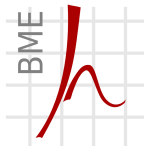


Bevezetés – Alapfogalmak

A programozás alapjai I.



Hálózati Rendszerek és Szolgáltatások Tanszék
Farkas Balázs, Fiala Péter, Vitéz András, Zsóka Zoltán

2020. szeptember 7.

Tartalom

1 Bevezetés

- Bemutatkozás
- Miről lesz szó
- Követelmények

2 Alapfogalmak

- Az imperatív programozási paradigma
- Az algoritmus

- Az adat – konstansok és változók
- Kifejezések
- Programnyelvek

3 C nyelvi alapok

- Történet
- Az első program
- Változók
- Beolvasás

1. fejezet

Bevezetés



Bemutakozás

- BME Villamosmérnöki és Informatikai Kar
 - Hálózati Rendszerek és Szolgáltatások Tanszék
- Fiala Péter – tárgyfelelős, előadó
 - email: fiala@hit.bme.hu
 - szoba: IE433
 - Tel: 06 1 463 2543
- Vitéz András – előadó
 - email: vitez@hit.bme.hu
- A tárgy honlapja:
<https://edu.vik.bme.hu>

Tárgyunk a BSc képzésben

Informatika II

Informatika I

Digitális technika II

A programozás
alapjai II

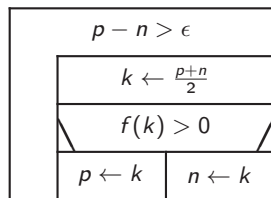
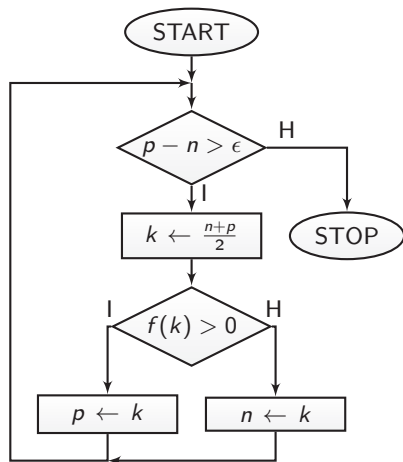
Digitális technika I

A programozás
alapjai I

A Számítás-
tudomány alapjai

Mit tanulunk meg?

- Az algoritmikus gondolkodás és strukturált programszervezés alapjait





Mit tanulunk meg?

■ A C programozási nyelvet

```
1  /* HelloWorld.c -- Az első program */
2  #include <stdio.h> /* printf-hez */
3
4  /* A főprogram */
5  int main()
6  {
7      printf("Szia, világ!\n"); /* Kiírás */
8      return 0;
9  }
```



Mit tanulunk meg?

- Vektoralgoritmusokat
 - Eldöntés
 - Keresés
 - Kiválasztás
 - Rendezés

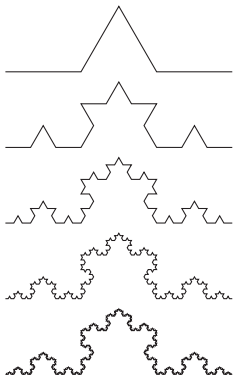


Mit tanulunk meg?

- A fájlkezelést

Mit tanulunk meg?

■ A rekurziót

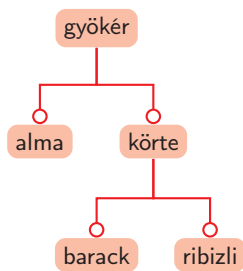


fig/rekurzio.png

Mit tanulunk meg?

