

Processzorok utasításkészlete.

2 irányzat $\rightarrow$ utasítások számát növeli, pl. Intel (MMX - multimedia)

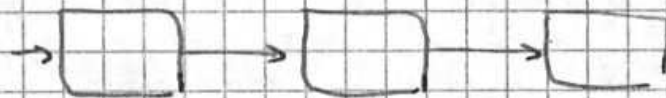
- könnyedesebb a programozása

$\rightarrow$ redukált utasításkészletű proc (RISC - proc)

- azonos órajelciklusok alatt feldolgozódnak

statisztika a használt utasításokról: 40-50 nagyon
sűrű, ezt valószínűleg meg u. a. órajel alatt
alább

Pl: 3 ciklus: pipe-line



Egyszerre 3-at tud végezni a kiél. egyidejűleg.

Neumann-elv: nem tudja, hogy adat vagy utasítás algoritmus, ha, akkor
végrehajtja. $\Rightarrow$ erre emlemből utasítások, prog. nyelvvel.

Magas szintű programozási nyelvek (pl. C)

- operátorok (bitok - bájtok kezelése)
- összetett adattípusok $\Rightarrow$ • típusok definíciója felh. által.
- utasítások struktúrája (egymásba ágyazás, ...)
- rekurzió lehetővé tétele/támogatása

$$n! = n \cdot (n-1)! , \text{ ahol } n=1 \text{ a triviális eset}$$

- hordozható/portális programok (proc. arch. váltás, OS-váltás)

Fordító pgm.

ferrésítés $\rightarrow$ futtatható kód (exe)

- bináris kódgenerálás, proc. szintű utasításokkal, kiegészítő perifériákkal
közli előfordul $\Rightarrow$ az adott op. OS-en fut.

assembly
A, B, C nyelv

↑
Kernighan, Ritchie, telefonközpontok programozására
70-es évek közepe

UNIX 90 %-a ellen kévelt el C-ben

↓
már el lehetett készíteni a többi magas szinten is.

ANSI szabványosította

- objektumorientáltág + C → C++

C az assembly-ből

- majdnem közvetlen mem. hozzáférés
- bitmanipulációs műv.
- alacsony szintű ^{byte} byteorientált I/O műv.
- OS-szel jól def. kapcsol.

Az első C pgm.

```
#include <stdio.h>
```

```
void main(void)
```

```
{  
    printf("Hello world!\n");  
}
```

← kódolás (= programozás)

hello.c

↓
gc → hello.obj
↓
hello.exe

Programozás

• specifikáció (mit akarunk megvalósítani) "írásban" rögzítjük

algoritmus
választás

elemi utasítások sor-a, amely néges számú lépésben
eredményre vezet.

adatszerkezet
megvalósítása

nyelv választás

feladatára
struktogram

tesztelés

tesztelés

documentálás

jó specifikáció alapján megírható

- programozási dokumentáció (algoritmusok, tesztelés, ...)
- felhasználói dokumentáció (minden kiírt felhasználó)

n! kiszámítása

• beadás, számok, kiírás

• $n! = n(n-1)!$

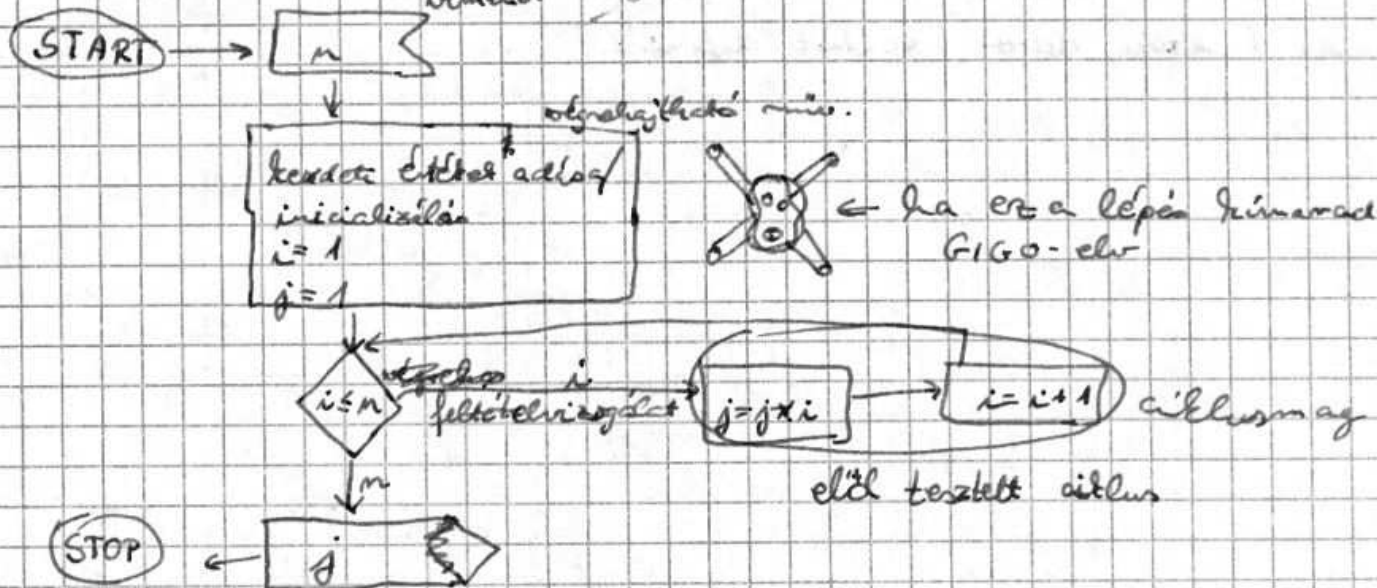
$$n! = 1 \cdot 2 \cdot 3 \cdot \dots \cdot (n-1) \cdot n \quad \prod_{i=1}^n i$$

változó: számoló i ← iterátor

n (számoló végeztető)

szorzószorzat tároló j ← akkumulátor

Semantika (a bontásban foglalkozunk)



C szintaktikai szabályok szerint kódolunk

```
#include <stdio.h>
```

```
void main(void)
```

```
{
```

```
    int n, i, j;
```

```
    printf("n=?"); scanf("%d", &n);
```

```
    i = 1; j = 1;
```

```
    while (i <= n)
```

```
    { j = j * i; i = i + 1;
```

```
    }
```

```
    printf("%d! értéke = %d \n", n, j);
```

```
}
```

— előre definiált adattípusok azonosítása

— for tárolási egység (adatt, kód) azonosítása

□ operator (műveletvégzés)

○ spec. szintaktikus, a művelet végzését irányítja a számítógép.

— kulcs, adott utasítást definiál.

- azonosítók

- változó
- függvényel (szubrutin)

- kulcsszavak

- spec. karakterek

- operátorok

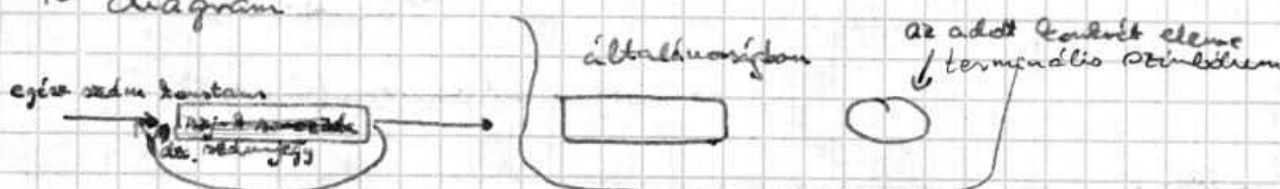
- konstansok

SZINTAKTIKA: hogy néz ki

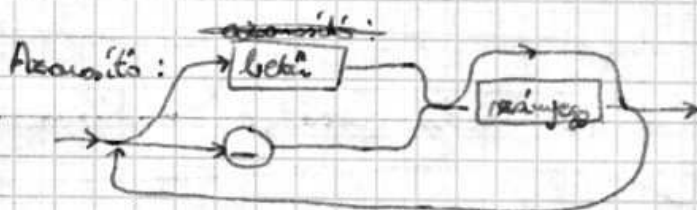
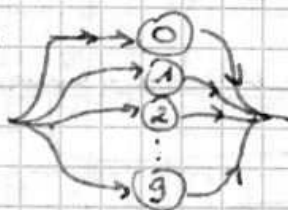
SZEMANTIKA: mit csinál

Szimulátika

~ diagram



Decimális számjegy:



Néhány jellegzetesség a C-ben:

- Kötelező írásmód, teljességes whitespace karakter (\n, \t, \r)
- kis- és nagybetű, - a C-ben különböztet

Konvenció: saját azonosítót nem kezdünk _-al.

kisbetűs azonosító a define szóra folytatva

- /* kommentár */ whitespace helyen

a=1 (azonosító hossza legyen ≤ 8, nagybetű nélkül)

