Pázmány Péter Katolikus Egyetem
Információs Technológiai Kar

Bevezetés a programozásba I

5. gyakorlat

C++ alapismeretek

© 2011.10.11. Giachetta Roberto
groberto@inf.elte.hu
http://people.inf.elte.hu/groberto

C++ alapismeretek

Történet

- Wikipédia: a C++ általános célú, magas szintű programozási nyelv, mely támogatja az imperatív, az objektum-orientált, valamint a sablonprogramozást
- Első változata 1979-ben készült (Bjarne Stroustroup) a C programozási nyelvből, eredetileg *C with Objects* névvel
  - célja: objektumorientált programozási lehetőségekkel való kiegészítése a nyelvnek
- Jelenleg a 2003-as szabványt használjuk, a fordítók is ennek megfelelően működnek
- Többek szerint közepes szintű nyelvnek tekinthető, mert alacsonyabb szinten használatos utasítások (bit szintű műveletek, direkt memóriaműveletek,…) is használhatóak

C++ alapismeretek

Történet

- Jelenleg is fejlesztés alatt áll: *C++0x*
- Az alap nyelv csak konzol-alkalmazások készítésére szolgál
  - sok kiegészítő található hozzá, amelyek segítségével sok megoldást implementálni lehet (pl. grafikus környezet)
- A világ egyik leggyakrabban használt programozási nyelve:
  - a szabvány teljesen szabad, ingyenes
  - önmagában minden lehetséges nyelvi eszközt megad, amelyre a programozás során szükségünk lehet
  - gyors, hatékony programokat készíthetünk benne
  - sok fordító, fejlesztőkörnyezet készült hozzá
  - több nyelv alapjául szolgált: JAVA, C#, PHP, …

Programok felépítése

- A C++-ban ugyanúgy vannak:
  - változók, konstansok
  - utasítások, vezérlési szerkezetek
  - konzol képernyő és fájlkezelés
- És még sok minden más:
  - programblokkok, függvények
  - objektumok, osztályok, sablonok
  - memóriakezelés, adatszerkezetek, …
- A lista kiegészítésekkel tetszőlegesen bővíthető (pl. adatbázis kezelés, grafikus felület, 3D-s megjelenítés)
- Mi egyelőre csak a nyelv beépített lehetőségeit fogjuk használni, később grafikus felülettel is kiegészítjük

A „Hello World” program

Feladat: Írjuk ki a Hello World! feliratot a képernyőre.

Megoldás:

```
// fejrész:
#include <iostream>  // további fájlok beolvasása
using namespace std; // használni kívánt névtér

// törzsrész:
int main() // főprogram deklaráció
{
   cout << "Hello, World!" << endl; // kiírás
   return 0; // érték visszaadás
}   // főprogram vége
```

Fejrész

- A program elején van a *fejrész*, ami tartalmazza:
  - a program működéséhez szükséges további fájlok neveit
  - a programban használt névtereket
  - a programban előre definiált elnevezések (makrók) értékeit
    - ezek olyan elnevezés - érték párok, amelyeknél a fordítás előtt behelyettesíti a programban az első érték összes előfordulását a második értékkel
    - szerkezete: **`#define <elnevezés> <érték>`**
    - pl.: **`#define ALMA 100`**

      `// az ALMA helyére mindenhova 100-t ír`
    - használata általában kiváltható konstans változókkal, ezért ritkán kerül elő

C++ alapismeretek

Fejrész

- A C++ utasításai és típusai több fájlban helyezkednek el, az írandó programok általában önmagukban nem működőképesek, használnunk kell a további fájlokban megírt utasításokat
- Mi is elhelyezhetjük a programkódunkat több fájlban (majd később lesz), amiket szintén használni szeretnénk
- Ehhez meg kell adnunk, hogy mely fájlokat szeretnénk használni a programban
  - **`#include <`*`fájlnév`*`>`** - a beépített fájlok között keres, amik a C++ környezetben megvannak
  - **`#include "`*`fájlnév`*`"`** – aktuális könyvtárban keres
- Ekkor a program fordításakor a berakott fájlok teljes tartalma átmásolódik a mi programunkba

