1. előadás

2003. 11. 5

Kötelező: Fordán Tamás, Pomakási S.: Adatszerk és algor. !

Csőke Lajos, Garamvölgyi G.: A számítógép progr. logikai alapjai !

Zsakó László: Programozási feladatok I., II.

Lipschutz: Adatszerkezetek

Magyarné Huba Ágnes: Algoritmusok és adatszerk.

Algoritmus: számítógépes program → elemi lépések sorozata

adatszerkezet: Észlelfogható formára hozni az algoritmust

- a algoritmushoz illeszkedik az adatszerkezet
- a alg.-t és az adatsz-t a számgép főbb össze
- ∀ dolgot sorozatokkal jellemezhetők
  ↳ matematikai fogalom
  (sorozatokban közös, hogy az adatai azonos típusúak)
- A sorozat ⟨s⟩ elemei homogének (azonos jellegűek) → többféle különbség közöttük · milyen műveletek végezhetők velük

Egy adatszerkezettel megmutatjuk, h. 1 feladat is. oldható meg, de nem írunk rá más algoritmust, mert ugyanaz. (Nem egy adatszerkezetben valósítjuk meg.)

Fogalma, jellemzői:

pl: több szám kiválasztása, rekurzív algoritmus

Algoritmus: - ∀ probléma megoldására kívál módot.
- véges

alapalgoritmusok = programozási tételek ⇒ jól használhatók bonyolult feladatok mo.-ára

Legyen véges! ⇒ véges sok lépésszámban végrehajthatók (A végtelen algoritmus végtelen idő múlva hajtódik végre)

- egyértelmű

pl.: rekurzív algoritmus

Egyértelműen megállapítható, h. hol a vége, mi vele a cél
Minden adott lépés után el kell dönteni, h. mi következik

- determinisztikus
  ↳ jelentés: előre meghatározott

  ∄ nemi ellentmondás
  pl.: másodfokú egyenlet $a, b, c \Rightarrow x_1, x_2$

  • ugyanazt a kimenetet adja vissza ugyanolyan bemenet esetén

  - teljes

  pl.: másodfokú egyenlet

  • az adott példára vonatkozóan ne csak 1 adott számkombinációra tudja végrehajtani.

Megadásának módjai:

pl.: telefonfülke ⇒ piktogrammok

- ha futtatható állományt készítünk, az is egy megadási módszer

3 féle vezérlési szerkezet:

Szekvencia: utasítások egymás utáni végrehajtása

Selekció: az alg. választási lehetőséget ad

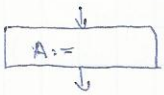
Iteráció: valamely utasítás (v. u. csoport) többször futtatható

Algoritmus:

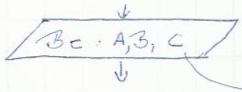
elejc

vége

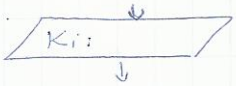
} miatt teljesül az egyértelműség



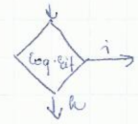
értékadó utasítás



Bemeneti utasítás (aminek kívülről várunk értéket)



Kimeneti utasítás
lehet: változó (A), konstans (5), kifejezés (a*5)



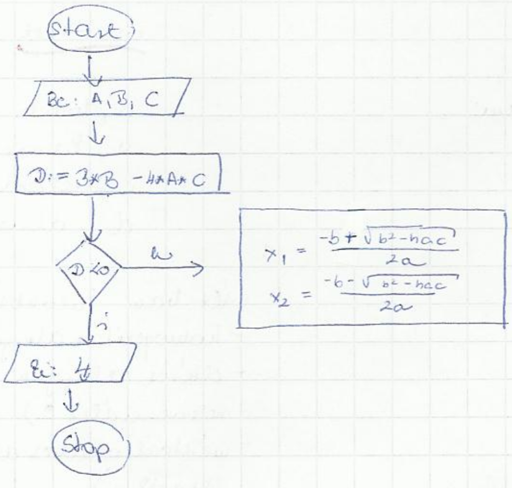
logikai utasítás

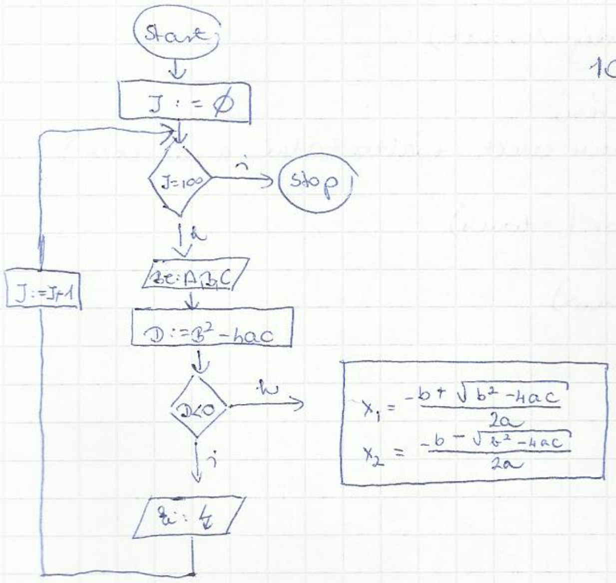
A rendszer előnye: mindent le lehet írni, de ált. -ban a bonyolultabb alg. leírására már nem alkalmas

