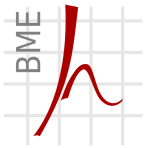
Függvények. Házi feladat.

A programozás alapjai I.



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

2018. szeptember 24.

Tartalom

1. fejezet

Függvények

Szegmentálás – motiváció

Írjunk programot, mely kiírja a 12-nél kisebb pozitív egész számok négyzetösszegét! $(1^2 + 2^2 + \cdots + 11^2)$

```
1  #include <stdio.h> /* printf-hez */
2
3  int main(void)
4  {
5    int i, sum; /* iterátor és a négyzetösszeg*/
6
7    sum = 0;                      /* inicializálás */
8    for (i = 1; i < 12; i = i+1) /* i = 1,2,...,11 */
9      sum = sum + i*i;            /* összegzés */
10
11   printf("A négyzetösszeg: %d\n", sum);
12   return 0;
13 }
```

Szegmentálás – motiváció

```c
int main(void) {
  int i, sum1, sum2, sum3;

  sum1 = 0;          /* 12-re */
  for (i = 1; i < 12; i = i+1)
    sum1 = sum1 + i*i;

  sum2 = 0;          /* 24-re */
  for (i = 1; i < 24; i = i+1)
    sum2 = sum2 + i*i;

  sum3 = 0;          /* 30-ra */
  for (i = 1; i < 30; i = i+1)
    sum3 = sum3 + i*i;

  printf("%d, %d, %d\n",
    sum1, sum2, sum3);
  return 0;
}
```

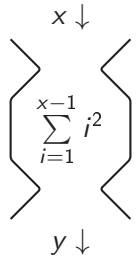
Írjunk programot, mely elvégzi az előbbi feladatot a 12, 24 és 30 számokra!

- Copy+Paste+javítgatás
- Sok hibalehetőség
- Hosszú program
- Nehezen karbantartható

Függvények

A függvény

- Önálló programszegmens
- Gyakran előforduló műveletsor elvégzésére
- Különböző paraméterekkel lefuttatható (hívható)
- Kiszámol valamit, és azt visszaadja a hívó programrésznek



Függvények – megoldás

```c
int squaresum(int n) /* függvénydefiníció */
{
  int i, sum = 0;
  for (i = 1; i < n; i = i+1)
    sum = sum + i*i;
  return sum;
}

int main(void) /* főprogram */
{
  int sum1, sum2, sum3;

  sum1 = squaresum(12); /* függvényhívás */
  sum2 = squaresum(24);
  sum3 = squaresum(30);

  printf("%d, %d, %d\n", sum1, sum2, sum3);
  return 0;
}
```


Függvény definíciója

Függvénydefiníció szintaxisa

```
<visszatérési érték típusa>
<függvény azonosító> (<formális paraméterek listája>)
<blokk>
```

```c
int squaresum(int n)
{
  int i, sum = 0;
  for (i = 1; i < n; i = i+1)
    sum = sum + i*i;
  return sum;
}
```

Függvény definíciója

A visszatérési érték típusa:

- A kiszámolt érték típusa

```
1 double average(int a, int b)
2 {
3   return 0.5 * (a + b);
4 }
```

- vagy `void` (üres), ha a függvény nem számol ki semmit

```
1 void print_point(double x, double y)
2 {
3   printf("(%.3f, %.3f)", x, y); /* (2.000, 4.123) */
4 }
```

- sokszor nem a kiszámolt érték, hanem a mellékhatás a fontos

Kitérő: Főhatás és mellékhatás

Főhatás a függvény kiszámolja és visszaadja a visszatérési értéket

Mellékhatás a függvény „csinál még valamit” (képernyőre, fájlba ír, lejátssza az MP3-at, kilövi a rakétát...)

- Bizonyos programnyelvek határozott különbséget tesznek különböző programszegmensek között:
  - **függvény** a főhatás a lényeg
  - **eljárás** nincs főhatás, a mellékhatás a fontos
- C-ben csak függvény létezik, az eljárást az üres (`void`) visszatérési típusú függvények testesítik meg.
- Általában törekedjünk a fő- és mellékhatás szétválasztására!