C++ alapismeretek

Névterek

- Amikor további beépített fájlokat használunk a programban, az ott található utasítások csoportosítva vannak úgynevezett *névterek*be, hogy megkülönböztessük a különböző funkciókat ellátó utasításokat
- A programban meg kell mondanunk, hogy melyik névtérből származnak a használni kívánt utasítások, ennek két módja
  - a fejrészben kiadjuk a **`using namespace <név>;`**, utasítást, ekkor a program hátralevő részében a megadott névteret használjuk (általában ez a célravezetőbb)
  - a használandó utasítást a következő a **`<névtérnév>::<parancsnév>`** formában írjuk le, ekkor megmondjuk, hogy a parancs a megadott névtérből való, pl.: **`std::cout`**

C++ alapismeretek

Törzsrész

- A programok törzsrésze *függvények*ből áll
  - a függvény a program egy saját működéssel rendelkező része, amely értéket ad vissza az őt meghívó utasításnak
  - a visszaadott érték a *visszatérési érték*, amelyet a **`return`** utasítással adunk meg, a függvény működése mindig leáll ennél a pontnál, amit utána írunk, az nem kerül végrehajtásra
  - az általunk írt programban a programhívó eljárás a **`main`** függvényt hívja meg (az pedig további függvényeket hívhat meg, ez majd lesz később…)
  - A **`main`** függvény egész típusú (visszatérési értékű), ezért a függvény vége előtt vissza kell adnunk egy egész számot

Törzsrész

- A **`main`** függvény szerkezete:

```
int main()
{
    …              // utasítások
    return 0;      // visszaadunk egy 0-s értéket
}
```

- A **`main`** függvényben az úgynevezett *hibakód*ot szokás visszaadni, azaz, hogy hiba történt-e a program futása során
  - 0: nem történt hiba
  - 1,2,…: valamilyen hiba történt
- A hibakód jelentése nincs előre szabályozva, a programozó dönti el, hogy a visszaadott hibakód milyen hibát kíván reprezentálni (ezt például dokumentációba lehet leírni)

- Lehetőségünk van arra, hogy a függvényben különböző értékeket adjunk vissza, elágazások segítségével:

```
int main() {
    …
    if (…) {
       // elágazás, ha történik valami hiba
       …
       return 1; // adjunk vissza 1-es kódot
    } else {     // különben
       …
       return 0; // adjunk vissza 0-s kódot
    }
}
```

- Szekvencia:
  - az utasítások végére `;`-t kell tenni
  - egy sorban lehet több utasítást is írni, illetve egy utasítást lehet több sorban is írni
- Programblokk:
  - utasítások csoportosításai
  - egy programblokk: **`{ <utasítások> }`**
  - programblokkok tartalmazhatnak további blokkokat, így egymásba ágyazott szerkezetet készíthetünk, pl.:

```
{ <ut.> { <ut.> }
  { <ut.> } <ut.>
}
```

- Elágazás:
  - feltételtől függően különböző utasítások végrehajtása:

```
if <feltétel> // logikai kifejezés
   { <igaz ág utasításai> }
else
   { <hamis ág utasításai> }
```

  - a hamis ág itt is elhanyagolható, ekkor a szerkezet:

```
if <feltétel> { <igaz ág utasításai> }
```

  - ha csak egy utasítást akarunk írni, a blokkot elhagyhatjuk:

```
if <feltétel>
   <igaz ág utasítása>
else
   <hamis ág utasítása>
```

- Ciklus:
  - utasítások ismétlése a megadott feltétel függvényében, a feltétel logikai értékű kifejezés lehet
  - a ciklusmag egy, vagy több utasításból állhat, ha csak egy utasítást írunk, nem kell blokkot használni
  - előtesztelő:

```
while <feltétel>
    { <utasítások> }
```

  - utántesztelő:

```
do
    { <utasítások> }
while <feltétel>;
```

- a számláló szerkezete itt teljesen elkülönül:

```
for (<számláló kezdőérték>;
      <számláló feltétele>;
      <számláló inkrementálás>){
   <utasítások>
}
```

