterátadási mód van:

- címszerinti
- értékszerinti

A paraméter átadás módját a függvény fejlécében lehet meghatározni.

Címszerinti paraméterátadás

A fv hívásakor a **paraméternek a címe** kerül átadásra. A szubrutin a cím alapján tudja módosítani az átadott paraméter értékét. Látszólag úgy tűnik, hogy a megváltoztatott érték a hívó programba való **visszatérés után** is megmarad.

A szubrutin fejlécében a paraméter neve előtt ott szerepel a *

Általános INFORMATIKAI Tanszék

Értékszerinti paraméterátadás

A fv hívásakor a szubrutin is helyet foglal a memóriában a változónak, ahova a paraméterként **megkapott értéket bemásolja**. Ha a függvény megváltoztatja ennek a paraméternek az értékét, akkor a hívó programba való visszatérés után a megváltoztatott érték nem kerül vissza, hanem **elveszik**.

Értékszerinti paraméterátadáskor a paraméter lehet konstans vagy kifejezés is.

Lefutása után a szubrutin az általa lefoglalt memória területeket felszabadítja.

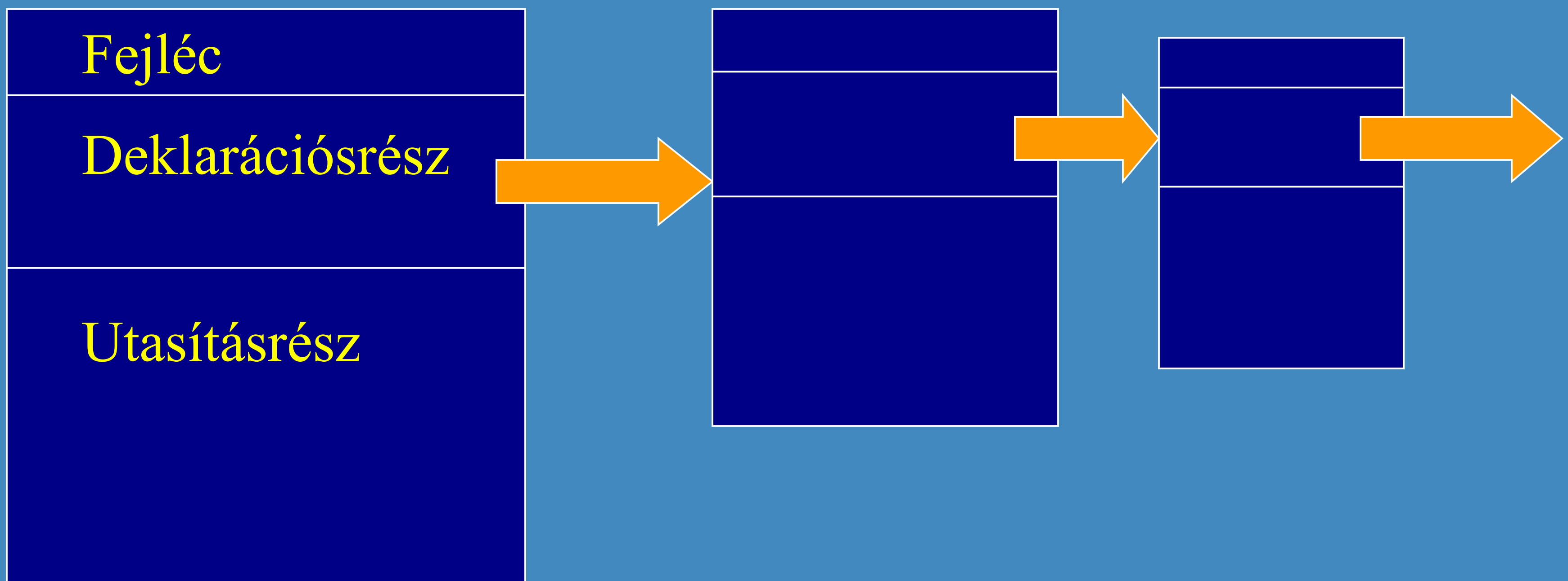
Összehasonlítás

- Címszerinti:
  - Előny:
    - gyorsabb paraméterátadás, gyorsabb futás
    - rövidebb a lefordított program
  - Hátrány:
    - ha a szubrutin megváltoztatja a paraméter értékét, akkor az visszakerül a hívóprogramba (nem hátrány, következmény!!)
- Értékszerinti
  - Előny:
    - konstansok is átadhatók, így egyszerűbb a meghívás
  - Hátrány:
    - Nagyobb futtatható programméret
    - lassúbb végrehajtás

Azonosítók érvényességi köre

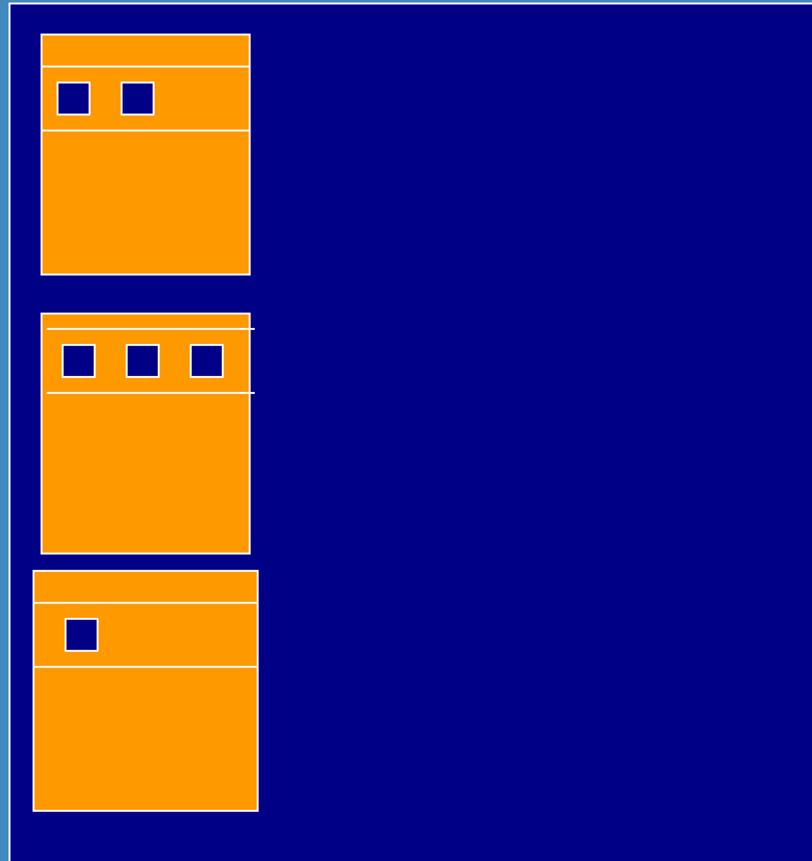
Kiindulási alap: a blokkszerkezetű programozási nyelvek (C, FORTRAN, PASCAL, …)