Öndokumentált program: a dokumentáció már a program-ból áll

- # preprocessor utasítások kezdete

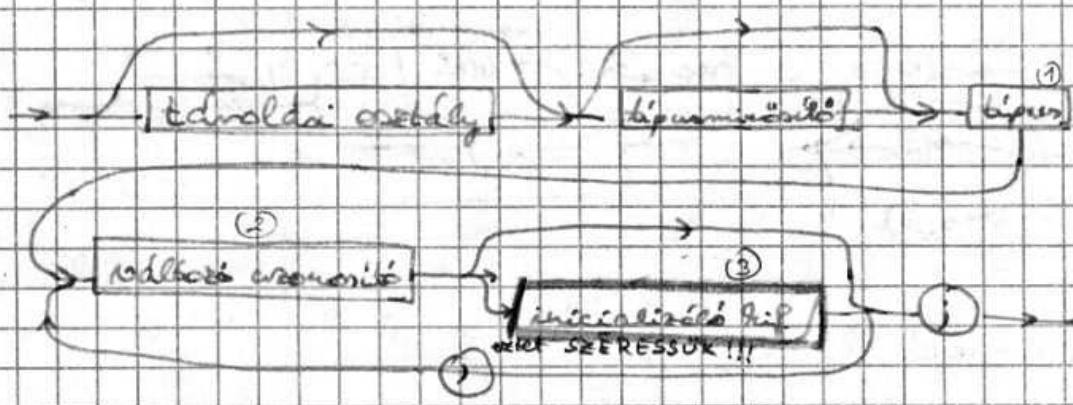


Neumann: a C nem tesz különbséget fvr.-t és változót között
 ↓
 Lényegi egységet.

• változ. foglalkozás → változás
 → konstans

• kódgeneráló tka. egg. (C fvr.-t)

Változó definíciók



① ② ③
 int m, i=1, j=1;

egész helyett valós szám: double j=1.0;

Adattípusok C-ben:

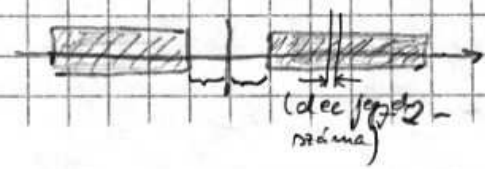
előre def:

felh. által def

egész
 nem teljes egészeken alapul
 char (ASCII) short int long
 → előre definiálva:
 $0 \dots 2^n - 1$
 → előre definiálva:
 $-2^{n-1} \dots 2^{n-1} - 1$

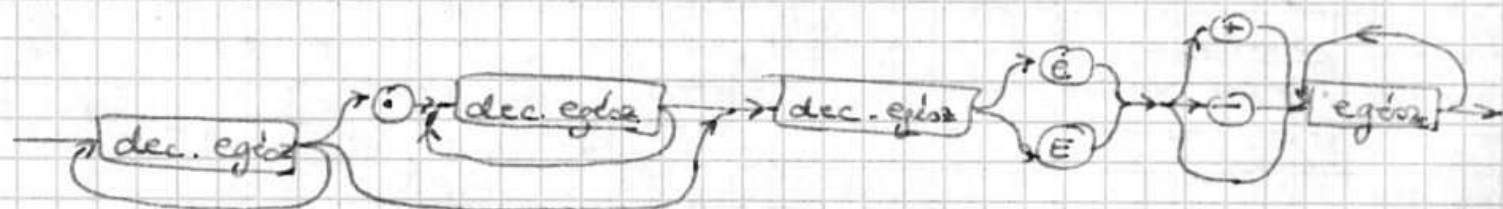
valós
 float IEEE [a]triple i] 4 bjt $10^{\pm 38}$
 double IEEE 8 bjt $10^{\pm 308}$
 long double

Hozzájárulás
 • nem tudunk állományon his
 számot megadni
 • nem tudunk állományon his
 különbséget csinálni 2 más
 között



sizeof (típus) $\text{sizeof}(\text{char}) = 1 \text{ (byte)}$ $1 = \text{sizeof}(\text{char}) \leq \text{sizeof}(\text{short}) \leq \text{sizeof}(\text{int}) \leq \text{sizeof}(\text{long})$ $\text{sizeof}(\text{float}) \leq \text{sizeof}(\text{double}) \leq \text{sizeof}(\text{long double})$
32 bit 64 bit 80 bitBelső formátum:negatív számoknál kettes komplementum ($\Rightarrow$ bináris helyett összeadás)Külső formátum:

char: az ASCII kódjával (belső formátummal megegyezően)

} Konverzió: az az előre definiált típusokra működik.

pl: printf %d << >>

scanf %d

műveletek

A C nyelv utasításai

- ; írás utasítás
- kifejezés utasítás (több operátor és operandus)
 - ↳ kif. kiértékelése

konstans : $a = 1 + 3 \times 5$ már a fordító elvégzi; kiértékelés 1×16

- Vezérlési utasítás: if if...else

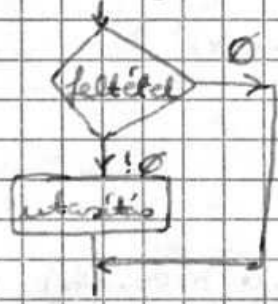
switch

(ugró utasítás)
csak utána
NEM SZERESSÜK

- ciklus utasítások
- összetett utasítás { }

↑ az így kezdés, ellenőrzés és az operátor

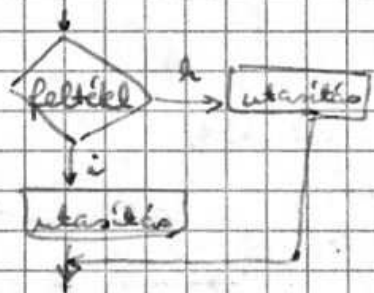
Égőcsú elágazás



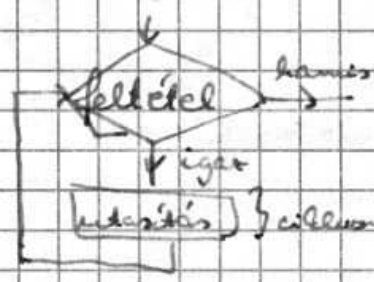
if (a) u;

Logikai érték

0 hamis
1 (nem nulla) igaz

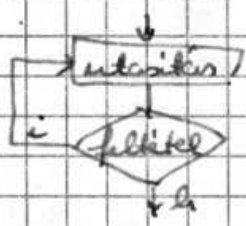


if (a) u; else v;



while (feltétel) utasítás;

! while (1)
! ∞ ciklus
! goto



do utasítás while (feltétel)

∞ ciklus

kezdeti érték
léptek
végfeltételhez kapcsolódás

for (kezdeti érték; kapcsoló; léptető) utasítás;
j = 1;
for (i = 1; i ≤ n; i = i + 1) j = j * i;

Kifejezés utasításokA C nyelv operátorai

Egy operátor jellemzői:

- hány operandusa van
 - milyen típusú op?
 - az eredmény típusa (explicit: hamis=0, igaz=1)
 - az operandusok írástételeinek sorrendje, ASSOCIATIVITA'
 - FŐ HATA'S
 - MELLÉK HATA'S
- } SORRENDJÜK
- PRECEDENCIA (mi előbb, mi később)
 - TÍPUSKONVERZIÓ (természetes és automatikus)

(R) Az értékelés művelete:

operátor: =
↳ eredmény
↳ érték

$a = b$
 kifejezések lehetnek
 balérték = jobbérték
 lvalue = rvalue
 ↓
 bármilyen érték adható (konstans, változó, stb.)
 bármilyen érték van (konstans, változó, stb.)
 (bármilyen érték adható (konstans, változó, stb.)
 (bármilyen érték van (konstans, változó, stb.)
 (bármilyen érték adható (konstans, változó, stb.)
 (bármilyen érték van (konstans, változó, stb.)

op típusa: alacsony, de!

- lvalue/rvalue KOMPATIBILIS TÍPUSÚ LEGYEN

ha lvalue „bővebb”: rvalue kikeresése

ha lvalue „szűkebb”: rvalue konvertálása!”, ha belefér

 $\mathbb{Z} \rightarrow \mathbb{B}$

FŐ HATA'S:

Eredmény: rvalue értéke

rvalue idézi.

lvalue idézi.

Mellékhatás: lvalue-ba bemásolódik rvalue.

Következő: TÖBBES ÉRTÉKADÁS

$$a=b=c=d=1$$

Az előző példában
1 int m, i, j

$$\left. \begin{array}{l} i=j=1; \\ \end{array} \right\}$$

egészlet:

unsigned = signed !! (idem) bitmanta maddas

signed = unsigned !

Valdovak:

double = float ✓

float = double, ha kell " (vismalépjünk a pontosságban)

Vegetables

$\text{int} = \text{val} \rightarrow \text{int}$, ha befejez, a történet csomópontja

valds = int ✓

Művelésvégzet ^{és}	egészeli	bővel
	valósít	double- no

A C legfontosabb operátorai

1.) ARITMETIKAI MŰV

+ - * a student's dilemma

1 - egész osztás : maradék és osztás végeredménye

- valto omnia: ha legalitás az egyik gerendés valós
rendes köztartás.