- de természetesen előtesztelő ciklussal is meg lehet fogalmazni a számlálót, az eredmény ugyanaz lesz:

```
<számláló kezdőérték>;
while (<számláló feltétele>){
     <utasítások>
     <számláló inkrementálás>
}
```

Megjegyzések

- Megjegyzést bárhol elhelyezhetünk a kódban
  - elhelyezhetjük a sor végén, akkor az utána a sorba írt utasításokat nem veszi figyelembe:
    ```
    <utasítás>; // megjegyzés
    ```
  - elhelyezhetjük a sor közben, illetve bármely utasítás közben (ügyelve arra, hogy nem szó közben írjuk), ekkor a komment után lévő utasításokat figyelembe veszi
    ```
    <utasítás>; /* megjegyzés */ <utasítás>;
    /*
    megjegyzés
    */
    <utasítás eleje> /* megjegyzés */
    <utasítás vége>;
    ```

- Értékadás (=)
  - jelentése: a balérték egyezzen meg a jobbértékkel
  - a bal oldalán változó állhat, a jobb oldalán olyan kifejezés, amelynek eredménye ugyanolyan, vagy típusú, mint a változó, vagy kompatibilis
  - pl.: `b = 3 // a b értéke legyen 3`
- Matematikai műveletek
  - eredményünk a bemenettel megegyező típusú
  - összeadás (+), kivonás (–), szorzás (*), osztás (/), maradékvétel (%)
  - pl.: `a % 3 // a modulo 3`

- Értékmódosítások
  - eggyel növeli, vagy csökkenti az egész szám értékét
  - inkrementálás (**++**), dekrementálás (**--**)
  - pl.: `a++ // növeljük meg a értékét 1-gyel`
  - két módon lehet megadni, a változó előtt, illetve mögött, a különbség a végrehajtási sorrendben van
  - azaz ha előtte értékadás található, akkor történhet az érték átadása a módosítás előtt, illetve után is
  - pl.:

```
b = a++; // a értékét átadjuk b-nek, majd
         // a-t növeljük
b = ++a; // a értékét növeljük, majd
         // átadjuk b-nek
```

- Összehasonlító műveletek
  - eredményük logikai típusú lesz
  - a balértéket hasonlítja a jobbértékkel
  - egyenlőségvizsgálat (==), különbségvizsgálat (!=), kisebb (<), nagyobb (>), kisebb, vagy egyenlő (<=), nagyobb, vagy egyenlő (>=)
  - pl.: `b == 3    // a b értéke egyenlő-e 3-mal?`
- Logikai műveltek
  - logikai értékeken végeznek logikai eredményű műveletet
  - tagadás (`!`), és (`&&`), vagy (`||`)
  - pl.: `!(a && b) // nem (a és b)`

Operátorok

- Speciális értékadások, amelyekkel több utasítást egyszerre adhatunk ki:
  - add hozzá (**+=**), pl.: **`a = a+2`** helyett **`a+=2`**
  - vond ki belőle (**-=**)
  - vedd modulo (**%=**), pl.: **`a = a%2`** helyett **`a %= 2`**
  - feltételesen tedd egyenlővé (**>>=**, **<<=**), pl.: **`a >>=2`** jelentése: ha a nagyobb kettőnél, akkor legyen az értéke 2
- Tömbelem megcímzése (**`[]`**)
  - tömb (vektor), illetve szöveg bármely elemét lekérdezhetjük, módosíthatjuk
  - az indexelés 0-tól kezdődik
  - pl.: **`a[3]`** **`// az a tömb 3-as indexű eleme`**

- Bitenkénti műveletek
  - az értékeink tényleges eltárolt bináris változatát is tudjuk manipulálni
  - bitenkénti eltolás balra (**<<**), bitenkénti eltolás jobbra (**>>**), komplemens képzés (**~**), bitenkénti és (**&**), bitenkénti vagy (**|**), kizáró vagy (**^**)
  - pl.: `a ^ b  // a XOR b`