Egy program szerkezete:



Azonosítók érvényességi köre (2)

Minden egyes blokk tehát újabb blokko(ka)t tartalmazhat



Általános INFORMATIKAI Tanszék

Azonosítók érvényességi köre (3)

A blokk deklarációs részében azonosítókat lehet deklarálni.

Az azonosítók és a blokkok viszonyától függően háromféle érvényességi kört különböztetünk meg.

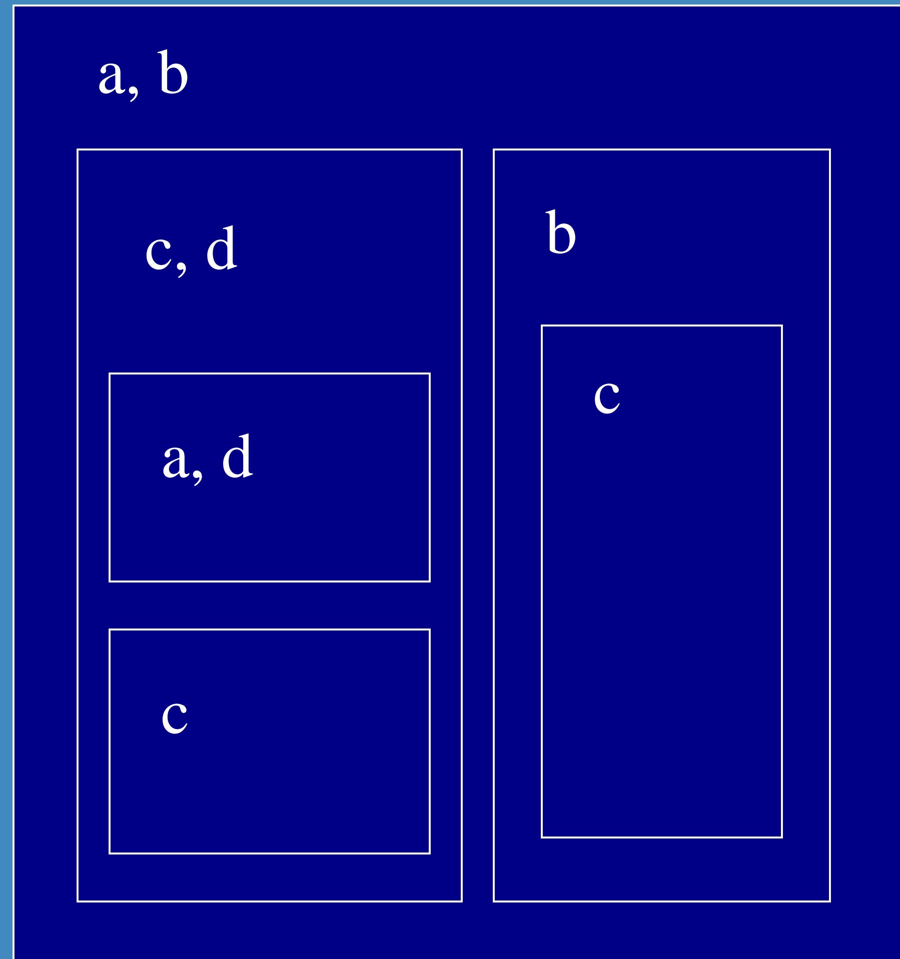
```
int j;
```

Azonosítók érvényességi köre (4)

- Lokális: ...
- Globális: ...
- Láthatatlan: ...

Bár globális azonosítókra időnként szükség van, de a globális változókat NE használjuk!!

Átláthatatlanná teszik a programot.



Általános INFORMATIKAI Tanszék

Felhasználó által definiált típusok

Helye: célszerűen azon a függvényen belül, ahol használni akarjuk. (Érvényességi kör!!)

Módja:

typedef <típusmódosító> <típus> <új típus neve>;

Pl.:

```
typedef unsigned long int hint;
```


Általános INFORMATIKAI Tanszék

Felhasználó által definiált típusok

Használata:

```
hint a;
scanf("%ld", &a);
printf("a= %ld\n", a);
…
a = 216;
…
```

A Char típus (1)

- Értelmezési tartománya: az ASCII karakterkészlet. 0-255-ig.
- Sorszámozva (!) tartalmazza a karaktereket, azaz minden karakterhez hozzá van rendelve egy (szám)kód.
- A karakterek egy-egy byte-ot foglalnak el a memóriában. E byte értéke az adott karakter ASCII kódja, megjelenési formája (pl. képernyőn való kiíratás esetén) a karakter képe.

A Char típus (2)

Használata:

```
…
char c;
c = 'x';
printf("c = %c\n", c);
scanf("%c", &c);
printf("c = %c\n", c);
…
```

Fontos: egyetlenegy karakter tárolására alkalmas!