Függvény definíciója

Formális paraméterlista

- Paraméterek deklarációja külön-külön, vesszővel elválasztva, hogy a függvényben adott néven hivatkozhassunk rájuk

```
1 double volume(double x, double y, double z)
2 {
3   return x*y*z;
4 }
```

- Számuk lehet 0, 1, 2, ... tetszőlegesen sok (127 🙂)
- 0 számú paramétert `void`-dal jelölünk

```
1 double read_next_positive(void)
2 {
3   double input;
4   do scanf("%lf", &input) while (input <= 0);
5   return input;
6 }
```


Függvény definíciója

A `return` utasítás

- megadja a visszatérési értéket, megszakítja a függvényblokk végrehajtását, és visszatér a hívóhoz
- több is lehet belőle, de az első végrehajtásakor visszatér

```
1 double distance(double a, double b)
2 {
3   double dist = b - a;
4   if (dist < 0)
5     return -dist;
6   return dist;
7 }
```

- `void` típusú függvényben is lehet `return;`

Függvényhívás

```c
double distance(double a, double b)
{
  ...
}
```

Függvényhívás szintaxisa

<függvény azonosító> (<aktuális paraméterek kif>)

```c
double x = distance(2.0, 3.0); /* x 1.0 lesz */
```

```c
double a = 1.0;
double x = distance(2.5-1.0, a); /* x 0.5 lesz */
```

```c
double pos = read_next_positive(); /* üres () */
```


A főprogram mint függvény

```c
int main(void) /* már értjük, hogy ez mi */
{
  ...
  return 0;
}
```

A főprogram is függvény

- Az operációs rendszer hívja meg a program indításakor
- Nem kap paramétert (ezt később még módosítjuk)
- Egész (`int`) értéket ad vissza
  - Hagyományosan helyes lefutás esetén 0-t, egyébként hibakódot

```
Process returned 0 (0x0)   execution time:  0.001 s
press ENTER to continue.
```


A függvényhívás mechanizmusa

```c
/* Téglalap területe */
int area(int x, int y)
{
  int S;
  S = x * y;
  return S;
}

/* Főprogram */
int main(void)
{
  int a, b, T;
  a = 2;              /* alap */
  b = 3;          /* magasság */
  T = area(a, b); /* Terület */
  return 0;
}
```

regiszter: 0

A függvényhívás mechanizmusa

Érték szerinti paraméterátadás

- A függvények az aktuális paraméterek kifejezéseinek értékeit kapják meg paraméterként
- A paramétereket változóként használhatják, melyek a hívás helyén kapott kezdeti értékkel rendelkeznek.
- A függvények módosíthatják paramétereik értékét, ennek semmilyen hatása nincs a hívó programrészre.

Változók láthatósága és élettartama

Lokális változók

1. A függvény paraméterei és
2. a függvényben deklarált változók

- A függvénybe való belépéskor jönnek létre, megszűnnek visszatéréskor.
- Külső programrész nem látja őket. (még a hívó sem)

Globális változók – ha lehet, kerüljük

A függvényeken (`main()`-en is) kívül deklarált változók

- A program futása alatt végig léteznek
- Mindenki írhatja-olvashatja őket!
- Névütközés esetén a lokális változó elfedi a globálisat

Rejtvény

Mit ír ki az alábbi program?

```c
#include <stdio.h>

int a, b;

void func(int a)
{
  a = 2;
  b = 3;
}

int main(void)
{
  a = 1;
  func(a);
  printf("a: %d, b: %d\n", a, b);
  return 0;
}
```


Összetett feladat

Írjunk C programot, mely a felhasználótól bekér két egész számot (`low` < `high`), majd kilistázza a két szám közé eső prímeket.

- A megoldás pszeudokódja szegmensekre bontva:

főprogram

```
BE: low, high
MINDEN i-re low-tól high-ig
  HA prímteszt(i) IGAZ
    KI: i
```

prímteszt(p)

```
MINDEN i-re 2-től p gyökéig
  HA i osztja p-t
    return HAMIS
return IGAZ
```

- Figyeljük meg a két *i* és *p* szerepét

Összetett feladat – megoldás

```c
#include <stdio.h> /* scanf, printf */

int low, high; /* globális változók */

void read(void) /* beolvasó függvény */
{
  printf("Kérek egy kisebb és egy nagyobb számot!\n");
  scanf("%d%d", &low, &high);
}

int isprime(int p) /* prímtesztelő fv. */
{
  int i;
  for (i=2; i*i<=p; i=i+1)    /* i 2-től p gyökéig */
    if (p%i == 0)  /* ha p osztható i-vel, nem prím */
      return 0;
  return 1; /* ha ide eljutottunk, prím */
}
```

Összetett feladat – megoldás

```
19
20 int main()
21 {
22   int i;
23
24   read(); /* függvénnyel beolvassuk a határokat */
25
26   printf("Prímek %d és %d között:\n", low, high);
27   for (i=low; i<=high; i=i+1)
28   {
29     if (isprime(i)) /* függvénnyel tesztelünk */
30       printf("%d\n", i);
31   }
32
33   return 0;
34 }
```


Tervezési alapelv

- A függvények a program többi részével paramétereiken és visszatérési értékükön keresztül tartják a kapcsolatot.
- Hacsak nem kimondottan ez a feladatuk,
  - nem írnak képernyőre
  - nem olvasnak billentyűzetről
  - nem használnak globális változókat

2. fejezet

Nagy házifeladat

Téma kiválasztás

Milyen témát válasszunk?

