Bevezetés a programozásba
8. Gyakorlat

Ismétlés, fájlkezelés

Ismétlő feladatok

1. feladat - Kérjünk be három pozitív egész számot a konzolról
2. feladat - Vizsgáljuk meg, lehetnek-e ezek a számok egy háromszög oldalai
3. feladat – Keressük meg a legnagyobbat közülük
4. feladat – Keressük meg az első két szám közös osztóit
5. feladat - Töltsünk fel egy 10 hosszú tömböt páros számokkal, majd írassuk ki a konzolra
6. feladat - Írassuk ki a tömböt visszafelé

C++ beolvasások (konzol)

- iostream
  - cin // egyszerű típus függő beolvasás
  - getline() // sorvégéig tartó beolvasás
- cstdio
  - getchar() // karakter beolvasás
- cin.ignore(255,'\n'); // a következő 255 karakter ignorálása vagy ENTER leütésig ignorálás
- cin.sync( ); // a még beolvasatlan karakterek eldobása
- **Hiba típus:**

```
cin >> val1;          // => Enter-t a konzol bemeneti buffer-ében hagyja
getline(cin, valt2);  // => üres értéket olvas be, hisz Enter-ig olvas
```

*ezért a **cin** után érdemes takarítani (**cin.sync();**)*

Szöveg számmá konvertálása

- A fájlból az adatokat getline() függvény segítségével olvassuk be, akkor a szám értékeket szövegből kell átalakítanunk
- Ehhez használt eszköz: **stringstream**
  - Használata: #include <sstream>
  - Karakterláncot (string) tárol
  - Hasonlóan működik mint a ifstream és a ofstream, azaz << és >> operátorok segítségével lehet belőle olvasni
  - Ha egy szám értéknek megfelelő karakterláncot teszünk bele, akkor kiolvasható belőle számként

Szöveg számmá konvertálása

Példa I: *(szükséges hozzá a: #include <sstream>)*

```
string szam = „25”;
stringstream ss;
int i;
ss << szam;
ss >> i;          // Ennek hatására az i egész
                  változó értéke 25 lesz
```

Példa II: *(szükséges hozzá a: #include <stdlib.h>)*

```
string number = „34”;
int result = atoi( number.c_str() );
```

vagy

```
char someChars[] = „47";
int result2 = atoi( someChars );
```

vagy

```
int result3 = atoi( „84” );
```

Véletlen számok generálása

- A C++ is ad lehetőséget véletlen számok előállítására, ahogy azt a PLanG-ban láttuk, ehhez két utasításra (függvényhívásra) van szükségünk:
  - *srand(<kezdőérték>)* : egy adott kezdőértékkel inicializálja a véletlen szám generátort
  - *rand()* : megad egy egész értékű pozitív véletlen számot
- 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)
  - A generált értékkel utána bármit tehetünk, mint a szokványos értékeinkkel

Véletlen számok generálása

- `#include <time.h>`
  - Az aktuális idő lekéréséhez szükséges library
- `#include <cstdlib>` (ez is jó: `#include <algorithm>`)
  - Az srand ebben a library-ban található
- `void srand( unsigned int seed);`
  - Majd az ‘aktuális időpillanat’ seedértékkel inicializáljuk a véletlen szám generátort
- `int rand( void );`
  - Megad egy egész értékű pozitív véletlen számot

Programozási feladat

- Számkitalálós játék
  - Írjuk egy olyan interaktív játékprogramot, amely gondol egy számra 1 és 100 között és a felhasználónak kell kitalálni, hogy melyik ez a szám!
  - A játékos tippelhet egy számot, melyről a program megmondja, hogy kisebb vagy nagyobb a gondolt számnál illetve sikerült-e kitalálni a helyes számot.
  - A program számolja azt is, hogy hányadik tippre sikerült meghatározni a helyes számot!

- **A profibbaknak**
  - **Írjál újrajátszási lehetőséget a játékhoz!**
  - **A program jegyezze meg az aktuális futás legjobb menetét!**
  - **Legyen tartós „HighScore”-ja a programnak!**

Megoldás 1. feladat

```
#include <iostream>
#include <cstdlib>
#include <time.h>
using namespace std;

int num;
void szam_be()
{
   while(!(cin >> num))
    {
      cout<<"HIBA, ez nem szam! Kerek egy masik szamot!\n";
      cin.clear();
      cin.ignore();
    }
}

int main()
{
   srand(time(0));
   int rand_num=rand()%100;
   cout<<"Kerek egy szamot?\n";
   szam_be();

   int counter=0;
   bool talalat;
   bool jatek;

while(!jatek){
  if(num!=rand_num){
     counter++;
     if(num<rand_num){
        cout<<"A szam tul kicsi!Kerek egy masik szamot!\n";
        szam_be();
     }else{
        cout<<"A szam tul nagy!Kerek egy masik szamot!\n";
        szam_be();
     }
  }else{
     cout<<"A valasz helyes. A tippelasek szama: " <<counter<<"\n";
     talalat=1;
     cout<<"Szertnel meg jatszani, igen vagy nem?\n";
     string valasz;
     cin>>valasz;
     if(valasz=="nem"){
        jatek=1;
     }else{
        rand_num=rand()%100;
        cout<<"Kerek egy szamot!\n";
        counter=0;
        szam_be();
     }
     }
  }

  return 0;
}
```