A sztringek (1)

- Vagy másképpen: változó hosszúságú karakter sorozat. (De nem dinamikus módon történik a helyfoglalás.)
- Hossza: 0 .. 255 karakter
- Összetett típus

- String konstans: **“abc”**
- String változó:

```
char nev[30];
```

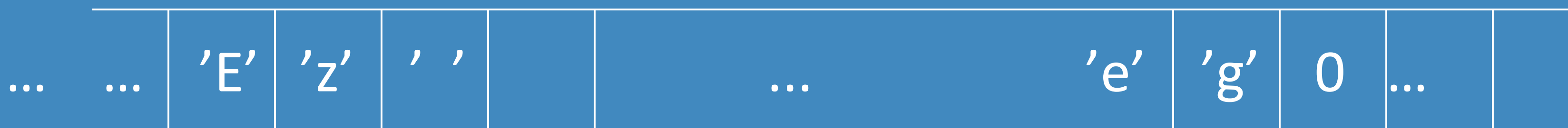
A C-ben (a Pascaltól eltérően) nincs önálló sztring típus, hanem karakter tömbként kell kezelni.

Egy fontos jellemzője van: mivel változó hosszúságú, a karaktersorozat végét valamilyen módon jelölni kell. C-ben: \0

A sztringek (2)

- Használatuk többnyire függvényekkel (#include <string.h>)
- Tárolása:

strcpy(**ss, “Ez egy példa szöveg”);**



karakterkódok

A sztringek (3)

- A tárolási mód miatt gondolni kell a véget jelző \0 karakterre is, emiatt ha a tárolni kívánt karaktersorozat 30 karakter hosszú lehet, a deklarálás ílyen kell legyen:

- char nev[31];

Összehasonlítás, értékadás, stb. függvényekkel történik.

Típus konverzió

Kétféle létezik:

- Automatikus

```
…
int a;
double b;
a = 18;
b = a;          // nincs probléma
b = a / 11; //probléma: b értéke 1 lesz
```

-Kikényszerített (typecast)

```
…
int a;
double b;
a = 18;
b = a;                // nincs probléma
b = (double) a / 11.0; // nincs pr.
```

Struktúrák

A tömbökről volt szó. Fontos jellemzőjük, hogy a tömb minden egyes eleme azonos típusú kell legyen.

A valós életben sokszor adódik olyan feladat, hogy összetartozó, de nem azonos típusú adatokat kell tárolni:

Név: karakter (sztring)

Fizetés: valós

Gyerekek száma: egész

Lakcím: karakter (sztring)

Struktúrák

- C-ben erre szolgál a struktúra (Pascal: record)
- Struktúra használata:
- Két részből tevődik össze:
  - Létre kell hozni a kívánt struktúra típust
  - Deklarálni kell ennek segítségével a változót

Struktúrák

```
struct s_nev {
        <típus> <tag>;
        <típus> <tag>;
        <típus> <tag>;
        …
        <típus> <tag>;
        };
```

```
struct ember{
        char nev[41];
        double fizetes;
        int gysz;
        char lcim[81];
        };
```


Általános INFORMATIKAI Tanszék

Struktúrák

```
struct s_nev  valt_nev;
```

Pl.:

```
struct ember dolgozo;
```

A kettő összevonható:

```
struct ember{
        char nev[41];
        double fizetes;
        int gysz;
        char lcim[81];
        } dolgozo;
```

Általános INFORMATIKAI Tanszék

Struktúra

Ha szükséges, lehet létrehozni új struct típust. Ekkor használni kell a typedef-et:

```
typedef struct ember{
        char nev[41];
        double fizetes;
        int gysz;
        char lcim[81];
        } EMBER;
EMBER dolgozo;  // ebben az esetben nem kell a struct kulcsszó.
```

Struktúra használata

```
dolgozo.gysz = 2;
dolgozo.fizetes = 236000;
strcpy( dolgozo.nev, “Nemeth Geza”);
strcpy(dolgozo.lcim, “Miskolc, Kiss utca 16.”);
```

Két azonos struktúra típusú változó közt megengedett:

```
dolgozo1 = dolgozo2;
```

de természetesen lehet tagonként is…

Beolvasás:

```
scanf(“%lf”, &dolgozo.fizetes);
```


Általános INFORMATIKAI Tanszék

A sizeof operátor

Több esetben van szükség arra, hogy egy adott változó mennyi helyet foglal el a memóriában.