- A PLanG-gal ellentétben változókat bárhol deklarálhatunk a kódunkban, nincs külön deklarációs rész
  - ha minden programblokkon, függvényen kívül deklaráljuk őket, akkor a deklarálás pontjától a program végéig bárhol (bármilyen függvényben, illetve programblokkban) elérhetőek lesznek, ezek az úgynevezett *globális változók*
  - ha valamely programblokkban, függvényben deklaráljuk őket, akkor csak annak végéig lesznek elérhetőek, utána megsemmisülnek, és nem lehet hivatkozni rájuk, ezeket *lokális változók*nak nevezzük
  - globális változók használatát lehetőleg kerüljük el, mert könnyen vezetnek hibás működéshez

Konstansok

- A C++ erősen típusos nyelv, azaz minden változónak létrehozásakor meg kell adnunk a típusát, és azt a fordító nyomon fogja követni a program lefordításakor
  - változó deklaráció:
    <típus> <változónév>;
  - lehetőségünk van kezdőértéket is adni a változónak:
    <típus> <változónév> = <kezdőérték>;
- Egyszerre több ugyanolyan típusú változót deklarálhatunk, vesszővel elválasztva a neveket
- A konstansok olyan változók, amelyeknek a kezdőértékét nem módosíthatjuk a program során, a **`const`** kulcsszóval megjelöljük őket, és adunk nekik kezdőértéket:
  `const` ***<típus> <változónév> = <kezdőérték>;***

- Logikai típus:
  - kulcsszava: **`bool`**
  - felvehető értékek: **`true`**, **`false`**
  - meg lehet adni neki egész számokat is, ekkor a 0 értéke a hamis, minden más igaz, ha kiíratunk egy logikai változót, akkor 0-t, vagy 1-t ír ki
  - logikai, illetve bitenkénti műveletek végezhetőek rajta, a bitenkénti művelet megfeleltethető a logikai műveletnek (pl. tagadás és komplementer)
  - pl.:
```
bool a, b, c;
    a = true; b = false;
    c = a || !(~a && b) && true;
```

- Egész típusok:
  - attól függően, mekkora méreten szeretnénk eltárolni az értéket, különböző kulcsszavakat használhatunk:
    - **`short`** (16 bit): –32768 - +32767
    - **`int`** (32 bit): –2147483648 - +2147483647
    - **`long`** (32 bit): –2147483648 - +2147483647
  - ezek az általános méretek, de a szabvány nem adja meg pontosan, ezért fordítóként különbözhet a méret
  - kompatibilisek egymással, illetve a logikai és valós típussal
  - egészek osztása egész eredményt ad
  - pl.:
```
int a = 2, b = 4;
    a = (a * (b + 6) – 4.3) % a;
```

- Valós, lebegőpontos típusok:
  - kulcsszavai: **`float`**, **`double`**, **`long double`**
  - a méretek implementációfüggőek (balról jobbra növekvően), a típusok kompatibilisek egymással, az egész típusokkal, illetve a logikai típussal
  - a maradékvétel kivételével mindent tud, amit az egész
- Karakter típus:
  - kulcsszava: **`char`**
  - a karaktereket szimpla idézőjelben kell megadnunk
  - mivel ténylegesen a karakterek ASCII kódját tárolja, használhatóak a matematikai műveletek is rajta
  - pl.: **`char a, b ='y'; (a++ == 'b')`**

- Szöveg (string) típus:
  - kulcsszava: **`string`**
  - dupla idézőjelben kell megadnunk a konstansokat
  - nem beépített típus, ezért használatakor hivatkoznunk kell az őt tartalmazó fájlra (**`#include <string>`**)
  - igazából karakterek tömbjeként működik, azért használható az indexelő (**`[]`**) operátor
  - lekérdezhető a hossza: ***`<változónév>`*`.size();`**
  - szöveget lehet konkatenálni az összeadás (**`+`**) operátorral
  - pl.:
```
string a = "a", b = "szöveg";
a = a + " " + b;  // konkatenáció
int x = a.size(); // x = 8 lesz
```

- Tömbök:
  - minden típusból készíthetünk tömböt olyan módon, hogy megadjuk a méretét a változó létrehozásakor a `[ ]` operátor segítségével
  - méretként csak egész érték adható meg (egész típusú változó is, de ezt ne használjuk)
  - a tömb csak az adott típusból tartalmazhat értékeket
  - pl.:
```
int a[10]; // 10 elemű egészekből álló
           // tömb létrehozása
```
  - elem lekérdezése és beállítása ugyanezzel az operátorral, ahol az indexek 0-tól a tömb mérete -1-ig tartanak, ha túlindexelünk, az futási idejű hibához vezet
  - lehet többdimenziós tömböket (mátrixokat) is készteni