szemléletes, egyszerű

csatlakozási pontok

Blokkdiagram: másodfokú egyenlet megoldóképlete





100 db másodfokút old meg

Típusok:

Meghatározza, h. elvégezhető-e a művelet, és mi a felvehető értékek halmaza

Fontos a megfelelő típus kiválasztása, mert:

- ha alulméretezzük a típust ⇒ adatvesztés lép fel
- ha túl nagy helyet hagyunk ⇒ leterheljük feleslegesen a gépet

(a lebegőpontos számok kódoltan több helyen tárolódnak, és vissza kell kódolni → idő- & memóriaveszteség)

Típusok

elemi
amelyeket adott szinten már nem tudunk tovább bontani

a, egész
b, valós
c, karakteres
d, logikai
e, mutató

összetett (elemi adatokból)

a, tömb
b, karakterlánc
c, rekord
d., állomány

Ált-ban különbséget teszünk:
- homogenitás alapján (azonos v. nem)
- elérés alapján (közvetlen v. szekvenciális *)
- méretváltozás alapján (dinamikus, statikus)

Tömb (sorozat ⇒ kitelernél mindig sorozat)
- homogén
- ált-ban közvetlen elérésű
- ált-ban statikus (nem lehet +változtatni a méretét)

String: - karakter típusú vektor (= tömb)

Rekord: - heterogén (nem azonos)
- közvetlen elérésű
- ált-ban statikus

* az előtte lévőket dolgozza fel

2. előadás

Adatok 3 féle lehetnek: a, konstans

b; változó

c; kifejezés

- Progr. nyelv: külön kezeli a konstansokat a változóktól.

  lesznek változó és konstansazonosítók

- felmerül a típus fogalma

  Mindennek van típusa.

  Vannak konstansok, amelynek külön azonosítója van.

  pl.: π

pl.:

```
for i := 1 to 100          kulcsszavak, azonosítók, konstansok
for j := 5 to 100

a : array [1..100]
```

Jobb, ha van konstantazonosító, mert ha módosítani akarom, csak egy helyen kell, és nincs keresgélés!

Pl.: → ha 2 db 100-ig menő ciklus van

A Hpussal meghat. a felvehető értéket halmazát, a műveleteket, és a tárolásukat.

Itt a felvehető értéken van a hangsúly és a műveleteken.

pl.: az oktályszatok tárolása pl.: byte

Típusdeklarációk: megadunk 1 v. több Hpusazonosítót
↳ „,"-vel elválasztva

és = - jellel a Hpus leírását

↳ lehet már korábban megadott Hpus: pl.: alap Hpus-ok

konstans konstansazonosító (vagy k.a. lista) = érték
↳ „ , " -vel elválasztva

változó változóazonosító : típusleírás

——— o ———

Szerkezet:

BE: változó azonosítólista;

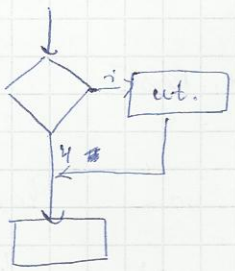
KI: adatlista;

(Kifejezés: Konst. és változók megfelelő műveleti jellel összekapcsolva)

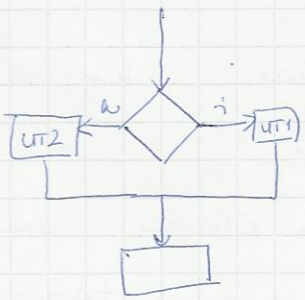
változóazonosító := kifejezés
↳ az ehhez tartozó memóriaterület (típusa) fog megváltozni

bemenetkor is a memóriaterület változik meg

- Ha <log.kif> akkor
  út;
  Ha vége;



- Ha <logkif> akkor
  út.1;
  különben
  út.2;
  Ha vége.



Az utasítás helyett szerepelhet utasítások csoportja is.