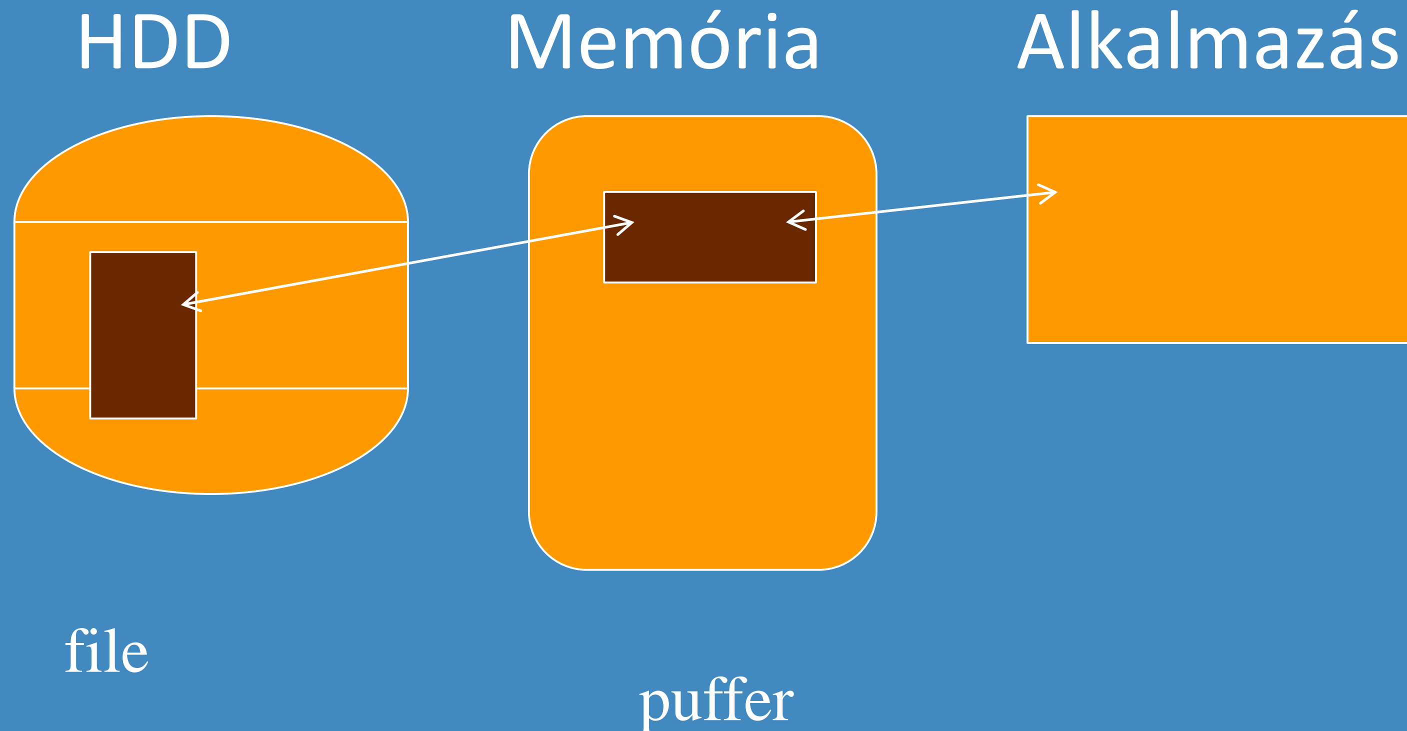
Mivel a változónak meg kell adni a típusát, ezért a helyfoglalása kétféleképpen kérdezhető le a sizeof operátorral:

- sizeof (típus)   például: n = sizeof (int);
- Sizeof változó   például: int k;

  n = sizeof k;

A sizeof eredménye egy előjelnélküli egész érték.

File kezelés C-ben (1)



Általános INFORMATIKAI Tanszék

File kezelés C-ben (2)

A file kezeléséhez szükséges egy speciális változó:
- Típusa: FILE *
- Ez egy struktúrára mutató pointer
- Az stdio.h-ban definiálva
- Tagjai:
  - File sorszáma
  - FLAG byte (utolsó művelet utáni állapot. Pl: EOF, vagy error)
  - Puffer címe
  - Pufferben levő byte-ok száma
  - file pointer (a file-on belüli következő adathoz), …

File kezelés C-ben (3)

Valójában a FILE * struktúra típusú tömbre mutat,

Elemeinek száma 20.

Ebből következik, hogy egyidőben maximum 20 file lehet nyitva

Az első 5 foglalt:

- stdin – standard input csatorna
- stdout – standard output csatorna
- stderr – standard error csatorna
- stdaux – standard aszinkron átviteli csatorna
- stdprn – standard nyomtató kimeneti csatorna

File kezelés C-ben (4)

A file-t a használat előtt meg kell nyitni!

Az első szabad tömb elembe kerülnek bele a most megnyitott file adatai.

A file megnyitás paraméterei:

- filenév az útvonallal
- megnyitási mód

File kezelés C-ben (5)

Filenév az útvonallal:

- Windows alatt: ”meghajtó:\\útvonal\\filenév”
- Unix alatt: ”\\útvonal\\filenév”

Megnyitási mód:

két adatot kell megadni:

- milyen jellegű a file (text, bináris)
- milyen módon kívánjuk használni (olvasás, írás)

File kezelés C-ben (6)

- Szöveges file megnyitható text és bináris módban is
- Bináris file csak bináris módon nyitható meg

(Különbség az adatok tárolásának módjában van!!)

File kezelés C-ben (7)

Használat módjának megadása:

”r” – létező file, olvasásra (ha nem létezik, hiba lép fel)

”w” – nem létező file létrehozására, és írására (ha létezik törli azt, és újra létrehozza!!)

”a” – létező file hozzáfűzésére (ha nem létezik, létrehozza)

”r+” – létező file olvasására, írására

”w+” – nem létező file létrehozására, írására, olvasására

”a+” – létező file hozzáfűzésére, írására, olvasására

File kezelés C-ben (8)

Filekezelő függvények:

- fopen() – megnyitás
- fclose() – lezárás
- fgetc(), fgets(), fscanf(), fread() – olvasás
- fputc(), fputs(), fprintf(), fwrite() – írás
- fseek() – pozicionálás a file-on belül (pointer állítása)
- ftell() – pointer állásának lekérdezése
- rewind() – file elejére való pozicionálás
- feof() – file végén van-e a file pointer

File kezelés C-ben (9)

Filekezelő függvények:

- ferror() – hibaállapot lekérdezése
- fflush() – memóriában levő puffer kiürítése írás/olvasás esetén
- setbuf() – memória puffer megadása (ha nem az automatikust akarjuk használni)
- freopen() – a file átirányítása

File kezelés C-ben (10)

Mivel az írás, olvasás pufferen keresztül történik, lezárás előtt mindenképp javasolt az fflush()-sel történő puffer ürítése.

Írás, olvasása esetén a file pointer mindig automatikusan továbblép eggyel, a következő adatra.