C++ fájlkezelés

- A fájlkezelés módjával már megismerkedtünk PlanG-ban, C++-ban teljesen hasonló
  - fizikai fájlnevek reprezentálják a tényleges fájlokat a tárolókon
  - logikai fájlnevek a programban használt változók, amelyeket társítanunk kell a fizikai fájlnévhez
  - vannak külön kimeneti és bemeneti fájlok, egyszerre egy fájl csak egy csatornán lehet
  - az adatokat sorban olvassuk ki és írjuk ki a fájlokba, ugrálni nem lehet az adatok között
  - az előreolvasási technika használható
- De C++-ban már tényleges fizikai fájlokat kezelünk, nem csak szimulált fájlokat, ezért még óvatosabbnak kell lennünk a kezelésükkel! (felülírás, törlés…)

Fájltípusok

- A fájlműveletek használatához az *fstream* definíciós fájlra lesz szükségünk: `#include <fstream>`
- A fájl típusok az std névtérben találhatóak:
  ```
  using namespace std;
  ```
- Két logikai fájltípust használhatunk, mindegyikben bármilyen adat található, amelyet bármilyen sorrendben kezelhetünk:
  - bemeneti fájl: **ifstream**
  - kimeneti fájl: **ofstream**
  - a fájl a deklarálástól a programblokk végéig él, de csak akkor használhatjuk, ha társítjuk fizikai fájlhoz (különben futási idejű hibát kapunk)
  - egy logikai fájlt több fizikai fájlhoz is társíthatunk egymás után

Fájlok megnyitása

- A logikai és fizikai fájl társítását kétféleképpen végezhetjük:
  - a logikai fájlnév deklarációját követően:

    <logikai fájlnév>.open(<fizikai fájlnév>)

    parancs segítségével, *pl.:*

```
ifstream f;                    //‘f’ nevű logikai fájl
f.open(”adatok.txt”);          //‘f’-t az ‘adatok.txt’ fájlhoz társítjuk
```

  - a logikai fájl definíciójával együtt megadott paraméterben, pl.:

```
ifstream f(”adatok.txt”);
//f már a létrehozásától az adatok.txt fájlhoz van társítva
```

Fájlok megnyitása

- Fizikai fájlnévként karaktertömb (char[ ]) típusú adatokat adhatunk meg, amit lehet változó segítségével is, ezért a program futása közben is bekérhetjük a fájlnevet
- Ha string típusú változóba szeretnénk bekérni a fájlnevet, akkor azt át kell alakítanunk
  - a string típusban található egy olyan függvény, amely karaktertömbbé alakítja a szöveget, ezt használjuk: *<változónév>.c_str()* pl.:

```
ifstream f;                 // logikai fájl
string fnev;                // egy string (a fizikai fájl neve)
cin >> fnev;                // beolvassuk a stringet
f.open(fnev.c_str());       // a beolvasott fájlnevet
                               próbáljuk megnyitni
```

Megjegyzés: A C++11-es szabványtól kezdve nem szükséges a string karaktertömbbé alakítása, felismeri az open() függvény.

Fájlok megnyitása

- Nem garantált, hogy a megadott elérési úton van is egy fájl, amit a program megnyithat, ezért mielőtt bármilyen tevékenységet végzünk rajta, célszerű ellenőrizni a fájl helyességét
- A megnyitás sikerességét a *<logikai fájlnév>.fail()* függvénnyel kérdezhetjük le, ha ez igaz, akkor nem sikerült megnyitni a fájlnevet, pl.:

```
ifstream f;
f.open(”data/bemenet.dat”); // megnyitás
if(f.fail())                // ha nem sikerült megnyitni
  cout<< ”Nem sikerült megnyitni a fájlt!”;
else { … }                  // ha sikerült megnyitni ebben az ágban dolgozhatunk a fájlon
```