2. eigene von Erstellern: modulo (Pascal: MOD)

maximal order maximal : $a \sim b \rightarrow \{a, b\}$

(7e)

int a=2, b=3;

double x, y=2.0;

x = a / b /* 0.666... */ ! 

x = y / b /* 0.6666... */

típuskonverziós operátor:

x = (double)(a) / (double)(b)

EXPLICIT TÍPUSKONVERZIÓ:

(új-típus) (kifejezés)

↓
prefix operátor

PRECEDENCIA : * / %
+ -

Op. rend : B → I

x = $\frac{a+b}{c \cdot d}$ kódolva : x = (a+b) / (c * d) egészben () !!!

~~1.~~ KIFEJEZÉSVÉRTÉKELŐ OP : a precedenciája a legmagasabb
() - operátor, 1 operandus

2.) Logikai operátorok

! NEM NEM (IGAZ) → HAMIS

NEM (HAMIS) → IGAZ

|| (ÉS) VAGY

Ígazságtábla
De Morgan

A || B B → I

ha triv. az eredmény, akkor B nem értékelendő

RÖVIDZÁR

ha az fontos min és leírás
bej, min pl minis volt ~~mellette~~
melléklet

B - ben ne legyen melléklet

&& ÉS A && B B → I

RÖVIDZÁR !



3) Relációs műveletek

$<$
 $>$
 $<=$
 $>=$

Relációs

Operandus: olyan, ahol a sorrendesség logika
 értékelhető
 → számok
 → memóriacímek
 → értékek
 → adatok

4) Egyenlőségvizsgálatok

$=$ if (a == b) else _
 $!=$ if (a != b) 

Konvenció: if-ben mindig értékelhető

5) Egyszerű operátorok

sizeof ()
 - típus
 - adat

sizeof (char) ; sizeof ("hello")
 ↓
 6 h e l l o \0
 EOS

(új típus) (kif) TYPE CAST

Bitenkénti műveletek

~ megfordítás

$\&$ és
 $\wedge$ XOR (kizáró vagy) akkor igaz, ha !=
 $|$ vagy
 pl. kurzor.

$\ll$ bitmintha eltolás balra

$\gg$ jobbra

Pé: $\sim 0000 \rightarrow 1111$

$\begin{array}{r} 1001 \\ 1101 \\ \hline 0100 \end{array}$

$\gg$ $\gg$

$\ll$ $\ll$

mit $\ll$ menesztve

Pé: $0001 \ll 1$ dec 1
 0010 dec 2

Operátor: $(felt) ? igaz\ kif : hamis\ kif$

Pé: $b = x ? a : c ;$

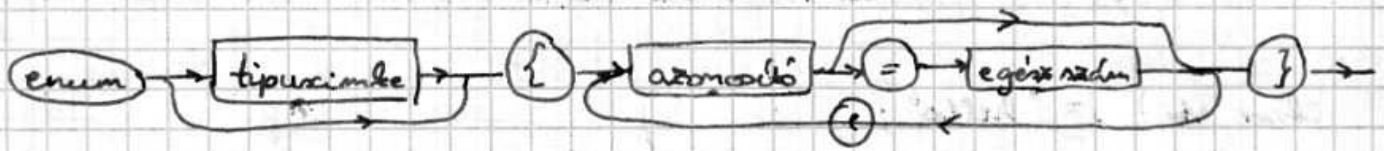
if (x) $b = a ;$
else $b = c ;$

↗ a várt mellékletés nem létezik benne
(pl. értékelés)

Eddig: alaptípusok - ún. egyszerű típusok

Most: felh. által. def. típusok - saját egyszerű típus: enum (sorrendezett/felbor-
e egy eleméhez konkrét belső értéket tárolnak.

Def. ált. alakja:



Pé. A hét napjai:

enum napok { hétf, kedd, szer, csu, pé, szo, vas } ma ;

enum napok tegnap, holnap = kedd ;

int-tel kompatibilis : egészre értelmezett műveletek végrehajthatóak vele

pl: ++/--

for (ma = hétf; ma < szo; ma++)

{ switch (ma)

{ case hétf: C eloadas();

case kedd: Munka ();

case szo: C gyar (); break;

}

}



Vigyázzunk, ha

az értékelés után

előfordulhat.

Állapotgépes modell

↳ állapotábra, állapotgráf (lényeg)

- az enum típus a program belső állapotának nyilvánvalóan jól használható.

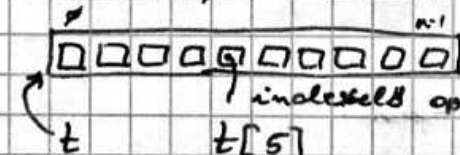
- Öröklött típusok definíciója:

- egy egységesen töltő adatot tárolhatunk $\rightarrow$ tömb / array
- egy egységesen különböző adatot tárolás $\rightarrow$ struktúra: struct (RECORD)

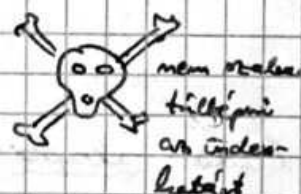
int i;

skalár
i

int t[10];



számszerű típus



double x[5][5]

tömbök tömbje: 2 dimenziós tömb = mátrix

x[j][i] adott sor adott eleme

char str[] = "hello";

méret = sizeof(str) / sizeof(char)

int t[] = {0, 1, 3, 7, 5, 2};

méret = sizeof(t) / sizeof(int)

logikai méret = $\frac{\text{sizeof}(tömbtípus)}{\text{sizeof}(alaptípus)}$

Gond: Fix méret

Pl: m x n -es nx? Dinamikus helyfoglalás

↳ memória az OS-tól.

↳ megjegyezni a lefoglalt mem. kezdőcímét

új adattípus: pointer / mutató $\rightarrow$ saját típus

↳ pointer változóin keresztül hozzáférni a mem.-hoz.

↳ lefoglalt mem.-t visszaadni (felszabadítani)

Pointer típusok definíciója C-ben

$\text{int } *ip;$
 ↑
 címet
 egy egész érték (hely) mutathatja
 típus módosító op, prefix, mert a lista eleme előtt áll



tilos olyan nem mutatót
pointert definiálni.

$\text{int } i, j, *ip, k;$
 ↑
 ilyenkor csak az ip lesz pointer.

Hogy kapunk értéket?

- NULL : minden számmal megkülönböztethető
<stdlib.h>

$\text{int } *ip = \text{NULL};$

ezt mindig ~~is~~ kell végig definiálni

mindig azt jelzi, hogy nem mutat semmi

ha valahová mutat, akkor nem ~~is~~ NULL, hanem valami, amit én hivatkozom meg.

- egy másik változó címét hozzá ~~adom~~ értékel.

$ip = \&i;$

i 00...01 $\leftarrow$ int
 $\vdots$
 ip &i $\leftarrow$ int *

Indirekció operátora : *

↓
a mutatott adat elérése

*ip - a mutatott memóriacímre (olyan, mint a tömböt elérési módja és hivatkozása a [])

- tömb kezdőcímét is megkapjuk

$ip = \&(t[0]); \iff ip = t;$

a tömb neve egyben a kezdőcíme is

Pointer-aritmetika

++ / --

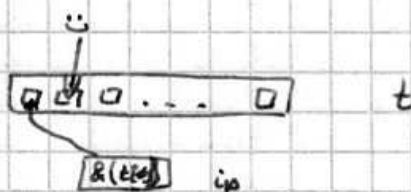
+ egész

- egész

$ip = \&(t[0]);$

$ip++;$

a C tudja, hogy int, és hogy mekkora egy elemméretje, hogy kell mutatni a művelet után (i)



előre ism.

* $ip \equiv t[0]$

* $(ip+1) \equiv t[1]$

* $(ip+n) \equiv t[t]$

eltérő/offset cím: mennyivel améltól kell menni.

Működik és szabad: $ip[0]$
 $ip[1]$

Memória allokáció

#include <stdlib.h>

void* malloc(size_t n)

↓
kérjen egy egész szám → hány bajtot foglaljon a mem-ban

void free (void*) mem-t szabadít fel.

$ip = (int*) (malloc(10 * sizeof(int)));$

if ($ip == NULL$) HIBA! (memóriadoptáció hiba)

$ip[]$ *ip

$ip++$

$ip \dots ip9!$



if ($ip \neq NULL$) { free(ip); ip = NULL; }

Itt is pedig használhatjuk a dinamikus tömböt.