File kezelés C-ben (11)

Példa megnyitásra (1):

```
…
FILE * fp;
fp = fopen(”adat.txt”, ”rt”);
if (fp == NULL)
{
        printf(”A file-t nem sikerült megnyitni!\n”);
        return 1;
}
```

File kezelés C-ben (12)

Példa megnyitásra (2):

```
…
FILE * fp;
if ((fp = fopen(”adat.txt”, ”rt”))== NULL)
{
        printf(”A file-t nem sikerült megnyitni!\n”);
        return 1;
}
…
```

File kezelés C-ben (13)

Példa megnyitásra Windows alatt(3):

```
…
FILE * fp;
if ((fp = fopen(”c:\\adatok\\adat.txt”, ”rt”))== NULL)
{
        printf(”A file-t nem sikerült megnyitni!\n”);
        return 1;
}
…
```

```
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
        int i;
        FILE * fp;
        if ((fp = fopen (”c:\\adatok\adat.txt”, ”wt”)) == NULL)
        {       printf(”Hiba a megnyitaskor!\n”);
                return 1;
        }
        for (i = 1; i <= 10; i++)
                fprintf(fp, ”%3d”, i);
        fflush(fp);
        fclose(fp);

        // következő slide-on a visszaolvasás
```

```
		if ((fp = fopen (”c:\\adatok\adat.txt”, ”rt”)) == NULL)
		{		printf(”Hiba a megnyitaskor!\n”);
				return 1;
		}
		while(!feof(fp))
		{
				fscanf(fp, ”%d”, &i);
				printf(”%d ”, i);
		}
		fclose(fp);
		return 0;
}
```

Általános INFORMATIKAI Tanszék

File kezelés C-ben (14)

Az fwrite() használata:

```
fwrite(------, ------, ------, ------);
```

Az 1. paraméter: a memória terület címe, ahonnan adatokat akarok kiírni.

A 2. paraméter: 1 adatelem mérete byte-okban (sizeof-fal)

A 3. paraméter: az adatelemek darabszáma

A 4. paraméter: a filepointer

```
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
        int t[10], i, db;
        FILE * fp;
        for (i = 0; i < 10; i++)
                t[i] = i+1;
        if ((fp = fopen (”adat.bin”, ”wb+”)) == NULL)
        {       printf(”Hiba a megnyitaskor!\n”);
                return 1;
        }
        fwrite(t, sizeof (int), 10, fp);
        fflush(fp);

        // következő slide-on a visszaolvasás
```

Általános INFORMATIKAI Tanszék

```
    rewind(fp);
    db = fread(&i, sizeof(int), 1, fp);
    printf(”%d ”, i);
    fclose(fp);
    return 0;
}
```

Keresések

Általános feltétel:

N rekordból álló halmazból a rekordok egyikének lokalizálása.

További feltétel:

Minden rekord tartalmazzon egy kulcsmezőt.

Feladat:

Megtalálni azt a rekordot, amelynek kulcsa megegyezik a keresett kulccsal.

Két eset:

A keresés sikeres vagy sikertelen lehet.

Szekvenciális keresések

1. Adott az R_1, R_2, …, R_n rekordok halmaza, ahol K_1, K_2, …, K_n jelöli a megfelelő kulcsokat.

   Feltétel: n >= 1

   Megoldás:

   az első rekord kulcsának összehasonlítása a keresett kulccsal. Ha megegyezik, a keresésnek sikeresen vége. Ha nem egyezik meg, vizsgálat a file végére. Ha vége, a keresésnek sikertelenül van vége. Ha nincs vége, akkor a következő rekord kulcsának a vizsgálata...

Szekvenciális keresések (2)

2. Gyors szekvenciális keresés:

Elv: az előző algoritmusban 2 vizsgálat történik:

- kulcs egyezés van-e?
- file vége van-e?

Ha az egyik feltétel elvethető, akkor a keresés felgyorsul.

Megoldás: egy fiktiv (ál)rekord a file végére R_{n+1}-ikként, amelyre nézve $K_{n+1} = K$. Így a file vége vizsgálat kimaradhat.

Szekvenciális keresések (3)

3. Javított gyors szekvenciális keresés:

Elv: Megpróbálni egy cikluslépésen belül két rekordot megvizsgálni.

Megoldás: továbbra is a fiktiv rekordot felvenni R_{n+1} rekordként, majd a ciklust i = -1-ről indítva, és a ciklusváltozót kétszer növelve (Inc(i); Inc(i);) összehason-lítani az R_i, R_{i+1} rekordok K_i, K_{i+1} kulcsait K-val.

Eredmény: az 1. algoritmushoz képest kb. 30 %-kal gyorsabb!

Szekvenciális keresések (4)

4. Rendezett táblában szekvenciálisan keresni:

Elv: Az előző algoritmusok csak akkor tudták eldönteni a sikertelen keresést, haa file végére értek. Egy rendezett táblában ha a K_j kulcs nagyobb K-nál, akkor belátható, hogy a keresett rekord nincs a táblában.

Eredmény: sikertelen keresés esetén átlagosan kétszer gyorsabb!

Szekvenciális keresések (5)

5. A tábla rendezettségi szempontját megváltoztatni:

Elv: feltehető, hogy a kulcsokra nem egyforma gyakorisággal hivatkozunk. Ekkor:

K_i kulcs p_i valószínűséggel fordul elő, ahol:

$$p_1 + p_2 + \dots + p_n = 1 \quad (100\ \%)$$

Megoldás:

Azokat a rekordokat előre tenni a táblában, melyeknek a gyakorisága nagyobb.

Általános INFORMATIKAI Tanszék

Rendezések

Javasolt irodalom:

D. E. Knuth: A számítógép programozás művészete III. kötet

Rendezések

Alapelvek:

- Beszúró rendezés: egyesével tekinti a rendezendő számokat, és mindegyiket beszúrja a már rendezett elemek közé. (Kártyalapok rendezése)
- Cserélő rendezés: ha két elem nem a megfelelő sorrendben követi egymást, akkor felcserélésre kerülnek. Ez az eljárás ismétlődik mindaddig, míg további cserére már nincs szükség.
- Kiválasztó rendezés: először a legkisebb (vagy legnagyobb) elemet határozzuk meg, és a többitől valahogy elkülönítjük. Majd a következő legkisebbet választja ki, stb…
- Leszámoló rendezés: minden elemet összehasonlítunk minden elemmel. Az adott elem végső helyét a nála kisebb elemek száma határozza meg.

