Pázmány Péter Katolikus Egyetem
Információs Technológiai Kar

Bevezetés a programozásba I

8. gyakorlat

C++: szövegfolyamok, intelligens tömbök

groberto@inf.elte.hu
http://people.inf.elte.hu/groberto

Adatfolyamok

Hibalehetőségek

- Adatfolyamnak nevezzünk azt, ahol sorban tetszőleges számú adat megjelenhet a bemeneten, vagy a kimeneten, pl. konzol képernyő, fájl
- Bemenő adatok esetén sok esetben előfordulhat, hogy a bemenő adatfolyam nem a várt adatot, vagy adattípust tartalmazza
  - pl. pozitív helyett negatív szám, szám helyett szöveg, hibás fájlnév, stb.
  - bizonyos hibák kezelhetőek a beolvasás után, ám amennyiben a beolvasandó elem típusa nem megfelelő, akkor a beolvasás sem történik meg (pl. szám helyett szöveget ad meg a felhasználó)

Adatfolyamok

Hibakezelése

- Minden adatfolyam esetén lehetőségünk van az állapot lekérdezésére
  - a **`good()`** művelet megadja, hogy az adatfolyam konzisztens állapotban van-e és elérhető
  - a **`fail()`** művelet megadja, hogy a legutolsó művelet sikertelen volt-e
  - az **`eof()`** művelet megadja, hogy az adatfolyamnak vége van-e
  - a **`clear()`** művelet helyreállítja az elromlott állapotot
  - továbbá minden beolvasás (**`>>`**, **`getline`**) esetén a művelet visszaadja, hogy sikeresen megtörtént-e a beolvasás az adott típusú változóba

Adatfolyamok

Hibakezelése

- Pl.:

```
int num;
cin >> num; // megpróbálunk számot beolvasni
if (cin.fail()) // ha sikertelen volt a művelet
   cout << "A megadott érték nem szám!";
else
   cout << "A beolvasás sikeres!";

// ugyanez:
if (cin >> num) // sikeres volt a beolvasás
   cout << "A beolvasás sikeres!";
else
   cout << "A megadott érték nem szám!";
```

Szövegfolyamok

Példa

Feladat: Írjuk ki egy számnak a rákövetkezőjét, de ellenőrizzük azt is, hogy számot adott-e meg a felhasználó.

- ellenőrizzük a bemeneten, hogy sikeres volt-e a beolvasás

Specifikáció:

- bemenet: egy szám (*szam*)
- kimenet: a szám rákövetkezője, vagy a hiba jelzése

Szövegfolyamok

Példa

Megoldás:

```
#include <iostream>

int main(){
   int szam;

   cout << "Kérek egy számot: ";
   if (cin >> szam) // ha be tudtuk olvasni
      cout << ++szam << endl;
   else // ha nem tudtuk beolvasni
      cout << "A megadott érték nem szám!";
   return 0;
}
```

Szövegfolyamok

Használata

- A szövegfolyamok olyan folyamok, ahol az írás és olvasás szövegen keresztül történik, így nem íródik ki fájlba, vagy a konzolra
  - tetszőleges típusú változót írhatunk be és olvashatunk ki
  - alkalmas olyan típuskonverziókat elvégezni, amik automatikusan nem történnek meg (pl. szöveg-szám)
- A szövegfolyam típusa a **`stringstream`**, használatához szükséges az **`sstream`** fájl és az **`std`** névtér
  - pl.

```
#include <sstream>
…
std::stringstream sstr;
```

Szövegfolyamok

Műveletei

- A szövegfolyamba tetszőleges értéket a **<<** operátorral, vagy a **getline** művelettel helyezhetünk, az egymás után bekerült értékek egymás után kerülnek
- A szövegfolyamból tetszőleges értéket a **>>** operátorral vehetünk ki (a behelyezés sorrendjének megfelelően)
  - amennyiben sikertelen a kivétel (az érték nem konvertálható az adott típusba, vagy üres a folyam), akkor a **fail()** művelettel lekérdezhető, hogy hiba történt-e
  - a szövegfolyam végét az **eof()** művelettel ellenőrizhetjük
- Az **str()** művelet kiírja, vagy átalakítja **string** formátumúvá a teljes tartalmát, míg az **str(*<szöveg>*)** kitörli a korábbi tartalmat, és beírja az újat