Felh. def. típusok - enum

- összetett típus - tömböt $\&$ [] [indirekció]
- pointer $\ast$ $\ast$ indirekció

NULL & tömb inicializálása (tömb "new") malloc

```
#include <stdio.h>
#include <stdlib.h>
```

```
void main(void)
```

```
{ int m; act = A; } // m-tartja a tömb elemeinek számát, KÖTELEZŐ!
```

```
double *vector = NULL;
```

```
printf("m = "); scanf("%d", &m);
```

```
vector = (double*) malloc(sizeof(double) * m);
```

```
if (vector == NULL) { hibajelzés / hibakezelés }
```

```
else { vector[i] / *(vector + i) } // tömbben vagy vektorban  
használatos a pointer mem-t
```

```
/* felszabadítja a memóriát. */
```

```
if (vector != NULL) { free(vector); vector = NULL; }  
// visszaadja a 0-OS-ot a mem-nek, semmi m.  
// ha a vector == NULL => free elhagyható az if.
```

TÁBLÁZATKEZELÉSI MÓDSZEREK

→ keresést
→ rendezést

TÁBLÁZAT

1D tömb
"egyszerűsített adatbázis"

kulcs (ami szerint rendezni tudunk) $\equiv$ adat, ami a tömbben
általában egész szám, amit
a keresendő adatok adataiból generálunk.
(= adat, ami a tömbben
létezik)

- Egyszerű adatbázis-műveletek:

- beírás
- törlés
- keresés

int t[MAX]; /* int *t, m */

int m=0; (az adatbázisban tárolható értékes elemek száma)

Kötelező, különben



if (m < MAX)

{ t[m] = m. elem; m++; }

← logitailag

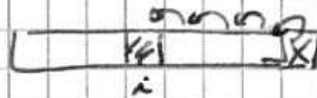
← frontailag

utolsó elem törlése: megnézem, van-e ilyen sorozatszáma, majd logitailag törlöm

if (m) m--;

közelben lévő elem törlése:

if (i < m-1)



{ for (; i < m; i++) t[i] = t[i+1]; }

m--; (az utolsó törlése)

Keresés

a) Lineáris Keresés

int found=0; int m=0;

int x, size;

element e, table[N];

⋮

size = sizeof(table) / sizeof(element); (a table logikai méretét adja)

⋮

m = 1;

e elemet szeretnénk megtalálni

for (x=0; x < m; x++) { if (t[x] == e) found=1; }

found = table[x] == e;

if (found) megvan az elem
jelző flag.

!!!

csak egy elemi típusra működik az összehasonlítás

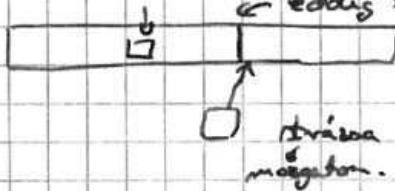
összetett típusoknál, ártomány: strcmp a hasonlító függvény

MŰVELETEK: hasonlítás

max. szám: 2 * N (vegyis 100): elemet elemmel, jelző flag a 0-val.
állítás, állásfaj

állítás: $2 \times N / 2 = N$

b) lineáris keresés szisztematikus



tömb

értéke

nem sikerül állandóan megtalálni a jelet a végére kerül, akkor nem sikerül.

for ($x = 0$; $e \neq \text{table}[x]$; $x++$)
 $i \leftarrow$ x mindig azonos egyenlőség
 üres cella.

• leáll, ha az elejére ér

• vagy ha egyenlőséget talál.

found = ($x \leq n$)

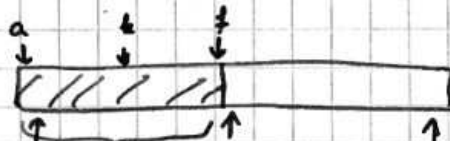
az index-nél kisebb voltam és a tömböt találtam

vagy pedig bent maradtam, és a (tömb) előtti index

- a keresett elem

c) Bináris / logaritmus keresés : átlagos műveleti szám $\log_2 n$.

Alapfeltétel: a tömb rendezett a kulcs szerint.



alho $\uparrow$ itt van középe $\uparrow$ felso (legnagyobb kulcsérték)

ha kulcs < középe

ha kulcs > középe

```
int n;  
int alho, felso, x;  
:  
alho = 0; felso = n;
```

```
while (alho <= felso)
```

```
{ x = (alho + felso) / 2;
```

```
if (e >= table[x]) alho = x + 1;
```

```
if (e <= table[x]) felso = x - 1;
```

```
}
```

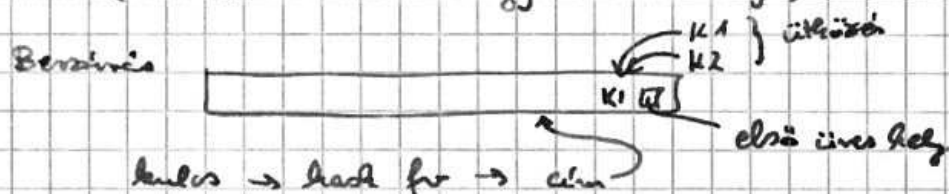
```
found = (alho + felso == 2)
```

d.) Condicion / keresés / keresés / hashing :

Ötlet:

kulcs $\rightarrow$ cím (azonnal eldöntjük, hogy hol kell lennie)
hash fr.

(az elembevitel is így kell történie)



Kérés: ennek ellenétje, ott van-e?

Cíki: nem egyértelmű a lépés

Megoldás: lin kereséssel az első
üres helyre betesszük $K2$ -t



Keresés: lin. kereséssel kell dolgozni.

Ritka a adatbázisnál jó, egyébként minden bonyolult.

pl. törlés: amit kitöröl, azt, ami ütközött vele, nem fogom megőrizni.

Hash fr: pl kulcs % $N \rightarrow 0 \dots N-1$ a lehetőségek értékei.

Kulcs: pozitív - értelmes
negatív - értelmentelen
0 - üres hely

Tags: enum status { valid, invalid, empty }

megértés, azt tudni, hogy a többi statusa milyen.

Karbantartás: invalid elemekből empty, ha nincs mellette ütközött elem, ha
van, akkor azt átmozgatjuk. $\rightarrow$ időigényes.

Keresés: • lin / lin + stríza $O(N)$

- bináris / logaritmikus $O(\log_2 N)$

↳ yeniden tümü keş

- hashing ($\in$ & Ertékü kulcsokból névleges kód $\Rightarrow$ ütközések $\Rightarrow$ törlés
 jelezni kell, hogy üres, erőltetés vagy érvénytelen adat
 NULL $\oplus$ $\ominus$ bonyodalmatok ok)

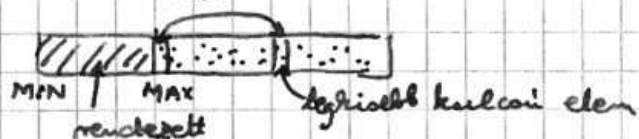
Renderiser : sorts.c

- **Q SORT** : gyors rendező alg.

- SHELL

- KUPAC (HEAT)

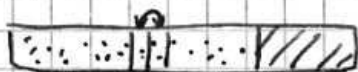
- Közvetlen kiválasztás rendezés: helyben rendezés (minimális sebesség függ a tömbre)



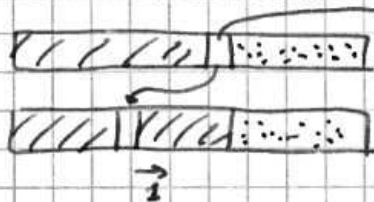
mindig megkeresem a legkisebbet; at első = min, majd ha $newlyt > első$
 $newlyt = min$

$O(N^2)$ - nagyágrandjék értéke

- Bubóné - rendezés



- Körperzellen bestimmen



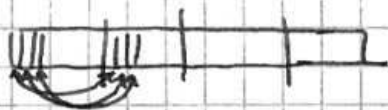
megkeresni a helyét a rendeltetésében.

$O(N^2)$ not adata megátással! $\rightarrow$ trükközni fogunk.

- SHELL - rendezés

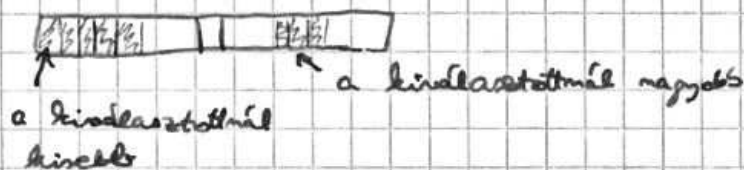


összehasonlítjuk, ha kell, csereünk $\Rightarrow$ majd $\frac{\text{táblaméret}}{2}$ -vel cserél

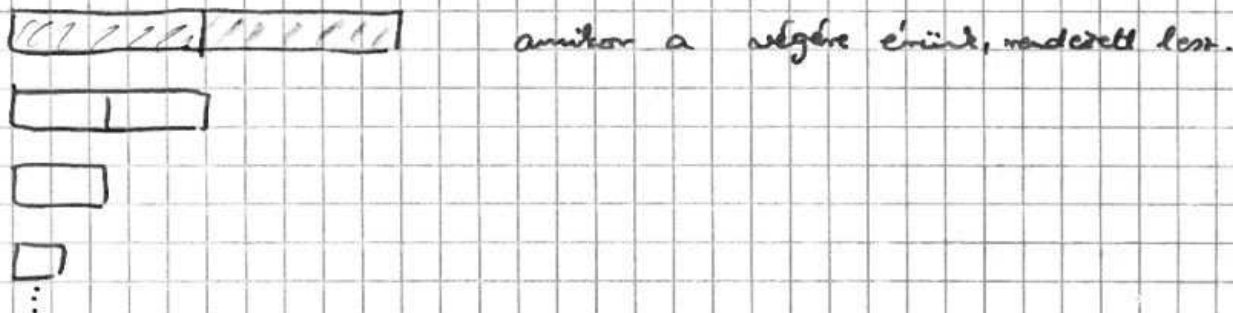


addig feleztük, míg 2 nem marad
 $O(\log N)$ lesz.