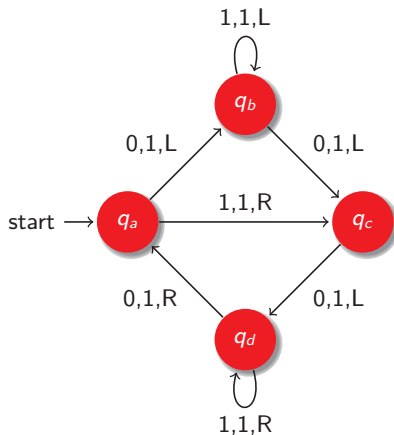
■ Dinamikus adatszerkezeteket

- Fák
- Listák
- Vermek
- Prioritásos sorok



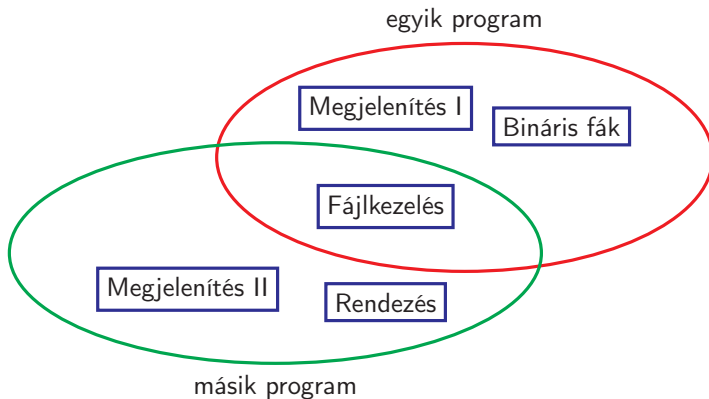
Mit tanulunk meg?

- Az állapotgépeket és az eseményvezérelt programozást



Mit tanulunk meg?

- A moduláris programozást





Követelmények

1 Jelenlét

- Előadáson
- Tantermi gyakorlaton (70%) – mi ellenőrizzük
- Laborgyakorlaton (70%) – mi ellenőrizzük

Már az első héten is megtartjuk mindhárom foglalkozást!

2 Számonkérések

- hat (3 · 2) kis ZH a tantermi gyakorlatokon
- két nagy ZH
- beugrók a laborgyakorlatokon

3 Házi feladat



Követelmények

- Kis ZH-k
 - (3 · 2) kis ZH a tantermi gyakorlatokon, előre meghirdetett időpontokban
(1 2) (3 4) (5 6)
 - mindegyiken 10 pont szereshető, páronként csak a jobb számít
 - követelmény: páronként min. 4 pont
- Nagy ZH-k
 - 2 nagy ZH előre meghirdetett időpontokban
 - papíron
 - mindegyiken 40 pont szereshető
 - követelmény: összesen min. 40 pont
 - pótlás: egyetlen pótZH-val



Követelmények

- Házi feladat
 - Az ismeretek készsége szintű megértését és alkalmazását elősegítő házi feladat, amely átfogja az adatszerkezet-tervezést, az algoritmizálást, az implementálást, a tesztelést és a dokumentálást.
 - Beadás:
 - 1 feladat pontosítása (7. hét)
 - 2 adatszerkezet és algoritmusok (11. hét)
 - 3 program és dokumentáció (13. hét)
 - Értékelés részletei a tantárgyi adatlapon
- Az elfogadott házi feladat előfeltétele a félévi jegy megszerzésének



Szorgalmasoknak

- Szorgalmi feladatokból kétféle van
 - 1 Laborvezetők által kiadott feladatok
 - Az labor órai anyaghoz kapcsolódó további feladatok
 - Beadás a Moodle rendszerben
 - A laborvezető értékeli és jelez vissza
 - A laborvezető összesen 3 pontot adhat, ami beleszámít a jegybe
 - 2 IMSc-szorgalmi feladatok
 - Beadás egy erre kijelölt felületen (később pontosítjuk)
 - Összesen 4 feladat, mindegyikért 5 IMSc pont kapható
- IMSc pontok szerezhetők a nagyzárthelyiken (6+6 pont)
- IMSc pontok szerezhetők a nagy házi feladat részletesebb és időben történő kidolgozásával (3 pont)

2. fejezet

Alapfogalmak



Programozás

Hogy készítsék tojásrántottát?

```
Forró olajban piríts meg kevés szalonnát  
Üss bele három tojást  
...
```