- Ami közel áll hozzánk, aminek értjük a rendszerét...
- Amiben van olyan érdekes jellemző, amit csak többféle adat összekapcsolásával tudunk kiszámítani
- Például:
  - Sportversenyek
  - Hobbihoz kapcsolódó nyilvántartás
  - Katalógusok

Milyet ne?

- Túlságosan összetett, bonyolult rendszer
- Ahol nem a programozáson van a hangsúly
- Ami szerepel a tárgy honlapján példaként
- Amit már más is megcsinált...

A mi példánk témája a *Formula-1*

Feladat alapok

Tisztázzuk, hogy

- Milyen adatokat veszünk figyelembe?
  A futamokon mért köridők, kiállási idők, egyéb versenyzői adatok
- **Mit** fog kiszámítani a program?

Ami még nagyon ráér:

- Pontosan milyen típusú adatként fogom tárolni az egyes információkat
- Hogyan fogja számolni a program a megoldást
- Hogy fogják hívni a forrásfájl(oka)t
- Milyen adatokkal fogom kipróbálni, hogy jól működik-e

Bemenet - kimenet

Az adatok további szűkítése:

- Miből mennyi adatot szeretnénk nyilvántartani?

Egy futam vagy több futam eredményei?	egy
Versenyzők száma?	több
Csapatokat kezeljük?	nem

- Egyszerűbb és bonyolultabb kérdések, amiket a program meg tud válaszolni:
  - *Ki nyerte a futamot?*
  - *Ki futotta a leggyorsabb kört?*
  - *Mennyi volt a körideje annak a versenyzőnek, aki a leglassabb átlagú körben a legjobb köridőt futotta?*

A fő kérdés

Akkor végül mit számítson ki a program?

- Elég egy (1) **bonyolultabb** kérdést megcélozni a fentiek közül
- Mindenképpen olyan kell, amiben szükség van az adatok és kapcsolataik tárolására

Adatok viszonya

Az adatok körének további pontosításához szükséges

- A feladat szempontjából lényeges **dolgok (résztvevők, objektumok)** azonosítása
  1. *Futam*
  2. *Versenyző* – egy futamhoz több versenyző tartozik
  3. *Időeredmény (köridő vagy kiállási idő)* – minden versenyzőhöz sok időeredmény tartozik

Tulajdonságok

A **dolgokhoz (résztvevőkhöz, objektumokhoz)** válasszuk ki a lényeges a tulajdonságokat és határozzuk meg a jellegüket is

- Azok mindenképpen kellenek, amiket a feladatban fel fogunk használni
- További tulajdonságok is szerepelhetnek, amik a témakörhöz kapcsolódnak

Futam		**Versenyző**		**Időeredmény**	
Helyszín	szöveg	Rajtszám	egész szám	Időtartam	valós szám
Körök száma	egész szám	Név	szöveg	Típus	köridő/boksz
Pálya hossza	egész szám	Rajtpozíció	egész szám		

Fájlok tartalma

- A kiírás szerint (legalább) két bemeneti fájl kell

Verseny.txt

1. sor Verseny éve: egész szám<TAB>Verseny helyszín: szöveg
2. sor Körök száma: egész szám
3. sor Pálya hossza méterben: egész szám
4. sor Üres sor
4. sortól Rajtpozíció<TAB>Rajtszám<TAB>Versenyző neve: 3 betű

Ido.txt

1. sortól Rajtszám<TAB>Idő másodpercben<TAB>Időtípus: K(ör) vagy B(ox) betű

Specifikáció készítése

- A hangsúly azon van, hogy
  1. milyen adatok alapján,
  2. mit fog csinálni a program.
- Mit veszünk figyelembe a világból és mit nem?
- Feltételezések az adatokról

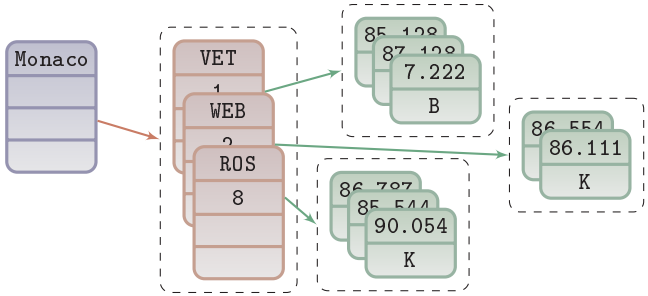
  Például: *Az időeredmények időrendben kerülnek be az adatsorba*
- A különleges eseteket hogyan kezeli a program?