Algoritmusok

- Leszámoló: -
- Beszúró:
  - közvetlen beszúrás
  - bináris beszúrás
  - Shell rendezés
  - lista beszúró rendezése
  - címszámító rendezés
- Cserélő:
  - buborék rendezés
  - Batcher párhuzamos módszere
  - gyorsrendezés (Quick sort)
  - számjegyes cserélés
  - aszimptotikus módszerek
- Kiválasztó:
  - közvetlen kiválasztás finomítása
  - elágazva kiválasztó rendezés
  - kupacrendezés
  - „legnagyobb be, első ki”
  - összefésülő rendezés
  - szétosztó rendezés

Általános cél

Adott $R_1, R_2, \ldots, R_n$ rekordokat $K_1, K_2, \ldots, K_n$ kulcsaik nem csökkenő sorrendjébe kell rendezni lényegében egy olyan p(1), p(2), …, p(n) permutáció megkeresésével, amelyre

$$K_{p(1)} <= K_{p(2)} <= \ldots <= K_{p(n)}$$

fennáll.

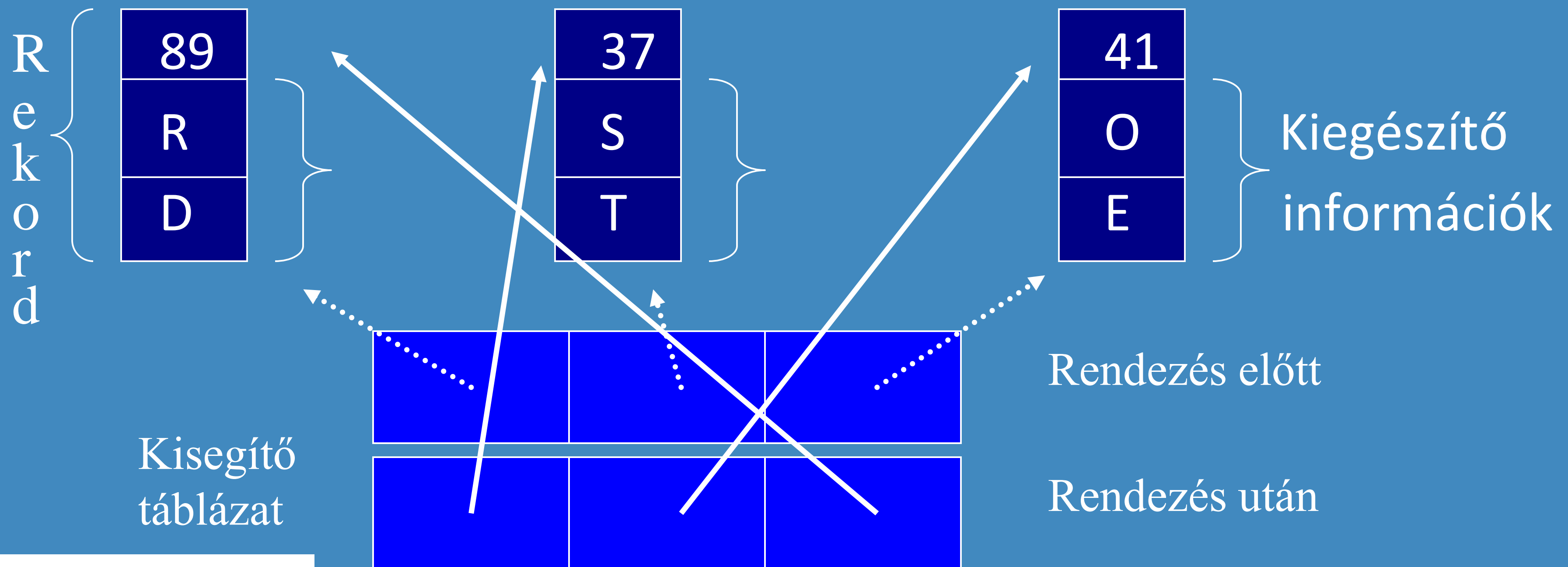
Ideiglenes feltétel:

Tekintsünk olyan halmazt, amelynek rendezése a memóriában megvalósítható.

Általános INFORMATIKAI Tanszék

Célszerű adatszerkezetek

1. Ha a rekordok mindegyike több szót foglal el a tárban, akkor célszerű a rekordokra mutató láncolt címek új táblázatának létrehozása és ezeknek a kezelése a rekordok mozgatása helyett. Ez a **CÍMTÁBLÁZATOK rendezése**.