Hogy pirítsak meg forró olajban kevés szalonnát?

```
Végy egy serpenyőt  
Tedd a tűzhelyre  
Önts bele kevés olajat  
Forrald fel  
Tégy bele kevés szalonnát  
Várj, míg megpirul
```

Hogy forralom fel az olajat?

```
Gyűjts alá  
Várj kicsit  
Forró az olaj?  
Ha nem, ugorj vissza a 2. sorra
```



Az imperatív programozási paradigma

Programozás

Megmondjuk a gépnek, hogy mit csináljon

Programozási paradigmák

Azok az elvek, amik alapján a programot elkészítjük

- Imperatív programozás ← Ezt tanuljuk mi
- Funkcionális programozás
- Objektum-orientált programozás
- stb. . .

Imperatív programozás

Lépésről lépésre előírjuk, hogy a gép mit csináljon

- egy algoritmus megfogalmazásával



A programozás folyamata

A félév folyamán mindig ezt fogjuk csinálni:

- 1 Megfogalmazunk egy feladatot
- 2 A megoldáshoz kitalálunk egy algoritmust
- 3 Elkészítjük a programot – kódoljuk az algoritmust



A programozás folyamata

Ugyanaz kicsit bővebben:

- 1 Megfogalmazunk egy feladatot
- 2 Megadjuk a pontos specifikációt
- 3 Adatszerkezetet választunk a probléma modellezéséhez
- 4 A megoldáshoz kitalálunk egy algoritmust
- 5 Mi idén kizárólag C-ben programozunk
- 6 Kódoljuk az algoritmust
- 7 Teszteljük a programot

Algoritmus

Algoritmus (módszer)

Gépiesen végrehajtható lépések véges sorozata, amely elvezet a megoldáshoz

- Kódolás előtt meggyőződünk róla, hogy
 - helyes – tényleg azt oldja meg, amit szeretnénk
 - teljes – minden lehetséges esetben megoldja
 - véges – véges sok lépésben befejeződik
- Nem elég kipróbálni, bizonyítani is kell!

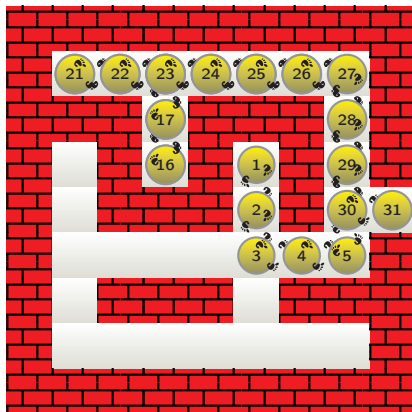


Algoritmusok – példák

- Feladat: Határozzuk meg az n szám köbgyökét!
- Megoldás: Osszuk el n -et négygyel!
- Tesztek:
 - 1 $n = 8, \quad n/4 = 2, \quad 2 \cdot 2 \cdot 2 = 8$
 - 2 $n = -8, \quad n/4 = -2, \quad (-2) \cdot (-2) \cdot (-2) = -8$
 - 3 $n = 64, \quad n/4 = 16, \quad 16 \cdot 16 \cdot 16 = 4092 \neq 64$
- Az algoritmus nem helyes

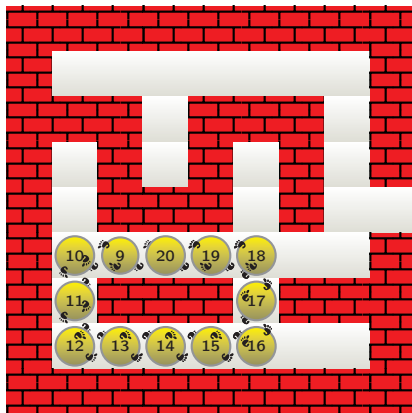
Algoritmusok

- Feladat: Meneküljünk ki a sötét labirintusból
- Megoldás: Bal vállunkat a falnak nyomva haladjunk, míg ki nem érünk.



Algoritmusok

- Feladat: Meneküljünk ki a sötét labirintusból
- Megoldás: Bal vállunkat a falnak nyomva haladjunk, míg ki nem érünk.