  Legalább néhány fontosabbat gondoljunk át
  Például: *Ha egyetlen versenyző sincs a rajtlistán, akkor a program válasza: SENKI*
- A feladat pontosítása olyan szintű kell legyen, hogy az alapján el lehessen kezdeni gondolkodni a megoldáson
- A bemeneti adatok pontosabb leírására jó módszer lehet megadni, hogy a bemeneti fájlok sorai mit tartalmaznak.

Adatszerkezet

Egyelőre ne gondolkodjunk programozási problémában, mert még nem tanultuk meg az összes eszközt, helyette:

- Hogyan oldanánk meg űrlapok segítségével?
  - Milyen adatok kerülnének egy lapra?
  - Melyik másik lapokat „csatolnánk” egy laphoz?



Felépítési és kiszámítási algoritmusok

Továbbra is az űrlapos sémában gondolkodva próbáljuk kitalálni, hogy:

Hogyan tölthetjük ki az űrlapjainkat a fájlok alapján, vagyis építhetjük fel az adatszerkezetet?

- Mikor kell új lapot betenni, és hová kell csatolni?
- Melyik lapokon kell módosítani?

Hogyan számíthatjuk ki az eredményt?

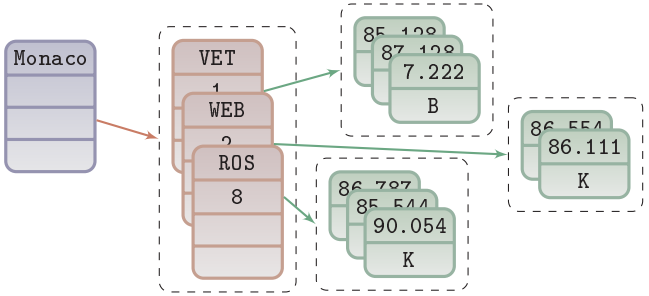
- Milyen rendszerben kell végignézni az űrlapokat?
- Milyen adatokat kell róluk gyűjteni közben?

Felépítés példa


```
Monaco
78
3340

1 1 VET
2 2 WEB
3 8 ROS
4 4 HAM
5 1 GRO
6 5 ALO
7 6 MAS
8 9 RAI
9 18  MAL
```

```
1 85.128  K
2 86.554  K
8 86.787  K
4 87.888  K
...
18  8.665 B
8 85.544  K
1 87.128  K
2 86.111  K
4 86.808  K
...
18  97.565  K
1 7.222 B
8 90.054  K
```



És ami még hátra van

- Az eddigiek leírása és időben való leadása
- Leprogramozás (Megvalósítás, Implementálás)
  - Folyamatosan ismerjük meg a szükséges eszközöket
  - A 9. előadás után már minden ehhez szükséges ismeret meglesz
- Tesztelés és hibajavítás, ha nem az jön ki, mint amit várunk
- Dokumentáció befejezése és időben való leadása

3. fejezet

Gyakorló feladatok

1. Gyakorló feladat

Írjon C programot, mely egy egész számot (R) olvas be a standard bemenetről, majd a standard kimeneten megjelenít egy 10×10 mező méretű karakterábrát.

- Az ábra mezőit balról jobbra (x) és fentről lefelé (y), 1-től kezdve egyesével számozzuk.
- Azon mezőkbe, melyekre $x^2 + y^2 < R^2$, a program a '#' karaktert írja, a többi mezőt a '.' karakterrel jelölje.
- $R = 8$-ra pl. az alábbi ábra jelenik meg:

```
#######...
#######...
#######...
######....
######....
#####.....
###.......
..........
..........
..........
```

Megoldás

2. Gyakorló feladat

Írjon C programot, mely a standard bemenetére érkező egész számokat dolgozza fel.

- A program feladata, hogy képezze az összes bejövő szám abszolút értékét, majd kiírja a standard kimenetre a legkisebb és a legnagyobb érték különbségét.
- A számsor végét a 0 szám jelzi, melyet már nem kell feldolgoznia.
- Feltételezheti, hogy legalább egy feldolgozandó szám érkezik.

Megoldás

3. Gyakorló feladat

Írjon C programot, mely egy legfeljebb 100 valós számot tartalmazó végjeles sorozatot olvas be a standard bemenetről.

- A program feladata, hogy a standard kimeneten megadja, hogy hány olyan érték érkezett, mely nagyobb, mint a sorban tízzel korábban érkező érték.
- A sorozat végjele a 0.0 érték.

Megoldás