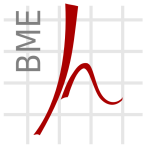
Függvények. A felsorolt típus.

A programozás alapjai I.



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

2020. október 5.

Tartalom

1. fejezet

Függvények

Szegmentálás – motiváció

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

```c
#include <stdio.h> /* printf-hez */

int main(void)
{
  int i, sum; /* iterátor és a négyzetösszeg*/

  sum = 0;                      /* inicializálás */
  for (i = 1; i < 12; i = i+1) /* i = 1,2,...,11 */
    sum = sum + i*i;            /* összegzés */

  printf("A négyzetösszeg: %d\n", sum);
  return 0;
}
```

Szegmentálás – motiváció

```
1  int main(void) {
2    int i, sum1, sum2, sum3;
3
4    sum1 = 0;          /* 12-re */
5    for (i = 1; i < 12; i = i+1)
6      sum1 = sum1 + i*i;
7
8    sum2 = 0;          /* 24-re */
9    for (i = 1; i < 24; i = i+1)
10     sum2 = sum2 + i*i;
11
12   sum3 = 0;          /* 30-ra */
13   for (i = 1; i < 30; i = i+1)
14     sum3 = sum3 + i*i;
15
16   printf("%d, %d, %d\n",
17     sum1, sum2, sum3);
18   return 0;
19 }
```

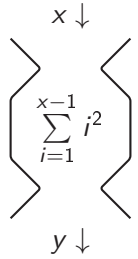
Írjunk programot, mely elvégzi az előbbi feladatot a 12, 24 és 30 számokra!

- 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

```
1  int squaresum(int n) /* függvénydefiníció */
2  {
3    int i, sum = 0;
4    for (i = 1; i < n; i = i+1)
5      sum = sum + i*i;
6    return sum;
7  }
8
9  int main(void) /* főprogram */
10 {
11   int sum1, sum2, sum3;
12
13   sum1 = squaresum(12); /* függvényhívás */
14   sum2 = squaresum(24);
15   sum3 = squaresum(30);
16
17   printf("%d, %d, %d\n", sum1, sum2, sum3);
18   return 0;
19 }
```


Függvény definíciója

Függvénydefiníció szintaxisa

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

```
1  int squaresum(int n)
2  {
3    int i, sum = 0;
4    for (i = 1; i < n; i = i+1)
5      sum = sum + i*i;
6    return sum;
7  }
```

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

A felsorolt típus

A felsorolt típus – Motiváció

- Mászkálós játékprogramot írunk, melyben a felhasználó a játékos mozgását négy billentyűvel vezérli.

- A felhasználói input beolvasására sokszor szükség van, ezért ezt a műveletet célszerűen egy `read_direction()` függvényre bízzuk
- A függvény a billentyűzetről olvas, majd visszaadja a haladási irányt a hívó programrésznek.
- Milyen típust adjon vissza a függvény?

A felsorolt típus – Motiváció

- 1. javaslat: Adja vissza a leütött karaktert (`'a'`,`'s'`,`'w'`,`'d'`):

```
1  char read_direction(void)
2  {
3    char ch;
4    scanf("%c", &ch);
5    return ch;
6  }
```

- Problémák:
  - A program többi részén (sok helyen) kell dekódolnunk a karakterekből az irányokat.
  - Ha a programot átírjuk ← ↓ ↑ → vezérlésre, ezer helyen kell módosítanunk.
- Megoldás:
  - Helyben kell dekódolnunk, és csak az irányt kell visszaadnunk.
  - De azt milyen formában?

A felsorolt típus – Motiváció

- 2. javaslat: Adjon vissza 0,1,2,3 `int` értékeket:

`'a'`	0	←
`'w'`	1	↑
`'d'`	2	→
`'s'`	3	↓

```
1  int read_direction(void) {
2    char ch;
3    scanf("%c", &ch);
4    switch (ch) {
5    case 'a': return 0; /* bal */
6    case 'w': return 1; /* fel */
7    case 'd': return 2; /* jobb */
8    case 's': return 3; /* le */
9    }
10   return 0; /* bal default :) */
11 }
```

- Probléma:
  - A program többi részén a 0-3 számokat kell használnunk az irányokra, a programozónak emlékeznie kell a szám-irány összerendelésre.

A felsorolt típus – Megoldás

- Egy `direction` nevű típusra van szükségünk, amely a `LEFT`, `RIGHT`, `UP`, `DOWN` értékeket tudja tárolni.
- C-ben csinálhatunk ilyet!

  A megfelelő felsorolt típus (enumerated type, `enum`) deklarációja:

```
1  enum direction {LEFT, RIGHT, UP, DOWN};
```

- A típus használata

```
1  enum direction d;
2  d = LEFT;
```


A felsorolt típus – Megoldás

- A végleges megoldás az új típussal

```
1  enum direction {LEFT, RIGHT, UP, DOWN};
2
3  enum direction read_direction(void)
4  {
5    char ch;
6    scanf("%c", &ch);
7    switch (ch)
8    {
9    case 'a': return LEFT;
10   case 'w': return UP;
11   case 'd': return RIGHT;
12   case 's': return DOWN;
13   }
14   return LEFT;
15 }
```

A felsorolt típus – Megoldás

- És a függvény használata:

```
1 if (d == RIGHT)
2   printf("Megevett egy tigris\n");
```

- Ugyanez a felsorolt típus nélkül ilyen lenne:

```
1 int d = read_direction();
2 if (d == 2) /* "magic" konstans, mit is jelent? */
3   printf("Megevett egy tigris\n");
```

- A felsorolt típus...
  - beszédes kóddal helyettesíti a „magic konstansokat”,
  - a tartalomra koncentrál az ábrázolás helyett,
  - magasabb szintű programozást tesz lehetővé.

A felsorolt típus – Definíció

A felsorolt (enum) típus

Szimbolikus néven hivatkozott egész típusú állandók összefogása egy típussá.

```
enum [<felsorolás címke>]opt
{ <felsorolás lista> }
[<változó azonosítók>]opt;
```

```
1 enum direction {LEFT, RIGHT, UP, DOWN} dir1, dir2;
```


enum példák

```
1  enum month {
2    JAN,   /*  0 */
3    FEB,   /*  1 */
4    MAR,   /*  2 */
5    APR,   /*  3 */
6    MAY,   /*  4 */
7    JUNE,  /*  5 */
8    JULY,  /*  6 */
9    AUG,   /*  7 */
10   SEPT,  /*  8 */
11   OCT,   /*  9 */
12   NOV,   /* 10 */
13   DEC    /* 11 */
14 };
15
16 enum month m=OCT; /*9*/
```

```
1  enum {
2    RED,       /*  0 */
3    BLUE = 3,  /*  3 */
4    GREEN,     /*  4 */
5    YELLOW,    /*  5 */
6    GRAY = 10  /* 10 */
7  } c;
8
9  c = GREEN;
10 printf("c: %d\n", c);
```

```
c: 4
```