↓ olv!

vnév 1 : típus 1

vnév 2. típus 2 absolute vnév 1.

↳ ha kisebb, akkor túllóg a vnév 1-en, de mindenképpen ugyanott kezdődik.

vnév 3. típus absolute tárcím

ennek a változónak a kezdete a tárcím kezdete

Van 3 rendszerváltozó:

— MEM [index] : tárcím

- a változót nem kell külön deklarálni
- egydimenziós tömb, ami a memória egészét lefedi
- a tárcím megjelöli azt a byte-ot, amivel dolgozni akarunk

— MEMW [index]

- lefedi a memóriát word tárcímmel

— MEML [index]

- longint adatokkal fedi le a memóriát

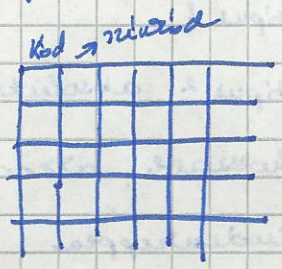
képernyő. par ⇒ programvizsgálat!

képernyő. par ⇒ MEM változóval

A negyedikhez hozzáadjuk a (sor - 1) * 160 - t.

Ha a 4. sor 3. karakterét akarjuk feltölteni:

3,4
↓ →
3. kar 4. sorban



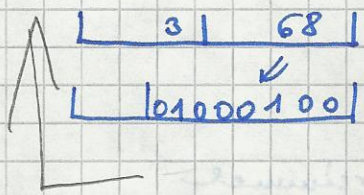
Ha a 4. sorba írunk: át kell lépni a 3x160-ba és a (j-1). helyre kell írni.

Minden karakterhez 2 byte tartozik: kód és színkód

MEM [$b800 : (i-1)*160 + (j-1)*2] := c

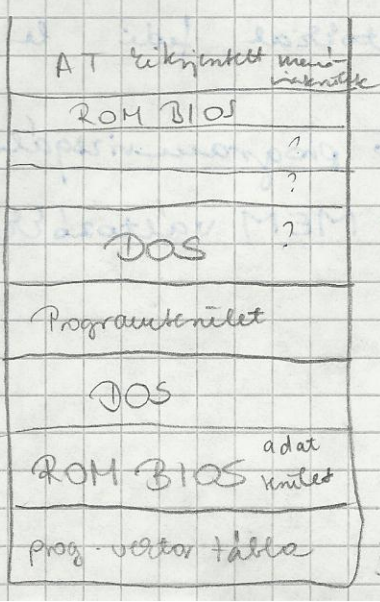
MEMW: word típusú változóba jelöl ki → ez 2 byte-os egység!

Arror a kódot: * kell 256-tal + kód (szín)



Memóriafelosztás:

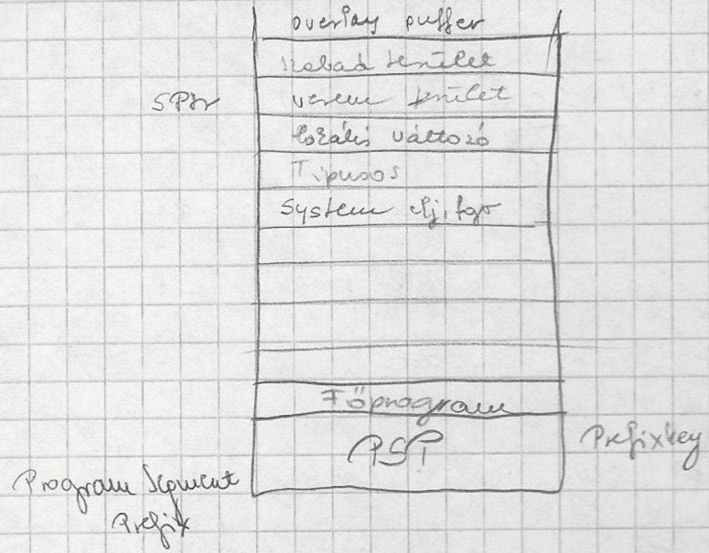
Assembly Esetében Programozás I.-II.



Prog. I. 1310

progr.