Fájlok megnyitása

- Ha a fájlnevet a felhasználótól kérjük be, célszerű a megnyitást ciklusba foglalni, addig kérjünk be új fájlnevet a felhasználótól, amíg nem sikerül megnyitni az adott fájlt, pl.:

```
ifstream f;
string fnev;
cin >> fnev;
f.open(fnev.c_str());          //fájl megnyitása
while(f.fail()){               //ha nem sikerült megnyitni
  f.clear();                   //fájl újrainicializálás
  cout<< ”Hibás fájlnév!”<< endl;
  cin >> fnev;
  f.open(fnev.c_str());        //megnyitás az új névvel
} //addig kéri be újra fájlnevet, amíg nem sikerül  megnyitni
```

Fizikai fájlok zárása, váltása

- Fizikai fájlt használat után be kell zárni
  - a bezárás automatikusan megtörténik a program végén, de azért célszerű mindenképpen külön bezárást végezni, amint befejeztük a programban a fájl használatát
  - bezárni a *<logikai fájlnév>.close()* függvénnyel lehet, pl.: `f.close();`
  - bezárást követően a logikai fájlnevet újra használhatjuk másik fizikai fájl kezelésére, pl.:

```
ifstream input(”adat.txt”); // adat.txt megnyitása
input.close();              // adat.txt bezárása
input.clear();              // input flage-ek kitörlése
input.open(”adat2.txt”);    // adat2.txt megnyitása
```

Adatok olvasása, írása

- A fájlműveletek kimeneti fájlnál az olvasás (>> operátor), bemeneti fájlnál az írás (<< operátor)
  - a logikai típustól függően csak az adott művelet használható
  - írhatunk sortörést az **endl** utasítással
  - az operátorokat ugyanúgy használjuk, mintha a képernyőre írnánk, azzal a különbséggel, hogy a fájlt adjuk meg célként/forrásként a képernyő helyett
  - pl.:

```
ofstream f(”kimenet.txt”);
f << ”File első sora” << endl << ”Második sor”;
   // tetszőlegesen kiírhatunk dolgokat a fájlba
f.close();
```

Adatok olvasása, írása

- Egy fájlba bármilyen adat lehet bármilyen sorrendben, olvasásnál ezért ügyelni kell arra, hogy mindig a megfelelő típusú adatot olvassuk ki.
- A fájl végén most is ott van az EOF jel, amit a <logikai fájlnév>.eof() függvénnyel tudunk lekérdezni, ez igaz értéket ad vissza, ha a végére értünk, pl.:

```
ifstream f;
int data;
f.open(”adatok.txt”);
f >> data;
while(!f.eof()){                    //amíg nincs vége a fájlnak
   …
   f >> data;                       //addig olvasunk
}
```

- Beolvasásnál ugyanúgy beveszi az EOF jelet, ezért célszerű előreolvasási technikával feldolgozni a fájlokat

C++ fájlkezelés

- Amire érdemes odafigyelni:
  - mielőtt elkezdünk beolvasni egy fájlból, nézzük meg, hogy sikerült-e megnyitni
  - a megnyitott fájlt mindig zárjuk be, ha már nincs szükségünk rá
  - bemeneti fájlba ne írjunk, kimeneti fájlból ne olvassunk
  - ne olvassunk tovább, ha már elértük az EOF jelet, ha nem előreolvasást használunk és nem vagyunk biztosak a fájl tartalmában, akkor ellenőrizzük le, nem értünk-e a fájl végére, mielőtt felhasználnánk az utolsó beolvasott adatot
  - különösen figyeljünk arra, hogy ne írjunk végtelen ciklusba fájl kiírást
  - ha egy fájlt újra megnyitunk írásra, akkor (alapból) korábbi tartalma törlődni fog a megnyitáskor

Sorok beolvasása

- Alkalmazhatjuk a teljes sor beolvasására szolgáló *getline(<logikai fájlnév>, <szöveg változó>)* utasítást, amely a teljes sor tartalmát kiolvassa - a sortörés kivételével - egy szöveg változóba
- Pl.:

```
ifstream bemenet(”valami.x”);
string s;
getline(bemenet, s);                (előreolvasás)
while(!bemenet.eof()                (amíg nincs vége a fájlnak)
)
{                                   (műveletek s-sel)
  …                                 (következő sor beolvasása)
  getline(bemenet, s);
}
```

Sorok beolvasása

- A sorbeolvasás egy speciális változata, amikor nem a teljes sort olvassuk be, csak egy részét
- Lehetőségünk van egy adott karakterig olvasni a sort:
- getline(<fájlnév>, <változó>, <karakter>)
  - ekkor a harmadik helyen megadjuk azt a karaktert, aminek első előfordulásánál megáll az olvasás
  - a karakter lehet vezérlőkarakter is (pl. tabulátor, '\t')
  - kiegészítve az előző változattal, több lépésben olvashatjuk be a teljes sort
- Pl. egy (legalább két 0-t tartalmazó) sor beolvasása három lépésben:

```
getline(f, s1, '0');        (beolvasás az első 0-ig)
getline(f, s2, '0');        (beolvasás a második 0-ig)
getline(f, s3);             (a maradék beolvasása)
```

Programozási feladat II.