Szövegfolyamok

Műveletei

- Pl átalakítás szöveggé:

```
int num; cin >> num;
stringstream converter;
   // az átalakítást szövegfolyammal végezzük
converter << num; // behelyezzük a számot
string output = converter.str();
   // szövegként kapjuk meg a tartalmat
```

- Pl. átalakítás számmá:

```
stringstream converter;
string input; cin >> input; converter << input;
int num;
converter >> num; // kiolvasás számként
if (converter.fail()) // sikertelen átalakítás
   cout << converter.str() << " nem szám!";
```

Szövegfolyamok

Példa

Feladat: Írjuk ki egy számnak a rákövetkezőjét, de ellenőrizzük azt is, hogy számot adott-e meg a felhasználó.

- a bemenetet nem rögtön számként, hanem szövegként olvassuk be
- használjunk szövegfolyamot a konverzió elvégzésére, csak akkor végezzük el a növelést, ha sikeres a konverzió

Specifikáció:

- bemenet: egy szám szövegként (*szoveg*)
- kimenet: a szám rákövetkezője, vagy a hiba jelzése

Szövegfolyamok

Példa

Megoldás:

```
#include <iostream>
#include <string>
#include <sstream> // kell a stringstream-hez

int main(){
   string szoveg; int szam;
   stringstream szfolyam; // szövegfolyam

   cout << "Kérek egy számot: ";
   cin >> szoveg;

   szfolyam << szoveg; // beírjuk a szöveget
   szfolyam >> szam; // kiolvasunk egy számot
```

Szövegfolyamok

Példa

Megoldás:

```
    if (szfolyam.fail())
       // ha nem lehetett konvertálni
       cout << szfolyam.str() << " nem szám!";
    else
       // ha lehetett konvertálni
       cout << ++szam << endl;
    return 0;
}
```

Egyszerű tömbök

Használata

- A C++ biztosít számunkra egy beépített tömb típust, amelyen az egyedüli értelmezett művelet az indexelő operátor használata
  - pl.: `int t[10]; cin >> t[0];`
  - lehetőségünk van ezzel a tömbbel mátrixot alkotni több index használatával, pl.:

```
int m[10][5]; // 10*5-ös mátrix
cin >> m[0][0]; // 1. sor 1. eleme
```

  - ezzel a tömbbel több probléma is felmerül: nincs tisztában a saját méretével, csak konstans adható meg méretként, nem lehet a méretét futás közben megváltoztatni
  - jó lenne, ha olyan tömböket is használhatnánk, amelyek a fenti tulajdonságokkal rendelkeznek

Intelligens tömbök

Használata

- A C++ ezért biztosít egy olyan tömb adatszerkezetet, amely a korábbi hiányosságokat pótolja, a neve **`vector`**
  - használatához:
    ```
    #include <vector>
    using namespace std;
    ```
  - létrehozásához speciálisan kell megadnunk a típust, illetve a méretet (amely lehet 0 is), pl.:
    ```
    vector<int> t1(10); // 10 elemű egész tömb
    vector<string> t2(5); // 5 elemű szövegtömb
    ```
  - létrehozáskor a méretet nem kötelező megadni, ekkor egy 0 méretű tömböt hoz létre (ezt persze később módosíthatjuk), pl.: `vector<int> t3; // 0 elemű egész tömb`

Intelligens tömbök

Műveletei

- A **`vector`** leggyakoribb műveletei:
  - elem lekérdezés, felülírás: **`<változónév>[<index>]`**
  - méret lekérdezése: **`<változónév>.size();`**
  - kiürítés: **`<változónév>.clear();`**
  - átméretezés: **`<változónév>.resize(<új méret>);`**
  - új elem behelyezése a tömb végére: **`<változónév>.push_back(<érték>);`**
  - utolsó elem kivétele: **`<változónév>.pop_back();`**
- A tömb méretének tehát változó is megadható, pl.:

```
int size;
cout << "Tömb mérete: ";
cin >> size;
vector<double> v(size);
```

Intelligens tömbök

Példa

Feladat: Olvassunk be valós számokat a bemenetről amíg 0-t nem írunk, majd adjuk meg az átlagnál nagyobb elemek számát.