Konzol használat

- A C++ adatbeolvasásra és megjelenítésre konzol felületet használ (persze lehet grafikus környezetet is írni hozzá), ezért az általunk írt programok futhatnak Dos, Windows és Linux alatt is, csak mindig az adott rendszeren kell lefordítanunk őket
- A konzolon csak egyféle színben, egyféle betűtípussal tudunk kiíratni karaktereket
- A konzolra történő beolvasás-kiírásnál a műveleteket külön fájlban, az *iostream* (*input-output stream*)-ben találjuk, ezért ezt kell használnunk a programban:

```
#include <iostream>
```

- Az iostream a standard névtérbe tartozik:

```
using namespace std;
```

- Kiírás a **cout**, beolvasás a **cin** utasítással történik, valamint a **<<** és **>>** operátorokkal, amikkel több kiírandó értéket választhatunk el
  - nem tévesztendőek össze a bitenkénti operátorokkal
  - bekérésnél csak változóba olvashatunk be értéket
  - kiírásnál az operátorok között lehetnek változók és konstansok tetszőleges típusból, illetve tördelőjelek, például a sorvége jel (**endl**)
  - pl.:

```
int a;
cin >> a;  // a bekérése
cout << "A értéke: " << a;  // kiíratása
cout << "Egy sor" << endl << "Másik sor";
                  // sortörés beiktatása
```

C++ alapismeretek

Forrásfájlok

- C++ programok a **`.cpp`** kiterjesztésű fájlok, ezek szerkeszthetőek szövegszerkesztővel, vagy valamilyen C++ programozási környezettel (pl.: DevC++, KDevelop, Eclipse, Code::Blocks, Visual Studio, ...)
- A programozási környezetek általában projektekben dolgoznak, hogy több fájlból álló programokat is könnyen tudjunk kezelni
- Mivel nincs automatikus kódformázás, nekünk kell figyelnünk arra, hogy:
  - a programunk kinézete, tabulálása megfelelő legyen
  - ne keverjük a kis-nagybetűket sehol, hiszen a C++ megkülönbözteti őket

Programok fordítása

- Ha megszerkesztettük a programot, fordítanunk kell, ezt a feladatot a *fordítóprogram* látja el
  - több lépésben végzi a feladatát, először assembly kódot generál, majd abból gépi kódot
  - vannak előfordítási lépések, mint a fájltartalom bemásolás, illetve a definíciók behelyettesítése
- Fordítás közben üzeneteket kapunk, ezek lehetnek:
  - hibák (*error*): hiba, amiért nem sikerült lefordítani a programot, ezeket javítanunk kell
  - figyelmeztetés (*warning*): nem hibák, de lehetséges, hogy futási idejű hibát okoznak, ezeket javasolt megnézni, és javítani, ha gondoljuk

- a fordító a legjobb tudomása szerint adja meg a hiba helyét, de ez lehet téves jelzés is

- Mi a **g++** és **mingw** fordítóprogramokat fogjuk használni
- A **g++** elérhető linux/unix alatt is, így a **digitus** szerveren is, használata:

  g++ <*kapcsolók*> <*fájlnév*$_1$> ... <*fájlnév*$_n$>

  - pl.: **g++ main.cpp**
  - alapértelmezetten egy **a.out** nevű programot készít, de ezt a **-o <fájlnév>** kapcsolóval megváltoztathatjuk
  - a **-w** kapcsolóval kikapcsolhatjuk a figyelmeztetéseket,

    -Wall megjeleníti az összes lehetséges hibalehetőséget
  - a **-pedantic** csak a szabvány kódot fogadja el

Feladat: Írjuk ki egy egész szám rákövetkezőjét.