Prog. 11. 13³⁰



8. előadás

III. 26.

Dinamikus tárkezelés

Pointer típusú változó:

a, típusos pointer

b, típus nélküli pointer

folyamat mutat: pointer

odamut : ^integer

↓

integer típusú pointer

Type tomb = array[1..50] of word;

ideamut : ^tomb

Akár típusos, akár típus nélküli a p : 4 byte, tartalma akármilyen tárolócím lehet.

mutat ← tárcím (0=nil)

↓ $2^{32}-1$ lehet a tárcím

$0 - 2^{32}-1$-ig

32 bitbe csupa 1-est írunk be

a mutatónak lehet értéket adni ⇒ lehet értékadás jobb v. bal oldalán, de kiírni és beolvasni nem lehet, csak használni (manipulálni).

értékadás: mutat:= addr (vált név)

a mutató felveni a tárolócím tartalmát

1 mutató átadhatja az értékét 1 másik mutatónak. ⇒

pl.: odemutat := mutat

mutat := Ptr (szegmenscím, offsetcím)
↓
pointer → argumentuma változó v. konstans

x: real;

mutat := addr (x);

y := x ⟸ y := mutat^
↓
mutató által mutatott érték

Dinamikus a tárkezelés

Előfordul, hogy a programba nem fér bele 1 adat. A pointer segítségével kiterjük 1 adatot.

new (mutató) → ha ez típusos, akkor annyi helyet foglal le, amekkora a típusa.

move (mut 1, mut 2, hossz)
byteban

↳ mozgassa át az adatokat mut 1 kezdettel a mut 2-által kijelölt adatterületre annyi bytot, amennyi a hossz.

- A hosszt le lehet kérdezni: size of (adatnév)
- Ha nincs szükség már erre az adatterületre (amit a mutató foglal) fel lehet szabadítani: dispose (mutató)

- dinamikus változókat deklarálhatunk program során is, míg a statikusnak a program futása során végig foglalják a helyek.

példa: mutat_1.pas

{$END} direktíva hiánya ....

MUTAT1.pas

Ha nem tud annyi HP-et kijelölni, ~~ak~~ amennyit ki akarnak, akkor hiba nélkül kijelöli azt a legnagyobb HP-et, amit tud.

tomb: ^tombtip: odamutat a tömb elejére, de tudja, hol a vége

new(tomb): a HIP-be lefoglal annyi helyet, amennyi a tömbhöz kell.

ha Talap: ^integer → 2 byte

Talap: ^tomb → annyit foglal le, amennyi a tömbnek kell.

inc(ossz, tomb^[i]) ⇒ ossz := ossz + tomb^[i]

Ha folytatódna a program, akkor be kellene írni a DISPOSE(tomb)-ot, mert felszabadulna a hely.

Konkrét eljárás: a lefoglalt helyet a progi, v. 1 másik progi tudja használni ⇒ le kell zárni

Az átlag 127-128 között ingadozik ⇒ a program jól működik

[keep]

Képernyő tartalmának elmentése a keep-be:

Kepment2.pas

Mielőtt kitesszük a másik menüt, a kell menteni a keep-be. Így ha a menü eltűnik, az alatta levő ott marad.

$6800 → képernyőmemória kezdőcíme

Láncok, listák használata

- Az adatok úgy tartanak egymáshoz, hogy az egyik adat mutatója rámutat a köv. elemre. ⇒ ez az EGYIRÁNYÚ LÁNCOLT lista
- Van 1 listafej: ez mutatja meg, h. hol kezdődik a lista
- az utolsó a nil-re mutat

pl.: 10 karakteres stringeket viszek be, a végét a * jelzi

program (dinlist2.pas)

kovet: pelem
↓
mutató, és a következőre mutat

a rekord típusú ~~mat~~ mutatót deklaráljuk

9. előadás

Mutató típusú változó → típusos
→ típus nélküli

mutat : ^ integer ⇒ integer típusú változó

kamut : ^ real ⇒ real típusú adatok kijelölésére nolgál 6 byte-ot kezel.

– Hol van az a hely a tárban, ahová mutat?

- előbb címet kell adni

mutat:=ptr (cím)
szegmens : ofs.

lehet ezután hivatkozni rá : (mutat ^)
ez egy változónevet helyettesít

Fontos, h. olyan tárcímet adjunk a mutatónak, amely valós adatterületre mutat, különben nem fog futni a program.

new (mutat) → így garantált, hogy a mutató benne lesz a heap-ben.
Ekkor a mutat^ olyan változó, amely a heapben ott található meg, ahová mutatunk.

a heapmutató a szabad terület elejére mutat.