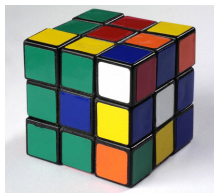


Algoritmusok – példák

- Attól, mert az algoritmus helyes, teljes és véges, még nem biztos, hogy kezelhető
- Fontos, hogy gyakorlatilag is véges legyen, vagyis
 - kivárható idő alatt lefusson
 - ésszerű mennyiségű adattal dolgozzon

Algoritmusok – példák

Isten algoritmus Adjuk meg azt a legkisebb lépésszámot, melyre igaz, hogy legfeljebb annyi forgatással a Rubik-kocka tetszőleges kezdőállapotból kirakható.



- 21 626 001 637 244 900 000 feldolgozandó állapot
- Ha másodpercenként 1 000 000-t dolgozunk fel, 685 756 évig tart.
- Az emberiség eddigi története $< 10\,000$ év



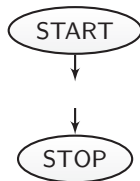
Algoritmusok leírása

- Algoritmusok nyelvfüggetlen leírási módja a pszeudokód (álkód)
- természetes nyelven, de precízen megfogalmazott utasítássorozat

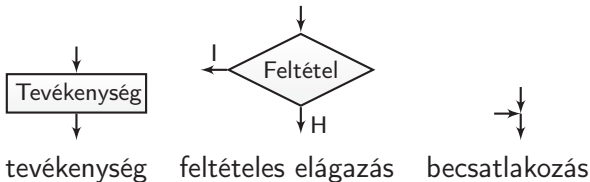
```
Végy egy serpenyőt  
Tedd a tűzhelyre  
Önts bele kevés olajat  
Gyűjts alá  
Várj kicsit  
Forró az olaj?  
Ha nem, ugorj vissza az 5. sorra  
Tégy bele kevés szalonnát  
Várj, míg megpirul  
Üss bele három tojást
```

Algoritmusok leírása

- Algoritmusok grafikus ábrázolásának eszköze a folyamatábra
- Egybemenetű és egykimenetű program folyamatábrája START és STOP elemek között helyezkedik el

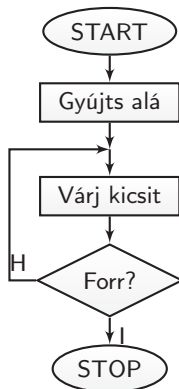


- A folyamatábra az alábbi elemekből épül fel



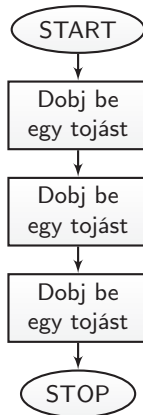
Folyamatábra – példa

- Adjuk meg a forralás algoritmusának folyamatábráját



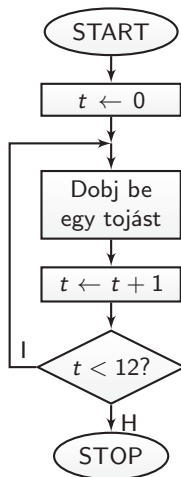
Folyamatábra – példa

- Hogyan dobunk be három tojást?



Folyamatábra – példa

- Hogyan dobunk be 12 tojást?
- Jelölje t a már bedobott tojások számát!





Az adat fogalma

- Az algoritmus adatokon, adatokkal dolgozik

Adat

Az adat minden, amit a külvilágból számítógépünkben leképezve tárolunk

- Az adatnak van
 - típusa (szám, betű, szín, ...)
 - értéke
- A típus meghatározza
 - az adat értékészletét
 - és az adaton végezhető műveleteket



Típusok – példák

típus	értékek	műveletek
szám	$0, -1, e, \pi, \dots$	összeadás, kivonás, \dots , összehasonlítás, rendezés
betű	a, A, b, γ, \dots	összehasonlítás, rendezés
logikai	{igaz, hamis}	tagadás, konjunkció (ÉS), diszjunkció (VAGY)
szín	piros, kék, \dots	összehasonlítás
hőmérséklet	hideg, meleg, forró, \dots	összehasonlítás, rendezés