- kell egy egész típusú változó, legyen a neve ‘a’
- az értékét inkrementálással megnöveljük
- használjuk a konzolról való bekérést és kiíratást, tehát szükségünk van az **`iostream`**-re

Specifikáció:

- bemenet: egész szám (*a*)
- kimenet: *a* + 1

Megoldás:

```
#include <iostream>
using namespace std;

int main(){      // főprogram
   int a;        // a nevű, egész típusú változó
   cin >> a;     // a értékének bekérése
   cout << ++a; // a érték megnövelése, kiíratás
   return 0;     // visszatérési érték
}                // főprogram vége
```

Feladat: Olvassunk be egy egész és egy valós számot, és írjuk ki a hányadosukat.

- egy egész és egy valós számot olvasunk be egymás után, az első számot osztjuk a másodikkal, az eredményt egy harmadik valós számba tesszük

Specifikáció:

- bemenet: egy egész szám (*a*) és egy valós szám (*b*)
- kimenet: a két szám hányadosa a harmadik (*c*) számban

Megoldás:

```
#include <iostream>
using namespace std;

int main(){
   int a; // egész szám
   float b, c; // valós számok
   cout << "Első szám: "; cin >> a;
   cout << "Második szám: "; cin >> b;
   c = a / b;
   cout << "Hányados: " << c << endl;
   return 0;
}
```

Feladat: Döntsük el egy egész számról, hogy páros-e.

- a bekérés előtt írassuk ki a képernyőre, hogy mit kérünk be
- elágazás segítségével szövegesen adjuk meg a választ
- párosság eldöntése: 2-vel való osztás maradéka

Specifikáció:

- bemenet: egész szám (*szam*)
- kimenet: „A szám páros”, ha *szam* páros, „A szám páratlan”, ha *szam* nem páros

Megoldás:

```
#include <iostream>
using namespace std;

int main(){
   int szam;
   cout << "A szám: "; cin >> szam;
   if (szam % 2 == 0) // ha páros
      cout << "A szám páros.";
   else // ha nem páros
      cout << "A szám páratlan.";
   return 0;
}
```

Feladat: Írjunk ki N darab *-ot a képernyőre.

- N értékét bekérjük a felhasználótól
- egy számláló ciklusban minden lépésben kiírunk egy csillagot, legyen a ciklusváltozó i

Specifikáció:

- bemenet: egész szám (*n*)
- kimenet: *n* db csillag

Megoldás:

```
#include <iostream>
using namespace std;

int main(){
   int n;
   cout << "A csillagok száma: ";
   cin >> n;
   for (int i = 0; i < n; i++){
      // számláló ciklus az i ciklusváltozóval
      cout << "*";
   }
   return 0;
}
```

Feladat: Adjuk meg egy természetes szám valódi osztóinak számát.

- természetes számot nem tudunk bekérni, úgyhogy ellenőrizzük le, hogy amit bekértünk egész szám, az pozitív-e, különben lépjünk ki
- számlálás programozási tételét használjuk
- ciklusban nézzük meg minden nála kisebb számról, hogy osztója-e, ha igen, növeljük a számlálót

Specifikáció:

- bemenet: egy egész szám (*sz*)
- kimenet: a valódi osztóinak száma (*c*)

Megoldás:

```
#include <iostream>
using namespace std;

int main(){
   int sz, c = 0;  // sz a szám, c a számláló,
                   // amelyet rögtön lenullázunk
   cin >> sz;

   for (int i = 2; i < sz; i++){ // 2-től n-1-ig
      if (sz % i == 0) // ha valódi osztó
         c++; // növeljük c-t
   }
```

Megoldás:

```
    /* lehetett volna ilyen ciklus is:
    int i = 2; // számláló kezdőérték
    while (i < n){
        if (sz% i == 0) c++;
        i++; // számláló növelés
    }
    ekkor i érvényes lesz a külső blokkban is
    */

    cout << sz << „ valódi osztók száma: " << c
                << endl; // kiírjuk az eredményt
    return 0;
}
```

Feladat: Olvassunk be 5 egész számot a képernyőről, és adjuk meg, van-e köztük negatív érték.