- GYORSRENDEZÉS / Quick sort (QSORT)



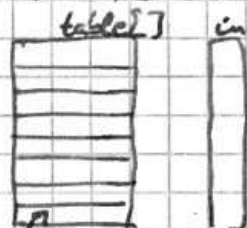
ALAP. SZÉTVALÓGATÁS



- megoldható: ciklussal
- rekurzívan

- "Hírléközi" lehetőségek

a) Index társult használat



lehet, hogy
 nagyon sok elem,
 nem célszerű
 tárolni

táblázat indexelése, $table[index[i]]$

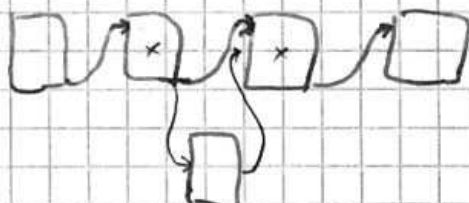
ezzel nyúlunk a tábl. elemekhez,
 hasonlítás

Rendezés előtt: $index[i] = i$

Adatmozgatás: $index[j] \leftrightarrow index[k]$

Élég csak az index-sorrendet megállapítani a
 rendezettséghez.

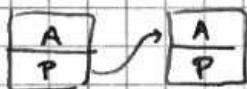
b) Fogjuk fel a rendezett adatokat listaként



(a kisebb kulcsú elem nyomtatás a ná
 következőre)

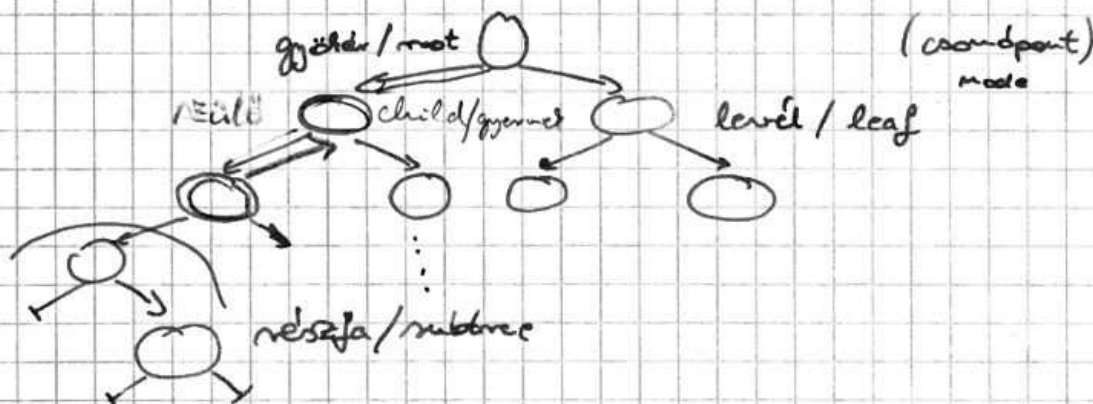
↑ ezt akarom beszúrni, megkeresem, hova

- adatmozgatás nélküli beszúrás

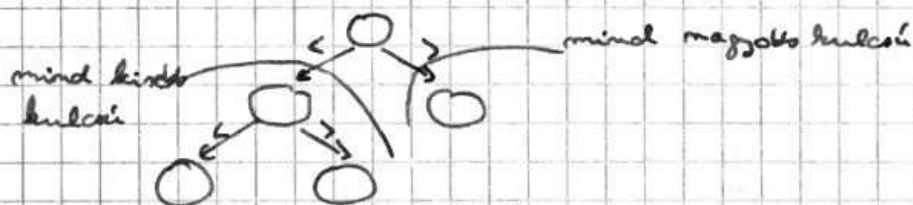


kérdés, hogy legyenek együtt.

c) Bintree's bináris fába



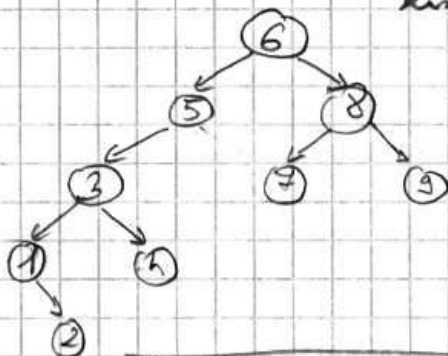
Rendezett bináris fa



ha ez minden részére igaz => rendezett bináris fa

6 5 3 8 9 1 4 2 10 7...

kis kulcsot műveztet a baloldali részre, nagyobb a jobb oldalra



Adatát kiíratásra: fa teljes bejárása

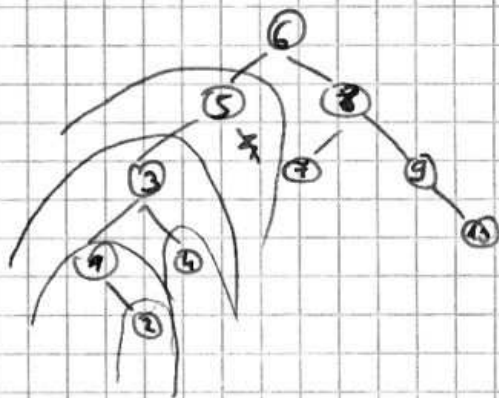


kiíratom a tartalmát



kiíratom a gyermekét majd önmagát.

Ált: ¹ kiíratjuk a baloldali részét, majd ² a teljes tartalmát kiíratjuk, majd magát a részét is ³ a jobboldali részét is.



1	2	3	4	5	6	7	8	9	10
---	---	---	---	---	---	---	---	---	----

Struktur

pl: 50 kar: en marek listja.

```
struct nl {  
    // ...  
    char mdr[51];  
} * lower; //
```

← (adapters)

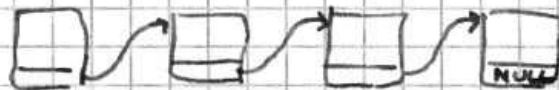
$\text{első. } \text{újraírtás}$ $\leftarrow$ painter
 $\text{első. } \text{már}$ $\leftarrow$ adatok
 $\uparrow$
 $\text{mezőhivárató operátor}$

(strcmp
strcpy)

```
struct szl { int xam;
             struct szl * konvettexo;
             } elsez, * futu;
```

```
else szam = 5;  
else koveltes = NULL;
```

Lincoln lists



bejdnář:

```

    futo = & else;

```

```
while (futura != NULL)
```

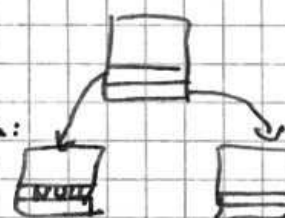
$$\{ \text{futo} = (+ \text{futo}). \text{konvertieren};$$

3

Több láné is lehet.