Állandók és változók

Az algoritmusban betöltött szerepe szerint az adat lehet

- állandó (konstans)
 - értéke nem változik az algoritmus futása során
pl. a fenti példában 12 (a bedobandó tojások száma)
- változó
 - azonosítóval jelöljük (pl. t)
 - értéke műveletekben felhasználható (olvasás), pl. $t < 12?$
 - értéke frissíthető (értékadás, írás), pl. $t \leftarrow 0$
- Az állandó típusa a megjelenési formából kiderül
- A változó típusát mindig külön meg kell adni (deklaráció). pl. „Jelölje t a bedobott tojások számát”



Kifejezések

Kifejezés

Állandókból és változókból a megfelelő műveletek (operációk) alkalmazásával kifejezések képezhetők

- A kifejezés kiértékelhető, típusa és értéke van.
- A műveleteket operátorok határozzák meg, melyek az operandusokon dolgoznak
- Kifejezés példák

kifejezés	típus	érték	megjegyzés
$2 + 3$	szám	5	
$-a$	szám	-3	ha $a = 3$
$2 * (a - 2)$	szám	2	ha $a = 3$
igaz ÉS hamis	logikai	hamis	



Kifejezések

- Kifejezések típusa nem feltétlenül egyezik meg az operandusok (összetevők) típusával. Vegyes kifejezések:

kifejezés	típus	érték	megjegyzés
$2 < 3$	logikai	igaz	
$(a - 2) \neq 8$	logikai	igaz	ha $a = 3$

- A kifejezések képzésének szigorú szabályai (szintaxis) vannak. Hibás (értelmezhetetlen) kifejezések:

kifejezés	hiba
$3/$	diadikus (bináris) operátor ($/$) egy operandussal
$piros < 2$	szín $<$ szám
$3 \cdot hideg$	szám \cdot hőmérséklet
$(2 < 3) + 5$	logikai $+$ szám



Programnyelvek

Programozási nyelv

Számítógéppel értelmezhető matematikai formalizmus

- Hasonlít a beszélt nyelvekhez, hogy könnyen érthető legyen, és egyszerűen tudjunk fogalmazni
- Szűk szókincs, szigorú nyelvtan (szintaxis)



Szintaktika és szemantika

- Szintaktikai (nyelvtani) hiba (syntax error)
 - helytelenül használjuk a programnyelv szabályait, a program értelmezhetetlen, végrehajthatatlan
 - A szintaktikai hiba hamar kiderül
 - általában egyszerűen, gyorsan javítható.
- Szemantikai (értelmezési) hiba (semantic error)
 - A program végrehajtható, lefut, de nem azt csinálja, amit specifikáltunk
 - A szemantikai hiba sokszor nehezen érhető tetten, nehezen reprodukálható, nehéz javítani.
 - A programtesztelés szakma.

3. fejezet

C nyelvi alapok



A C nyelv rövid története

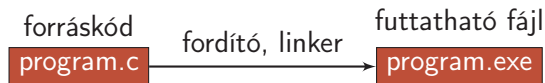
- 1972: Fejlesztés kezdete az AT&T Bell Labsban
UNIX kernel nagy része C-ben készült
- 1978: K&R C – Brian Kernigham, Dennis Ritchie:
The C Programming Language
- 1989: Szabványosítás: ANSI X3.159-1989

- 1999: C99-szabvány:
új adattípusok (komplex)
nemzetközi karakterkódolás
változó méretű tömbök
...
- 2007–: C1X szabvány, 2011-ben C11
C++ kompatibilitás
többszálú programok
...