- először olvassuk be a számokat, majd egy második ciklusban keressük meg, hogy van-e negatív érték (lineáris keresés)
- az értékeket el kell tárolnunk egy 5 hosszú egész tömbben
- használjunk számláló ciklusokat, a feltételét ki kell egészítenünk a logikai értékkel

Specifikáció:

- bemenet: 5 egész szám (*t*)
- kimenet: van-e negatív érték (*l*)

Megoldás:

```
#include <iostream>
using namespace std;

int main(){
   int t[5];    // 5 egész számból álló tömb

   for (int i = 0; i < 5; i++)
      cin >> t[i]; // tömb i. elemének bekérése
```

Megoldás:

```
     bool l = false; // logikai érték
     for (int i = 0; i < 5 && !l; i++)
        // a ciklusszámláló mellé másik feltétel
        l = (t[i] < 0);

      if (l)
         cout << "Van negatív szám!";
      else
         cout << "Nincs negatív szám!";
      return 0;
  }
```

Feladat: Add meg egy valamilyen hosszú valós számsorról, hogy hány eleme kisebb az átlagnál.

- először ki kell számolnunk az átlagot (összegzés), majd utána meg kell adnunk a kisebb elemek számát (számlálás)
- a bekérés és az összegzés kerülhet az első ciklusba, a számlálás a másodikba, mindegyik számláló ciklus lesz
- használjunk tömböt a tároláshoz, definíciót a méret megadásához

Specifikáció:

- bemenet: valós számok tömbje (*vtomb*)
- kimenet: az átlagnál (*atlag*) kisebb elemek száma (*kisebb*)

Megoldás:

```
#include <iostream>
using namespace std;

#define MERET 10 // definiáljuk a méretet 10-nek

int main(){
   double vtomb[MERET], atlag = 0, kisebb = 0;
   // változók létrehozása, és kezdeti érték
   // beállítás
   cout << "Bemenő számok: " << endl;
   for (int i = 0; i < MERET; i++){
      cout << i+1 << ". szám: ";
      cin >> vtomb[i]; // bekérés
```

Megoldás:

```
        atlag += vtomb[i]; // hozzáadás
    }
    atlag = atlag / MERET; // átlagszámítás

    for (int i = 0; i < MERET; i++){
    // használhatom ugyanazt a változónevet
    if (vtomb[i] < atlag) // kisebb számok
        kisebb++;
    }
    cout << "Az átlagnál " << kisebb
         << " kisebb szám van.";
    return 0;
}
```

Feladat: Írjuk a kimenetre a megadott szavak első betűjét, amíg ‘A’-val nem kezdődik egy szó.

- használjunk egy utántesztelő ciklust, minden lépésben kérjük be szót a képernyőről, majd írjuk ki az első betűjét
- lekérdezhetjük a szó 0. indexén lévő karaktert

Specifikáció:

- bemenet: szavak (*szo*) egymásután
- kimenet: minden szó első betűje

Megoldás:

```
#include <iostream>
#include <string> // kell használnunk a stringet
using namespace std;

int main(){
   string szo;
   do{
      cin >> szo;
      cout << szo[0] << endl;
      // első karakter kiírása új sorba
   } while (szo[0] != 'A');
   return 0;
}
```

Többágú elágazások

- Lehet olyan elágazást készíteni, amely egy változó aktuális értékének függvényében futtat le adott utasításokat, ez a többágú elágazás, erre a **`switch`** utasítás szolgál
- Csak meghatározott értékek esetén fut le az adott ág, nem lehet értékhatárokat, illetve egyéb kifejezéseket megadni, mint az egyszerű elágazás esetében, az adott ágat a **`case <érték>`** feltétellel kezdjük
- Egy ág végét a **`break`** utasítással jelöljük, ha ezt elmulasztjuk a következő ág első utasítása következik
  - így lehetőségünk van több ág utasításait egymást követően lefuttatni
- Lehet különben ágat készíteni, ennek jelölése **`default`**

Többágú elágazások

```
switch(<változónév>){
   case <érték1>:
      // az érték típusa kompatibilis a
      // változóval
      <utasítások>
      break;
      // ekkor befejeződik az ág futása, ha ezt
      // nem írjuk, akkor alatta folytatódik
   case <érték2>:
      <utasítások>
      break;
   …
   default: <utasítások>
}
```

Feladat: Kérjünk be egy számot, és írjuk ki szövegesen a megfelelőjét jegyben.