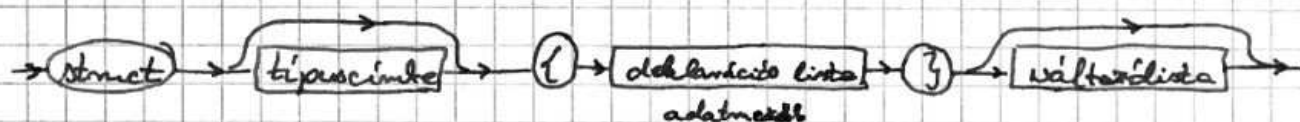
चरित्र :-



Newell
Indoi, hairy
cotton leaves

DYNAMIKUS
ADATZERKER

struktúrák



```

struct sz { int szam;
              struct sz * kovetkezo;
            } elsorz, * eleje, * futo;
int n;

```

```

elsorz. szam = 0; n = 0; elsorz. szam = NULL;
    ↖ mezőkiegészítő operátor

```

```

eleje = futo = & elsorz;

```

```

do { scanf ("%d", & n);

```

```

if (n)

```

```

{ p = (struct sz) malloc (sizeof (struct sz));

```

```

if (p != NULL)

```

```

{ (*p). kovetkezo = NULL;

```

```

  (*p). szam = 0;

```

```

  (*futo). kovetkezo = p;

```

```

  (*futo). szam = n;

```

```

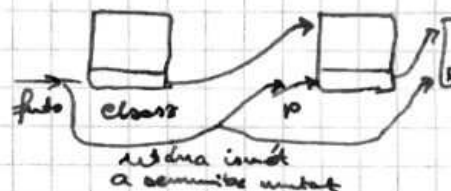
  futo = p;

```

```

} } while (n);

```



operátor: → a pointer által mutatott adatot jelenti

pl:

```

futo = eleje;

```

```

while (futo != NULL)

```

```

{ printf ("%d", futo -> szam);

```

```

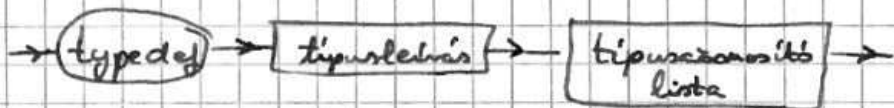
  futo = futo -> kovetkezo;
}

```

Lehet megmutatni: (x/sp) , adatkészlet
 $sp \rightarrow$ adatkészlet

Tipus de finalés

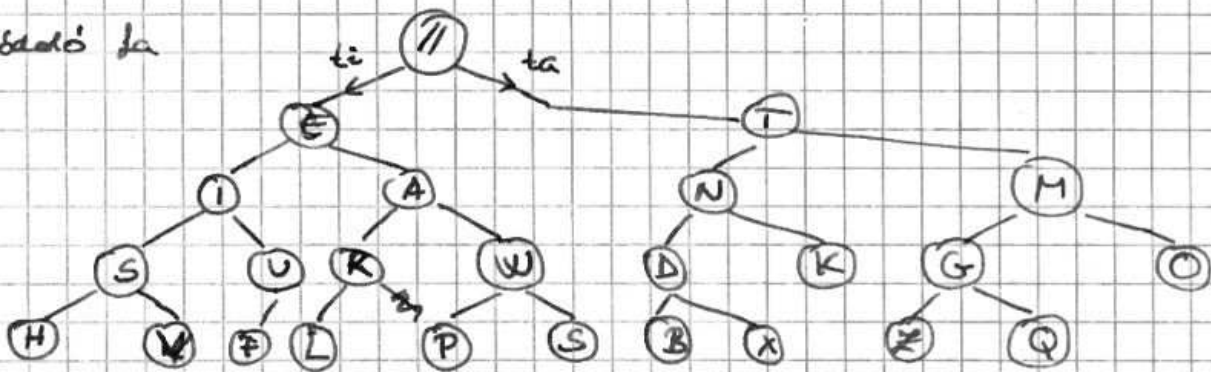
typedef int egesz;
 ↓
 új típus belső
 nevezzük
 új adattípus (definíció) azonosítója



```
typedef struct b { char ch;  
    struct b* ti, *ba;  
    btree;
```

→ brain's fa

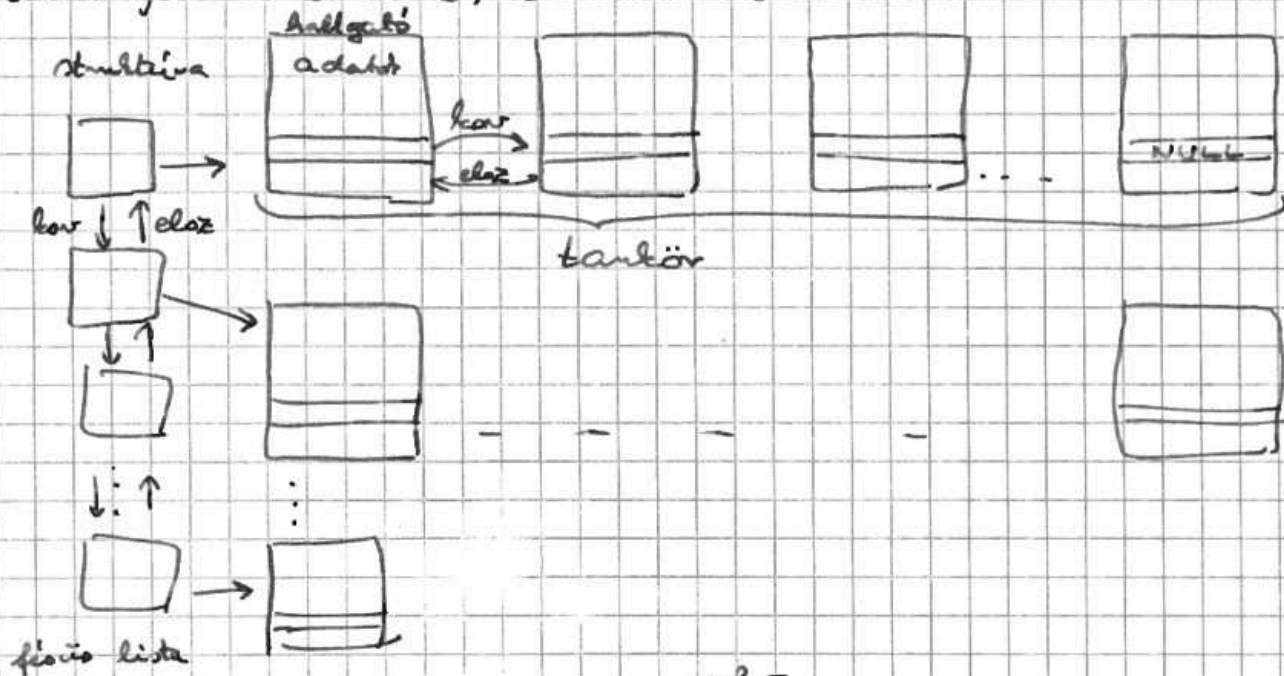
Morse - debadolo la



Pl: F: ti-ti-ta'-ti

HF: dekodolót írni a fával.

teljesítményes vállalkozás, ezekben tevékenység



```
#define NEVH 50
```

Prog. elmélet

2007.11.5.

Dr. Papp András

```
typedef char nev_t [NEVH+1];
```

```
typedef struct hallg_str { nev_t nev;
```

```
double atlag;
```

```
struct hallg_str * kor, * elo;
```

```
} hallg_t, * hallg_poi;
```

struktúra

struktúrapointer

tipus

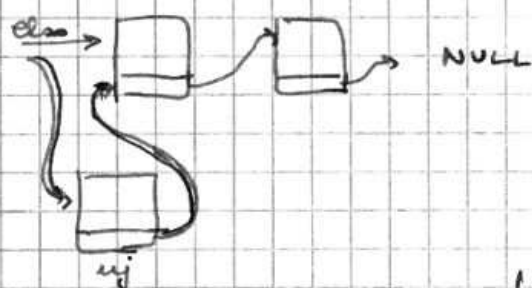
struct tkor_str

```
typedef { int korsz; hallg_poi hallgatok;
```

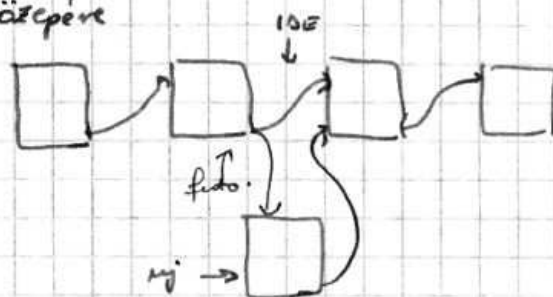
```
struct tkor_str * kor, * elo;
```

```
} tkor_t
```

Adatolt lista elejére új elem.