A C főbb tulajdonságai

- fordított nyelv



- „kis nyelv”: kevés (10) utasítás, rengeteg (>50) operátor
- tömör szintaktika
 - nehezen olvasható (ha nem figyelünk oda)
 - könnyű hibát véteni
 - nehéz megtalálni
- jól optimalizálható, gyors kódot eredményez
- jól hordozható



Az első C program

■ A minimálprogram forráskódja

```
1  /* first.c -- Az első program */
2
3  int main()
4  {
5      return 0;
6  }
```

[link](#)

- A program elindul, majd befejezi a futást
- `/*` és `*/` között komment, a programozónak szól
- `int main()` – Így kezdődik minden C-program
 - `int Main()` – Nem így. A C „case sensitive”
- `{ }` – blokk, a programtörzset zárja közre
- `return 0;` – A program végét jelzi



Az első C program

- ...ami már csinál is valamit

```
1 /* Helloworld.c -- Az első program */
2 #include <stdio.h> /* printf-hez */
3
4 /* A főprogram */
5 int main()
6 {
7     printf("Szia, világ!\n"); /* Kiírás */
8     return 0;
9 }
```

[link](#)

- Fordítás és futtatás után a kimenet:

```
Szia, világ!
```

- `#include` – más C programrészek beillesztése
- `printf` – kiírás, `\n` – soremelés



Eggyel bonyolultabb

■ Egymás után kiadott utasítások

```
1 /* football.c -- szurkolóprogram */
2 #include <stdio.h>
3 int main()
4 {
5     printf("Szódásüveget"); /* nincs újsor */
6     printf(" a bírónak,\n"); /* itt van */
7     printf("hajrá, Fradi!");
8     return 0;
9 }
```

[link](#)

```
Szódásüveget a bírónak,  
hajrá, Fradi!
```



Változó értékének kiírása

```
1 #include <stdio.h>
2 int main()
3 {
4     int num;          /* num nevű egész változó dekl. */
5     num = 2;         /* num <- 2 értékadás */
6     printf("A szám értéke: %d\n", num); /* kiírás */
7     num = -5;       /* num <- -5 értékadás */
8     printf("A szám értéke: %d\n", num); /* kiírás */
9     return 0;
10 }
```

[link](#)

```
A szám értéke: 2
```

```
A szám értéke: -5
```

- `int num` – változódeklaráció.
`int` (integer, egész) a típus, `num` az azonosító
- `num = 2` – értékadás, `num` változó felveszi a „2” kifejezés értékét
- `printf(<formátum>, <mit>)` –
`<mit>` kifejezés értékének kiírása `<formátum>` formában



Blokk és deklaráció

Blokk szerkezete

```
{  
  <deklarációk>  
  <utasítások>  
}
```

```
1 {  
2   /* deklarációk */  
3   int num;  
4  
5   /* utasítások */  
6   num = 2;  
7   printf("%d\n", num);  
8 }
```




Blokk és deklaráció

Deklaráció szerkezete

```
<típusnév> <azonosító> [ = <kezdeti érték> ]opt;
```

```
1 int n; /* inicializálatlan */  
2 int number_of_dogs = 2; /* inicializált */
```

- n értéke kezdetben memóriaszemét
- number_of_dogs értéke kezdetben 2



Adat beolvasása

```
1  /* square.c -- szám négyzetreemelése */
2  #include <stdio.h>
3  int main()
4  {
5      int num;                /* egész változó dekl. */
6      printf("Adj meg egy egész számot: "); /* info */
7      scanf("%d", &num);      /* beolvasás */
8      /* két kifejezés értékének kiírása */
9      printf("%d négyzete: %d\n", num, num*num);
10     return 0;
11 }
```

[link](#)

```
Adj meg egy egész számot: 8
8 négyzete: 64
```

- `scanf(<formátum>, &<hova>)` –
 <formátum> formátumú adat beolvasása a <hova> változóba



Adat beolvasása

- Így is lehet, az eredmény ugyanaz.

```
1         #include <stdio.h>
2         int main(){int num; printf
3 ("Adj meg egy egész számot: ");scanf("%d",
4         &num);printf("%d négyzete: %d\n",
5         num,num*num);return
6         0;}
```

[link](#)

- Persze gondoljunk az utókorra is!

Köszönöm a figyelmet.