Ha mutatóhoz tárterületet rendelünk, akkor annak típizált mutatónak kell lennie. Jelölése hivatkozáskor a cím=neve + „^"

Ha a mutatónév után „^" van, mindig azt jelenti, ahová mutat, így ez egy változó.

<u>program:</u> mut-szam.pas

p1^ : beolvasható ennek az értéke, és ezzel művelet is végezhető

deki.: var p1, p2, p3 : ^real;
v : ^char

new - vel lefoglalunk 1 helyet mindegyik mutatónak

readln (p1^) : olvasunk be a p1 által mutatott adatterületre

Ha erre nincs szükségünk, a program belül „dispose" -zal felszabadítható, és lehet másra használni.

A mutató nem lehet ciklusparaméter!

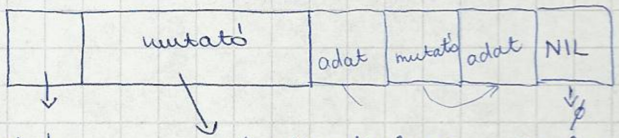
for i:=1 to 50 do → NEM LEHETSÉGES!

while i^ < 150 do → ez így már jó!

lehet növelni, lehet állást keresni, (csak a while ciklussal).

Lista kezelése

A lista elemeit kiírjuk a heap-be ⇒ láncolt lista



Adat — adattartalma a következő mezőre mutat.

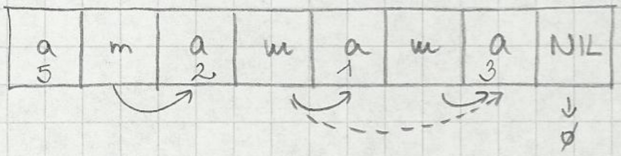
(string[10] (bármi lehet))

Ezután újra adat, mutató, ... stb, majd még egy adat

és utána a NIL, ami mutat sehová.

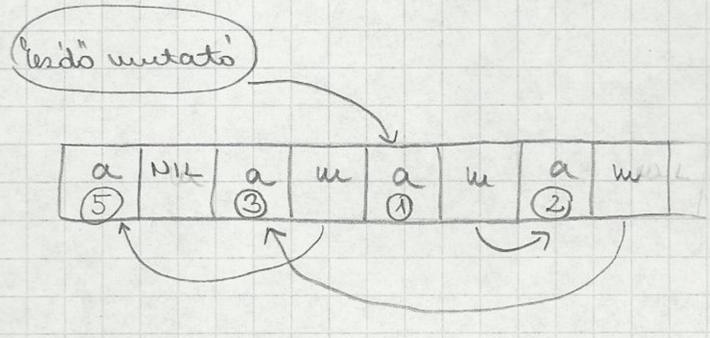
Attól dinamikus lista, hogy new-val foglalok helyet, és egyszerre föl, majd lezárom. Mindig dinamikusan nő a tár, de van határ, mert a heap sem végtelen.

pl.:



Ha törölni akarjuk az 1-est, akkor a mutató értékét át kell állítani, h. a 3-asra mutasson, és az 1-est dispose-zal törölhetjük.

Ha nem növ. sorrendbe akarjuk kiíratni, akkor mondjuk a kezdőmutatót nem az 1-re, hanem másra állítjuk. Szükséges feltétel a növekvő sorrendnél, h. a mutatók mindig nagyobb adatra mutassanak.



Ha rendezni akarunk nem az adatokat kell mozgatni, csak a mutatókat átrendezni.

pl.: 1, Hogyan lehet listát létrehozni?

↓

szekvenciális = az adatok egymás mögött vannak és azok mutatói a köv. adatra mutatnak. logikai és fizikai

2, Rendezett lista: csak logikailag len benne

3, Hogy lehet egyik, másik listát bővíteni?

4, Hogy lehet törölni?

5, Hogy lehet az állományokban keresni?

program: dinlista.pas

szekvenciális felvitel: mikor még nincs adathalmaz.

akkor elso := nil;

rekord adatmezejére hivatkozás ⇒ rekordnév.adatnév