- Összeg számolás fájlból
  - írjuk ki a képernyőre a „szamok.txt” fájlban tárolt számok összegét

- Észrevétel:
  - A **cin** ***whitespace***-ig olvas, ha az ***eof*** az utolsó ábrázolható karakter után van, akkor azt is hozzá olvassa a **cin** a többihez és a beolvasási csatorna fájl vége **flag**-e aktiválódik. Emiatt lehet az, hogy utoljára beolvasott adatot már nem dolgozza fel a ciklus. (Egyszerű megoldás: **enter** ütése az utolsó adat után)

Megoldás (1) 2. feladat

```
#include <iostream>
#include <fstream>
#include <string>
#include <stdlib.h>

using namespace std;

int main()
{
    ifstream bemenet("szamok.txt");
    ofstream kimenet("szamok_kimenet.txt");

    if(bemenet.fail()){
    cout<< "Nem sikerult megnyitni a fajlt!";
    }

    int sum=0;
     string szam;

while(!bemenet.eof())
    {
                getline(bemenet, szam,' ');
                sum=sum+atoi(szam.c_str());
    }
     kimenet<<"A szamok osszege "<<sum<<"\n";
     cout<<"A szamok osszeget a szamok_kimenet.txt kimeneti fajlban megatalalod";
     bemenet.close();
     kimenet.close();

     return 0;
}
```

Megoldás (2) 2. feladat – haladó

```
#include <iostream>
#include <fstream>
#include <stdlib.h>

#include <stdio.h>
#include <cstring>

using namespace std;

int main()
{
   ifstream bemenet("szamok.txt");
   ofstream kimenet("szamok_kimenet.txt");

   if(bemenet.fail()){
   cout<< "Nem sikerult megnyitni a fajlt!";
   }

   string szam_sorozat;
   int sum=0;
   char * number;

while(!bemenet.eof())
  {
  getline(bemenet, szam_sorozat);
  const char * elvalaszt=" ,.-";
  char * aktualis_bemente= const_cast<char*>(szam_sorozat.c_str());
  number = strtok (aktualis_bemente,elvalaszt);

     while(number != NULL)
     {
        sum=sum+atoi(number);
        number = strtok (NULL, elvalaszt);
     }
  }

   kimenet<<"A szamok osszege ("<<szam_sorozat<<"): "<<sum<<"\n";
   cout<<"A szamok osszege ("<<szam_sorozat<<"): "<<sum<<"\n";
   cout<<"A szamok osszeget a szamok_kimenet.txt kimeneti fajlban megatalalod";
   bemenet.close();
   kimenet.close();

   return 0;
}
```

Megoldás (3) 2. feladat – haladó++

```
#include <iostream>
#include <fstream>
#include <string>
#include <stdlib.h>

using namespace std;

int main()
{
    ifstream bemenet("szamok.txt");
    ofstream kimenet("szamok_kimenet.txt");

    if(bemenet.fail()){
    cout<< "Nem sikerult megnyitni a fajlt!";
    }

    string szam_sorozat;
    int sum=0;
    int length_of_number=0;
    int index=0;

while(!bemenet.eof())
    {
        getline(bemenet, szam_sorozat);
        for ( string::iterator it=szam_sorozat.begin(); it!=szam_sorozat.end(); ++it)
        {
            if(!isdigit(*it)){
                sum=sum+atoi(szam_sorozat.substr (index-length_of_number, length_of_number).c_str());
                length_of_number=0;
            }else{
                length_of_number++;
            }
            index++;
        }
    }
    kimenet<<"A szamok osszege ("<<szam_sorozat<<"): "<<sum<<"\n";
    cout<<"A szamok osszeget a szamok_kimenet.txt kimeneti fajlban megatalalod";
    bemenet.close();
    kimenet.close();

    return 0;
}
```

Házi feladatok

- 2.32 Fésülj össze két monoton sorozatot fájlból és írd ki azt egy harmadik fájlba (egy elem innen, egy onnan…)
  Bemeneti fájlok nevei: sorozat_be1, sorozat_be2
  Kimeneti fájl neve: sorozat_ki
- 3.27 Add meg egy tetszőleges (fájlból beolvastott) szöveg leghosszabb szavát és írasd ki azt egy fájlba .
  Bemeneti fálj neve: leghosszabb_szo_be
  Kimeneti fálj neve: leghosszabb_szo_ki
- 5.1 Véletlen tömb: egy tömb elemeit töltsd fel véletlen számokkal.
- 5.3 Vektor szórása (átlagtól való eltérések átlaga) ***(matematikailag nem teljesen korrekt, de nekünk jó lesz)***