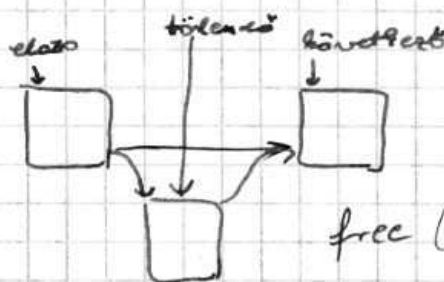
Közepére



végére:



Törles



free (törlando)



```
typedef struct l { valami adat;
```

```
struct l * kovetkezo;
```

```
} lista_t, * lista_poi;
```

```
lista_poi torles(lista_poi, elso, torlando);
```

```
{ if (elso == torlando) elso = torlando->kovetkezo;
```

```
else { elso->kovetkezo = torlando->kovetkezo;
```

/** !!! */

```
free (torlando);
```

```
return elso;
```

```
}
```

```
int ifunc (int a) {
    // típusbeírás
```

```
    return egész-kif;
}
```

```
b = ifunc(1);
    // függvényhívás.
```

```
typedef int ifunc (int);
```

ifunc f1, f2, f3; ← beírás, azt formailag mondja meg, hogy ezeket várja és ad vissza.

```
typedef ifunc * ifptr
```

(egész típusú f1, f2, f3-aknak tárolására alkalmas)

```
ifptr a1;
```

```
a1 = &f1;
```

mindkettő jó

```
ifptr fuggvenyek [ ] = { f1, f2, f3 };
```

```
int f1 (int x)
```

```
{ printf ("első f1 %d", x);
```

```
    return x+1;
}
```

```
int n;
int x, y;
```

```
scanf ("%d", &n);
```

```
if ((n >= 1) && (n <= 3))
```

← f1 pointer tömböt megindokálni → f1 első leper

```
y = (*fuggvenyek[n-1]) (n)
```

maga a f1

← definiálom → értéket ad vissza, ezt alapja y.

```
struct mtr { char key;
              char prompt[100];
              ifptr fuggveny;
              int count;
}
```

```
p1: (*fuggvenyek[n-1]) (1);
```

```
string.h -ban
strcmp (char *a, char *b)
```

0
-1
+1

```
float  
double } statisztika
```

Fü. pointerok

```
typedef int ifunc (int);
```

```
typedef ifunc* ifuncptr;
```

```
typedef struct m { char key;
                  char prompt[100];
                  ifuncptr function;
                } memitem;
```

egységes zárva
encapsulation

C++ class

```
memitem menu [ ] = { ... }
```

ide függvények jönnek, menu listát ad lehet meghívni

definíció/dt
()

felhívható kód
()

[]

[]

*

*

```
int fu (int x)
```

```
{
  :
  return ;
}
```

egész lif

"KISZÁMOLNAK" nemit

↳ érték, cím, végrehajtás
sikeressége

scanf : azt mondja meg, hogy
a beolvasást követően
kódját sikerült elvégezni

~ Pascal: "FUNCTION"

```
void main (void)
```

```
{
  :
  return;
}
```

Szegmentálás : hogy rövidítsen legyen a main

"PROCEDURE"

A fu-é paraméterei

- a fu-é bemenői; amivel majd nemit csinálunk

```
int max (int a, int b)
```

```
{
  int i, j;
  return (a > b) ? a : b;
}
```

formális
paraméterek

mutatók, érték, cím, kivétel, függvény

lokális változó : amikor a fu aktív,
konkrét példányok születnek meg,
a végén elhalnak

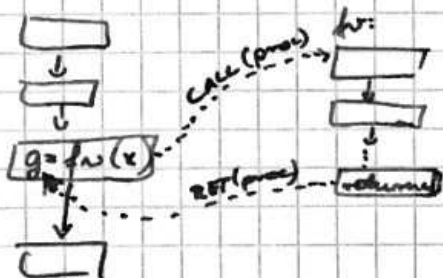
kívülre nem lehet rájuk hivatkozni

Aktuális paraméterek: a konkrét pm-t, amelyet behelyettesítünk

$$pl: x = \max(\underline{y}, 1);$$

Itt fv-hívás / fv-aktivizálás folyamata.

Hívó program:



Stack (verem)

visszatérési cím

stack frame

- Folyamat:
- visszatérési cím megállapítása
 - visszatérési cím számára hely
 - paraméterek számára helyfoglalás a stack-en
 - akt. pm-é értékei beírásuk
 - végül megkapja a fv kódja

- helyfoglalás a stack-en a lokális változókat számára
- végrehajtja a fv-t alkotó kódot
- return-ben specifikált értéket a stack-re másolja
- felszabadítja a stack-en lefoglalt területet:
 - lokális változó
 - paraméterek
- visszatérési cím elővétele, folytatás st.

C-hívási konvenció

Pascal-hívási konvenció

Assembly + C

- } • pl különbség, hogy melyik fv foglalja a helyet
 ↑ csak jelezni kell, hogy más nyelvi kódja
 Assembly + C : kell figyelni erre!

Paraméterátadási módok

- Érték szerinti paraméterátadás: az aktuális paraméter értéke a stack-en lefoglalt helyre másolódik.
 ↗ rekurzívnál jól kihasználható
 ↘ akt. pm-en és fv-nél gáz lehet
- Cím szerinti paraméterátadás: nem nagy adatbájtárat adunk át, hanem csak egy referenciát (a címét)

pl: Pascal-ban általában cím szerinti átadás, hívva a var pm-é értéket
 not-c.

C-ben csak érték szerinti pm-átadás van.

const char* mit char* konst
char my_strcpy(char mit[], char konst[]).

{ ... }

char str1[] = "hello";

str2[100];

my_strcpy(str1, str2)

véletl. változó, nem változhat meg az értéke
egy konstans, aminet nem címe (konstansnak
mintha címe)

Függvények - tömbök : tömböt mindig cím szerint

Főbb értékek visszajuttatása a külvilágba:

cím szerinti paraméterátvitel

int retval, error;

error = függvény (&retval);

int függvény(int* vissza);

{ *vissza = ...; }

return ...;
 ↖ hibahiba

Változó lokalitása, élettartása, láthatósága

ifunc f1; /* deklarálás! */

int x, y;

int f2(int a)

{ int x;
: x=2; y=1;
: return ...;
}

így lehet kifelé értéket adni
de ne használjuk!!

scope ()

{ int a, x;

} ↑
na ez az ami hiba
lehet.

void main(void)

{ x=2; y=1;

printf("%d", f2(3));
}

- a pm-k ott vannak, ahol deklaráltak.

Tárolási módok:

Lokális

- auto (alapért) : megszűlésekor elpusztul
- register : KÉRÉS ← nem a stack-ben, hanem a CPU registerben, ha lehet

pl:

```
for(i=0; i<100; i++)  
  for(j=0; j<100; j++)  
    for(k=0; k<100; k++)
```

← a k - hoz 10^6 -szor kell hozzáférni,
ilyenkor jó a register, mert gyors, 0 a
hozzáférési idő.

- static (nem auto) : örök életű lokális változó
nem a stack-en jön létre, korlátozott
látókörű

Ismétlés

függvény → tárolási egység

↓
azonosító

számként jelölés

() fv típus lépés op.

→ alaptípus → amilyen értéket a fv visszaad.

→ formális paraméterlista

paraméterek azonosítójával → def.

↳ C++: paraméter szignatúra

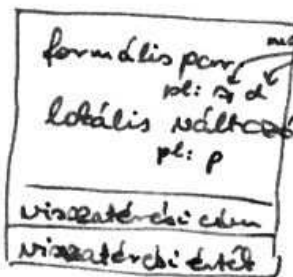
Deklaráció:

char * mystrcpy (char *, char *);

Definíció:

```
char * mystrcpy (char *s, char *d) {
    char *p = d;
    while (*d++ = *s++);
    return p;
}
```

Stack frame



érték szerinti pm. átadás

char * s1 = "hello"; s1[20];

mystrcpy (s1, s2);

↑
akt. paraméter

lokális / globális változó

élettartam, láthatóság

int x, y, a;

int ifunc (int a)

```
{ int x, z;
  static int t;
  { int y, z;
```

(= auto int x, z;)

register int x, z;

használat: x, z, a, y

használat: y, z, ~~x~~, a

auto tárolási osztályok
(megaműködés, meghaladás)

Rekurzió

— amikor egy fv. önmagát hívja.

$n! = \begin{cases} \text{ha } n > 1 & : n! = n * (n-1)! \\ \text{ha } n = 1 & : n! = 1 \end{cases}$ ← "eggyel egyszerűbb eset"

triviális megoldás

if () ↗ triviális
↘ rek. fv. hívás

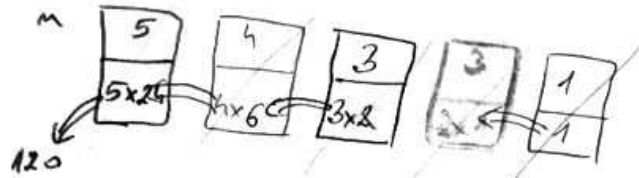
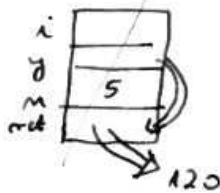
int rek_falt (int n)

```
{ if (n > 1) return n * rek_falt (n-1);
  else return 1;
}
```

átrekurzió : ami ciklusos is megoldható



m!	ciklusos	rekurzió
aktuális "m"	i	stack-en létrehozott m
aktuális index	y	stack-en a visszatérési érték
bemenő m	stack-en m	stack, lárd lent
	3+1	1+1



max mem
felhasználás:

4

10

használatos : útbejáró eljárások

pl: fábejárás — triviális eset: nincs része NULL

int fábejáró (pbtrec * root)

```

{ if (root == NULL) return 0;
  else { return 1 + fábejáró(root->bal) + fábejáró(root->jel); }
}

```

Változó paraméterlistájú fn-k

valami függvény (első param, ...)

int printf (char *fmt-str, ...)

int scanf (char *fmt-str, ...)

azt jelöli, hogy még hány paraméter van

és még további paraméterei lehetnek

tehetőleges konverzió

% adatkonverziós utasítások alapján kiírní. visszatérési érték, sikeresen végrehajtott adatkonverziót jelöl.