- ha nem 1-5-ig terjedő a szám, akkor írjuk ki, hogy nem jegy
- használjunk többágú elágazást 5+1 ággal

Specifikáció:

- bemenet: egy egész szám (*a*)
- kimenet: szöveges megfelelője jegyben

Megoldás:

```
#include <iostream>
using namespace std;
```

Többágú elágazások

```
int main(){
   int a; cin >> a;
   switch (a){ // többágú elágazás
      case 1: cout << "elégtelen"; break;
         // break-nél az elágazás végéhez megy
      case 2: cout << "elégséges"; break;
      case 3: cout << "közepes"; break;
      case 5: cout << "jó"; break;
      case 5: cout << "ötös"; break;
      default: cout << "nem jegy"; // különben
   }      // elágazás vége
   cout << endl;
   return 0;
}
```

Többágú elágazások

- A **`default`** ágat nem kötelező megírni
- Lehetőségünk üres ágakat is írni, ezzel lehetőségünk van több értékhez ugyanazt az ágat rendelni

```
switch(<változónév>){
    case <érték1>:
    case <érték2>:
    case <érték3>: <utasítások>
         // ezek az utasítások a négy érték
         // bármelyike esetén lefutnak
       break;
    …
    default: <utasítások>
}
```

Véletlen generálás

- A C++ lehetőséget ad véletlen számok előállítására, ehhez két utasításra (függvényhívásra) van szükségünk:
  - **`srand(<kezdőérték>)`**: inicializálja a generátort
  - **`rand()`**: megad egy egész értékű pozitív véletlenszámot
- Az utasítások a **`stdlib.h`** fájlban vannak, ezért ezt be kell ágyaznunk a programba (**`#include <stdlib.h>`**)
- A generálás mindig a kezdőértékhez viszonyítva történik, ezért olyan kezdőértéket kell megadnunk, amely biztosítja a folyamatos változatosságot a generált számok között
  - ezért általában az aktuális időpillanatot szokás megadni, amit lekérdezhetünk a **`time(0)`** függvénnyel (ehhez használnunk kell a **`time.h`** fájlt)

Véletlen generálás

Feladat: Generáljunk 5 véletlenszámot 0 és 100 között, és írjuk ki őket a képernyőre.

- mivel a generátor tetszőlegesen nagy számot adhat, az értéket korlátoznunk kell úgy, hogy maradékosan osztjuk, jelen esetben 100-zal, így minimum 0-t, maximum 99-t kaphatunk

Specifikáció:

- bemenet: nincs
- kimenet: 5 egész szám 0 és 100 között

Véletlen generálás

Megoldás:

```
#include <iostream>
#include <stdlib.h> // kell a rand()-hoz
#include <time.h> // kell a time(0)-hoz
using namespace std;

int main(){
   srand(time(0));
   // véletlengenerátor inicializálása
   for (int i = 0; i < 5; i++)
      cout << (rand() % 100) << endl;
      // generálás, majd maradékvétel
   return 0;
}
```

Feladatok

I. Kifejezések:

8. a) Add meg egy számtani sorozat első két elemének ismeretében a harmadik elemét.

   b) Add meg az N-edik elemét.

9. (*) Számítsd ki egy háromszög területét az oldalhosszaiból.

II. Vegyes feladatok:

1. c) Rajzolj ki egy N hosszú befogójú, egyenlő szárú derékszögű háromszöget *-okból.

5. Sorold fel a K-nál kisebb négyzetszámokat.

13. a) Egy két tagú névnek add meg a monogramját.

Feladatok

IV. Tömbök:

3. Vektor szórása (átlagtól való eltérések átlaga).
4. Van-e két egyforma elem a vektorban?

III. Programozási tételek:

7. (*) Add meg, hogy az A és B közötti egész számok közül melyiknek van a legtöbb valódi osztója.
15. a) Egy szigorúan növő egész számsorban add meg a legnagyobb ugrást (szomszédos elemek közötti legnagyobb előforduló különbséget).
27. Add meg egy tetszőleges szöveg leghosszabb szavát.