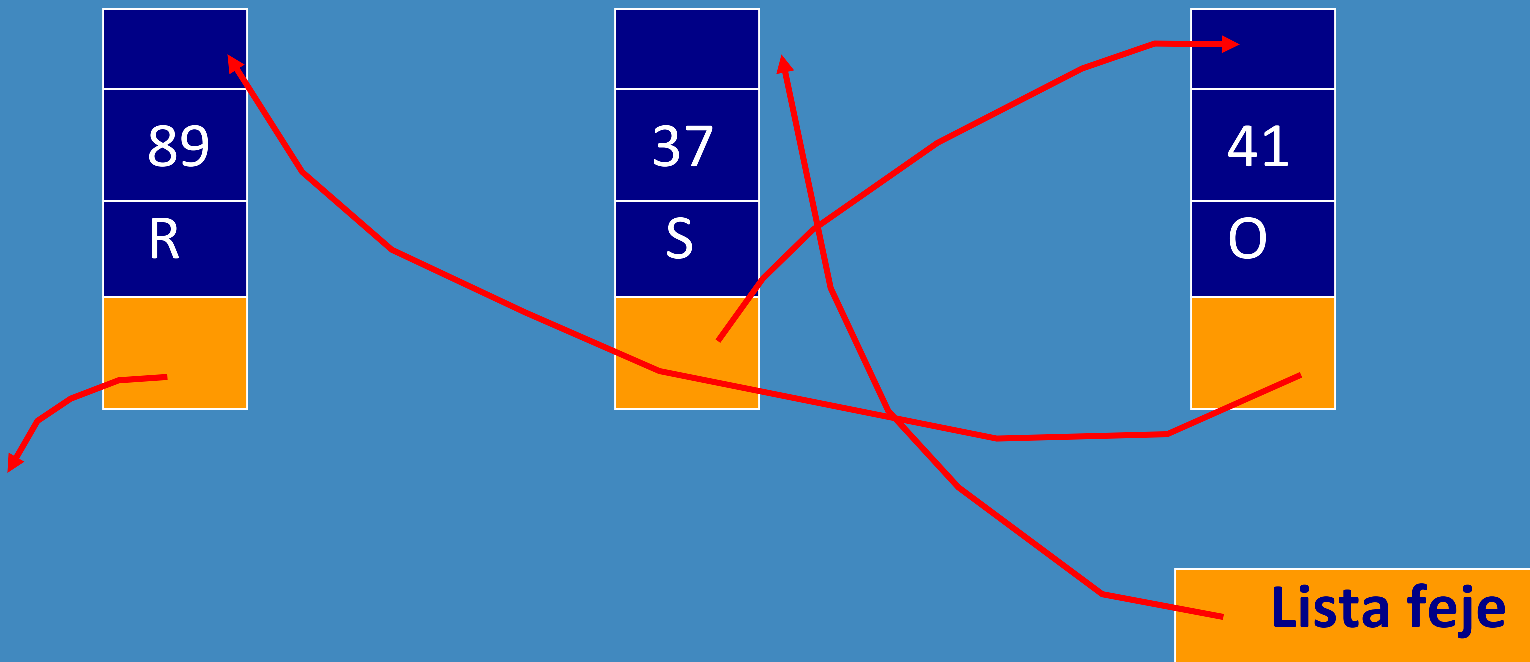
Általános INFORMATIKAI Tanszék

Célszerű adatszerkezetek (2)

2. Ha a kulcs rövid, de a rekord kiegészítő információja hosszú, akkor a nagyobb sebesség elérése érdekében a kulcs a láncoló címmel együtt tárolható. Ez a **KULCSOK rendezése**.

Célszerű adatszerkezetek (3)

3. Ha a láncoláshoz az egyes rekordokhoz csatolt segédmezőt alkalmaznak úgy, hogy a rekordok együtt egy lineáris listát alkotnak, amelyben minden láncoló cím a következő rekordra mutat, akkor az a **LISTA rendezése**.



Leszámoló rendezés

Elve: a rendezett listában a j-ik kulcs pontosan j-1 kulcsnál lesz nagyobb. (Ezért ha egy kulcsról tudjuk, hogy 27 másiknál nagyobb, akkor a neki megfelelő rekord sorszám a rendezés után a 28 lesz.)

A kulcsokat minden lehetséges párosításban össze kell hasonlítani, és megszámolni, hogy az éppen vizsgált kulcsnál hány kisebb van. Variációk:

1. Hasonlítsuk össze K_j-t K_i-vel $1 \leq j \leq N$ esetén, ha $1 \leq i \leq N$

   de felesleges:
   - minden számot önmagával is összehasonlítani
   - K_a-t K_b-vel, majd utána K_b-t K_a-val összehasonlítani

2. Hasonlítsuk össze K_j-t K_i-vel $1 \leq j \leq i$ esetén, ha $1 \leq i \leq N$

Fontos! Az algoritmus nem jár együtt a rekordok mozgatásával, inkább a címtáblázat-rendezéshez hasonlít. Futási ideje: **13 N + 6 A + 5 B - 4**, ahol:

N = a rekordok száma, $A = \binom{N}{2}$, B = az olyan indexpárok száma, amelyekre **$j < i$, és $K_j > K_i$**

Beszúró rendezés (1)

(„Bridzsező módszer”)

Feltétel: Az R_j rekord vizsgálata előtt a megelőző $R_1, \ldots, R_{j-1}$ rekordok már rendezettek. Ekkor az R_j rekordot beszúrjuk az előzőleg már rendezett rekordok közé. Változatok:

Közvetlen beszúrás, v. szitáló technika:

Tegyük fel, hogy $1 < j \leq N$, és hogy $R_1, \ldots, R_j$ rekordok rendezettek úgy, hogy $K_1 \leq K_2 \leq \ldots \leq K_{j-1}$. Ekkor a K_j kulcsot összehasonlítjuk a K_{j-1}, majd a K_{j-2} kulcsokkal mindaddig, míg R_j be nem szúrható R_i és R_{i+1} közé.

Bináris beszúrás:

Nem sorosan, hanem binárisan kell megkeresni az R_j rekord helyét a már rendezett rekordok között. **(N > 128 esetén már (!) nem javasolt)**

Előnye: kevesebb összehasonlítás

Hátránya: megtalált hely esetén továbbra is nagy mennyiségű rekordot kell megmozgatni a beszúráshoz.

Beszúró rendezés (2)

Kétirányú beszúrás:

Bináris kereséssel a hely megkeresése, majd beszúrásnál abba az irányba mozgatjuk a már rendezett rekordokat, amerre kevesebbet kell mozgatni. (Jellemzője a nagyobb memória igény!).

SHELL (v. *fogyó növekményes*) rendezés:

Eddig a rekordok rendezéskor rövid lépésekkel kerültek helyükre. Most: rövid lépések helyett hosszú ugrások legyenek. Megalkotója: Donald L. Shell. (1959. július).