Iteráció:

Egy utasítás többször hajtódik végre

a, számlálós ciklus

már belépéskor tudjuk, hányszor hajtjuk végre

b, közben dönt ki, hogy kell-e végrehajtani többször

feltételes ciklus

Elöltesztelő ciklus (lehet, hogy 1x sem hajtódik végre)

Hátultesztelő -"- (min 1x végrehajtódik)

Be: A; csak akkor tudom eldönteni, kell-e újból

belépni, ha már 1x beléptem

megadás:

CIKLUS

⋮

C. VÉGE logikai kif. esetén

pl:

ciklus

BE: A;

CIKLUS VÉGE A<Ø esetén

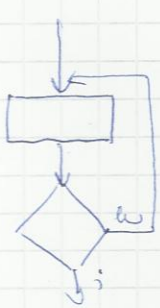
} hátultesztelő ciklus

CIKLUS MÍG log kif igaz

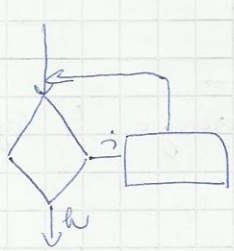
⋮

ciklus vége;

} elöltesztelő ciklus



hátultesztelő c.



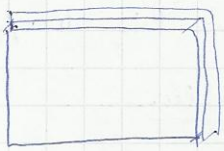
elöltesztelő c.

pl.: 

homogén adatok összessége

↓

1 ontály



↑

több ontályuál

DE: meg kell adni az „elérési utat"

Nem változott az adategyüttes homogenitása, csak az elérése bonyolódott

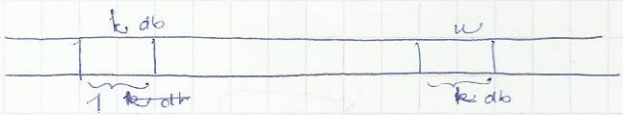
A tömböket dimenziószám szerint csoportosítjuk.

Megadjuk a tömb azonosítóját (jelöli az abba tartozó összes adatot), ha tovább keresünk, arról meg kell adni még infot. Ha 2 + infot kell → 2D (imenziós)
Ha 3 + infot kell megadni → 3D

stb.

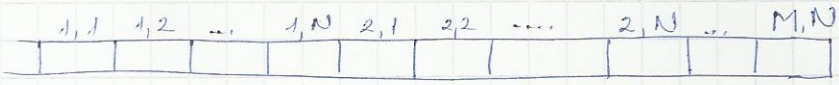
Tömb tulajdonságai, tárolása a memóriában:

- az összes tömb egy egybefüggő memóriaterületen van eltárolva.
- 1 byte-ot tudok megcímezni egyszerre



1 elem tárigénye 8 byte

Sorfolyó tárolási mód



ha az oszlopot tárolja: oszlopfolytonos tárolási mód

Lexigrafikus tárolás: betűrendes lista

pl: nótás, lexikon

címszavak → első néhány karakter azonos, a többi változik

Ugyanez vonatkozik a tömbre.

pl.:

MXN JEGY = EGÉSZ

Típus tanulótip = TÖMB[1..N] · JEGY ; → 1D -vektor

OSZTALYTIP = TÖMB[1..M, 1..N] JEGY ; → 2D

ÉVFTIP = TÖMB[1..K, 1.M, 1..N] JEGY → 3D

ISKTIP = TÖMB[1..L, 1..K, 1..M, 1..N] JEGY → 4D

típusazonosító indexhatár

VÁLTOZÓ BUGA JAKAB : TANULOTIP ; → az illető minden jegyét tárolja

pl.: BUGA JAKAB [2] → az indexhatárokon belüli értékek

OSZTÁLY : OSZTÁLYTIP

OSZTÁLY [5,2] → az 5. indexű tanuló, második tárgyának a jegyét jelenti

ÉVF : ÉVFTÍP

ÉVF [3; 5, 2]

↓

adott évfolyamon belüli a 3. osztály

ISK: ISKTÍP

ISK → minden évf., minden ost., minden tanuló, minden jegye

ISK [8, 3; 5, 2]

Alternatív megoldás:

OSZTÁLYTÍP1 = TÖMB [1..M] TANULÓTÍP → vektor

↓

OSZTÁLY1 : OSZTÁLYTÍP1

OSZTÁLY1 [5] [2]

vektor

Ha vektor adunk meg, utána mit fog adni 1 indexet.

hivatkozás az 5. tanuló 2. jegyére

ÉVFTÍP1 = TÖMB [1..K] OSZTÁLYTÍP1

ÉVF : ÉVFTÍP1

ÉVF1 [3] [5] [2]

PROGRAMOZÁSI TÉTELEK

a) Összegzés tétele:

Adott N elemű A sorozat

↓ ↓ ↓

végés neve homogén

A sorozathoz egy értéket rendel, amilyen értéktípusa = az elemek típusával.