; 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

printf ("%d! értéke = %d \n", m, y);

↳ visszatérési érték: 2

scanf ("%d", &n);

↳ ret: hány konverziót hajtott végre sikeresen.

% □

± (jelölés vagy balra igazítás)

0-nál kisebb (negatív)

0-val töltse fel (nulla), ha nincs, akkor space.

%d : decimális egész

%x : hexa egész

%X : HEXA egész

%e : valódi, normál alakban

%f : valódi, fix pontosan

%g : valódi a 2 között

%c : karakter

%12.4g
↑
12 egész pontosság

Változó pn-ű fnt-ek.

valami függvény (első pn, [...])

printf for család < stdio.h >

int printf (char * fnt, ...);

stdout

int sprintf (char * s, char * fnt, ...);

string

int fprintf (FILE * fp, char * fnt, ...);

file

Mit tartalmazhat a formátum setting?

- tetszőleges kar → kimenet
- % % → maga a % jel
- % _ → adatkonverzió

% flag → mezőszélesség
 ✓ → .p, dőlt vezető
 → pontosítás
 + jobbra igazítás
 - balra igazítás . * -

d, i int → dec

O int → octális

x, X ~~int~~ int → hexa, HEXA

u int → unsigned

c int → kar (ASCII)

s char*

f double → [-] m.ddd

e, E double → Elm. ddd e ± XX
 E ± XX

G, G double → vagy e, vagy f

(p) void* → memóriacím

Pl : %03d
 %15d
 %12.5g
 %10s

vezető 0-t

15 karakter, space-ok

12 szám, 5 tizedes pont

10 szűk helyre string

- fnt. string változó is lehet.

(- %*d : nem mond meg a mezőszélességet, a pn-lista első eleme adja meg)

a scanf család

int scanf (char * fnt, ...)

stdin -ről

int sscanf (char * s, char * fnt, ...)

stringből

int fscanf (FILE * fp, char * fnt, ...)

file-ből

Ca ritkaság konverzió nélkül

d decimális → int*

i oktális/hexa → int*

O oktális → int*

u unsigned egész → unsigned int* -

(scanf ("%d", &n));

x hexa $\rightarrow$ int *
 c char $\rightarrow$ char *
 s string $\rightarrow$ char * !!! méret 
 ↳ whitespace-et által határolt tömb
 ferg valószínűleg $\rightarrow$ float *
 lf, lg, lg "kettős" $\rightarrow$ double
 ↳ valószínűleg

Saját változó pm-számai fr-h <stdarg.h>

változó pm-lista kezelőkre $\rightarrow$ típusok
rutinok

va_list típus, a pm-lista kör. elemére mutató pointer típusa
va_start (va_list ap, elsőparam)

rutin, inicializálja a pm-lista kezelőket

va_arg (ap, típus)

elérhető a kör. paraméter, dynamic konvertálják, amit megadunk.

va_end (ap)

talán túlságosan magunk után ($\sim$ free (p))

(P) d, f, s, c konverzió:

#include <stdarg.h>

void myprintf (const char *fmt, ...)

va_list ap;

char *p, *sval;

int ival;

double dval;

va_start (ap, fmt)

for (p = fmt; *p; p++)

{ if (*p != '%')

{ putchar(*p);

}

else switch (*p)

{ case 'd' : ival = va_arg (ap, int); printf ("%d", ival); break;

case 'f' : dval = va_arg (ap, double); printf ("%f", dval); break;

case 's' : for (sval = va_arg (ap, char *); *sval; sval++)

{ putchar(*sval);

}

case 'c' : ... break;

default : putchar(*p);

}

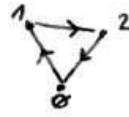
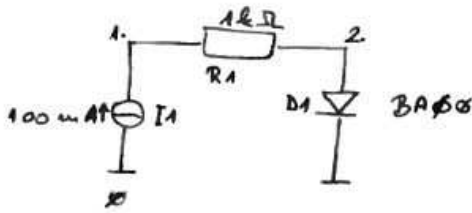
}

va_end (ap);

}

A union fogalma

1 adatkor leírás értelmezés



kezdő pont végpont el típusa
a gráf leírása

	kezdő pont	végpont	el típusa	elemértéke
I1	0	1	CURRENT (áramforrás)	1e-
R1	1	2	RESISTOR (ellenállás)	1e3
D1	2	0	DIODE	BA68

probléma: különböző típusok

```
typedef enum { real, string } attr_type;
```

```
typedef struct { char label[10];  
    int from, to;  
    attr_type valuetype;  
    attribute value;  
} element_type;
```

```
typedef union attribute { double elementvalue;  
    char elementtype[sizeof(double)];  
} attribute
```

(= értelmezés *)
1x pont 8 byte, de így nem használjuk *

```
double s; char ssp[8];
```

```
element_type netlist[ ]
```

```
if (netlist[i].valuetype == real)  
    s = netlist[i].attribute.elementvalue;  
else
```

```
strcpy (typ, netlist[i].attribute, elementtype);
```

adott union értelmezés neve

Érd meg tudni némi egy valóban 5. byte-ját is: union, ami vagy tömb, vagy double szám.

Kapcsolódás az OS-hoz

A main függvény

- void main(void)
 - int main (int argc, char** argv)
- argumentumok értéke
char** argv
argumentumok száma: argc >= 1

argv[0] → a program neve

myprog.exe C:\> myprog.

```
int main (int argc, char** argv; char* envp[])
{
    return 0;
}
```

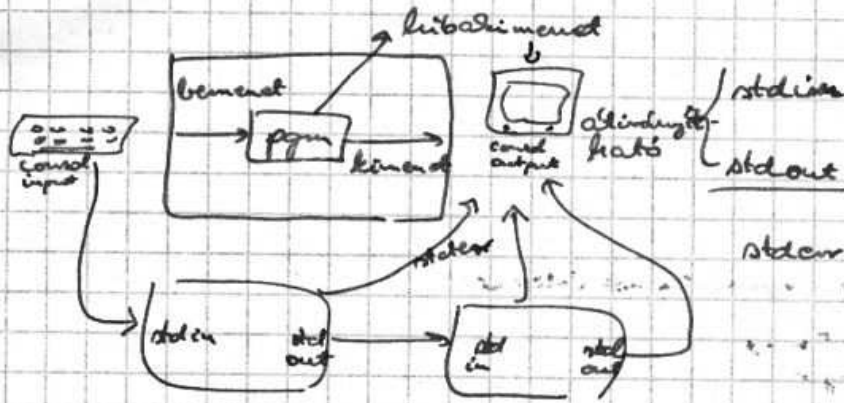
OS-től
OS-től függő környezeti változók
OS-nek

C:\> SET KÖRNYVALT = érték
envp[i]

EET hívja meg az OS

```
int main (int argc, char** argv[], char* envp[]) /* hívja az összes
                                                    változót */
{
    int i; /* PRIMUS INTER PARES */ = első az egymást követő
    for (i = 1; i <= argc; i++)
        printf ("%d argumentum = %s\n", i, argv[i-1]);
    i = 0;
    while (envp[i] != NULL)
        printf ("%s\n", envp[i++]);
    return 0;
}
```

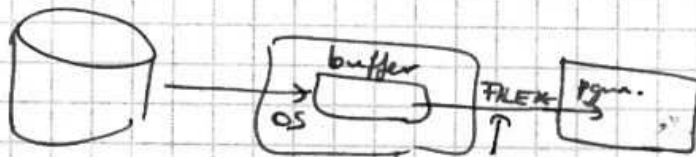
A file-ok: adatállományok



C:\> pgm 1 1, pgm 2 int ch;

C:\> (mypgm < input.txt) > output.txt

Levegőben szétváltatás hirtelen



amikor megérkezik
az ENTER-t, akkor
üntet ki a buffer
(flush), és tovább megy
a pgm.-es adatbázis

stdio.h

FILE struktúra

- ↳ hibaelőzet
- ↳ EOF állapot
- ↳ file pozíció
- ↳ buffer

OS-től függ

FILE * fp.

FILE * fopen (char * name, char * mode)
a file neve : a megnyitandó

"r" "rt" "rb" text mód: } "\n" konverzió

"w" "wt" "wb"

bináris mód

\n → CR

← CR LF

NULL :: nem sikerült megnyitni

→ FILE * fin, * fout;

fin = fopen("input.txt", "r");

if (fin == NULL) fprintf(stderr, "input.txt not found \n");

fclose { fprintf(fin, " ", ...)

fclose(fin);

fout = fopen("output.txt", "w");

if (fout == NULL)...



Runtime

Olvasási módok

getc

getchar

scanf

gets

FILE "

fgetc

fgets

fscanf

Írás

putc

putchar

printf

puts

fputc

fputs

fprintf

Más pgm-ek kivétel

<process.h>

system("OS parancs")

spawn

exec → pgm. lista arg, argv