Elve: A rekordokat először (pl.) kettesével csoportokba osztjuk. Ezeket a rekord csoportokat rendezzük (a két elem vagy jó sorrendben követi egymást, vagy felcseréljük őket). Ezután négyesével képezzük a csoportokat és rendezzük, majd újabb csoportokat képzünk, rendezzük, …, míg a végén már csak egy csoport lesz, a teljes rekordhalmaz. Addigra az már rendezett részhalmazokból áll.

Beszúró rendezés (3)

Shell rendezés folyt: minden közbenső fázisra igaz, hogy ***vagy viszonylag kicsi a csoport, vagy viszonylag jól rendezett***, ezért a rekordok gyorsan mozognak végső helyük felé.

Hatékonysága nagy mértékben függ az alkalmazott növekmé-nyek sorozatától. Erre táblázatban találhatók meg a javasolt növekménysorozatok. Ha ez nincs kéznél, akkor:

$\mathbf{n_1=1, \ldots, n_{s+1} = 3 \times n_s + 1}$ és akkor legyen vége, ha:

$$\mathbf{n_{s+2} > N}$$

Általános INFORMATIKAI Tanszék

Lista beszúró rendezése

(A közvetlen beszúrás javítása.)

Az alapalgoritmus 2 alapvető műveletet alkalmaz:

- rendezett file átnézése adott kulcsnál kisebb vagy egyenlő kulcs megtalálása céljából.
- új rekord beszúrása a rendezett file meghatározott helyére

Kérdés: melyik az az adatszerkezet, amely erre a legalkalmasabb?

A file egy lineáris lista, amelyet ez az algoritmus szekvenciálisan kezel, ezért minden beszúrási művelethez átlagosan a rekordok felét kell megmozgatni.

Beszúráshoz az ideális adatszerkezet a láncolt lista, mivel csak néhány címet kell átírni a beszúráshoz. Ehhez elegendő az ***egyirányú láncolt lineáris lista.***

Várható futási ideje: **7B+14N-3A-6** időegység, ahol N = a rekordok száma

A = a jobbról-balra maximumok száma, B = az eredeti permutációban levő inverziók száma (kb. 1/4 N^2)

Címszámító rendezés

Elve: könyvek vannak a padlón. Ezeket kell „abc” sorrendben egy polcra helyezni.

Meg kell becsülni a könyv végső helyét. Ezzel lecsökkenthető az elvégzendő összehasonlítások száma, valamint a rekord beszúrásához szükséges mozgatások száma.

Hátránya: általában N-nel arányos további tárterületet igényel azért, hogy a szükséges mozgatások száma lecsökkenjen.

Cserélő rendezések

<u>Buborék rendezés</u>:

<u>Elve</u>: hasonlítsuk össze K_1, K_2 kulcsokat. Helytelen sorrend esetén cseréljük fel R_1, R_2 rekordokat, aztán ugyanezt hajtsuk végre R_2, R_3 , majd R_3, R_4 , R_4, R_5 , … rekordokkal.

Az eljárás az R_N, $R_{N-1,}$ R_{N-2}, ... rekordokat juttatja a megfelelő helyre.

Függőleges ábrázolás esetén látható, hogy a nagy számok fognak először „felbuborékolni” a helyükre.

<u>Más neve</u>: **cserélve kiválasztás** v. **terjesztés**.

<u>Futási ideje</u>: 8A + 7B + 8C +1, ahol

A = a menetek száma, B = a cserék száma,

C = az összehasonlítások száma

<u>Max értéke</u>: $7.5\ N^2 + 0.5\ N + 1$ (N^2 -tel arányos idő…)

A buborék rendezés algoritmusa

(Nem azért mert jó, hanem mert elterjedt…)

Lépések:

1. Legyen korlát = N (korlát a legnagyobb index, amelyhez tartozó rekordról nem tudjuk, végső helyén van-e.
2. t = 0. Végezzük el a 3. lépést a j = 1, 2, …, korlát-1 értékekre, aztán menjünk a 4. lépésre. (Ha korlát = 1, akkor közvetlenül a 4. lépésre)
3. R_j és R_{j+1} összehasonlítása. Ha $K_j > K_{j+1}$, akkor cseréljük fel R_j -t R_{j+1} -gyel, és legyen t = j. (Ez jelzi, hogy volt csere…)
4. Történt-e csere? Ha t = 0, akkor nem, és az algoritmusnak így vége. Egyébként korlát = t, és visszatérni a 2. lépésre.

(A közvetlen beszúrással összehasonlítva, kétszer lassúbb!!)

Batcher párhuzamos módszere

(Leginkább a Shell-rendezéshez hasonlítható.)

Elve: összehasonlításra ne szomszédos párokat válasszunk ki, hanem általánosan felírva:

K_{i+1} -et K_{i+d+1} -gyel,

ahol d meghatározására van(!) algoritmus.

Kiválasztó rendezés

Elve:

1. Keressük meg a legkisebb kulcsot. A neki megfelelő rekordot tegyük a kimenő terültere, majd helyettesítsük kulcsát végtelennel.
2. Ismételjük meg az 1. lépést. Ekkor a 2. legkisebb kulcsot találjuk meg, mivel az elsőt végtelennel helyettesítettük.
3. Az 1. lépést ismételni mindaddig, míg az összes N rekord kiválasztásra nem került.

Feltétel: a rendezés megkezdése előtt minden elem jelen legyen.

Hátránya:

- a végső sorozatot egyesével generálja
- minden egyes elem kiválasztásához N-1 összehasonlítást végez el.
- külön output területet igényel.

Nem vesszük (sajnos...)

- gyorsrendezés (Quicksort)
- kupacrendezés
- „legnagyobb be, első ki”
- összefésülés
- szétosztó rendezés

(De aki gondolja, utánanézhet)

Házi feladat:

Tessék megpróbálni megkeresni a cserélve történő kiválasztás helyét a tanult rendezési alapelvek között! (vajon cserélő, vagy kiválasztó-e?)