- kénytelenek vagyunk eltárolni az elemeket, ahogy az összegzés és számlálás tételét is alkalmazni tudjuk
- mivel nem tudjuk az elemek számát, olyan adatszerkezet kell, amelynek mérete növelhető futás közben, tehát **`vector`**-t kell használnunk, és annak a **`push_back`** függvényét

Specifikáció:

- bemenet: valós számok sorozata (*v*)
- kimenet: az átlagnál (*atlag*) nagyobb elemek száma (*c*)

Intelligens tömbök

Példa

Megoldás:

```
#include <vector> // használjuk a vector-t
…
int main(){
   vector<float> v;
   // float típusú vector, alapból 0 hosszú lesz
   float akt; cin >> akt; // előreolvasás
   while (akt != 0) {
      v.push_back(akt); // berakjuk a végére
      cin >> akt;
   }
   /* ugyanez hibaellenőrzéssel:
   while (cin >> akt && akt != 0)
      v.push_back(akt); */
```

Intelligens tömbök

Példa

Megoldás:

```
    float atlag = 0; // összegzés
    for (int i = 0; i < v.size(); i++)
        atlag += v[i]; // kiolvasunk minden elemet
    atlag /= v.size(); // mérettel osztunk

    int c = 0; // számlálás
    for (int i = 0; i < v.size(); i++)
        if (v[i] > atlag) c++;

    cout << "Az átlagnál " << c << " elem
              nagyobb." << endl;
    return 0;
}
```

Intelligens tömbök

Példa

Feladat: Olvassunk be valós számokat tartalmazó sorokat egy fájlból és írjuk ki a sorok átlagának minimumát.

- mivel soronként akarunk feldolgozni, a beolvasást getline segítségével végezzük, amelyből a számokat egy szövegfolyam közbeiktatásával vesszük ki
- az így kapott számokat átlagoljuk (ehhez összegzés és számlálás szükséges), az átlagot eltároljuk egy vektorban, amelyre lefuttatjuk a minimumkeresés

Specifikáció:

- bemenet: valós számok sorozata egy fájlban (f)
- kimenet: a sorok átlagának minimuma (min)

Intelligens tömbök

Példa

Megoldás:

```
#include <iostream>
#include <fstream>
#include <sstream>
#include <vector>
using namespace std;

int main(){
   ifstream f;
   string fname, line;
   stringstream converter;

   // fájlnév bekérése és fájl megnyitása
   cout << "Kérem a beolvasandó fájl nevét: ";
```

Intelligens tömbök

Példa

Megoldás:

```
cin >> fname;
f.open(fname.c_str());
while (f.fail()){
   cout << "Hibás fájlnév!" << endl;
   f.clear();
   cout << "Kérem a beolvasandó fájl nevét: ";
   cin >> fname;
   f.open(fname.c_str());
}
f.close();

double number;
   // számok, amiket kiolvasunk a sorból
```

Intelligens tömbök

Példa

Megoldás:

```
double sum; // összegzéshez
int count; // számláláshoz
vector<double> avgs; // átlagokat tároló vektor

// fájl kiolvasása
getline(f, line); // előreolvasási technika
while (!f.eof()) {
   sum = 0;
   count = 0;
   converter.str(line);
      // beállítjuk a tartalmat a sorra
   converter.clear();
      // a korábbi hibát is kitöröljük
```

Intelligens tömbök

Példa

Megoldás:

```
        while (converter >> number) {
            // konvertálás és hibaellenőrzés
            sum += number;
            count++;
        }
        if (count > 0)
            // ha sikerült beolvasnunk legalább egy
            // számot
            avgs.push_back(sum / count);
                // az átlagot behelyezzük a vektorba

        getline(f, line);
    }
```

Intelligens tömbök

Példa

Megoldás:

```
    // minimum megállapítása
    if (avgs.size() > 0) {
       double min = avgs[0];
       for (int i = 1; i < avgs.size(); i++)
          if (avgs[i] < min) min = avgs[i];
       cout << "A legkisebb sorátlag: " << min
            << endl;
    } else {
       cout << "Nem sikerült egy számot se
                 kiolvasni a fájlból." << endl;
    }
    return 0;
}
```

Intelligens tömbök

Mátrixok kezelése

- Lehetőségünk van mátrixot létrehozni a tömbök tömbje koncepció alapján
  - ekkor a mátrix minden sora maga is egy **`vector`**
  - a típus helyére egy tömbtípust adunk meg
  - ilyenkor csak a külső tömb méretét adhatjuk meg, a belső tömbök 0 méretűek lesznek, így át kell őket később méretezni
  - pl. egy 10*5-ös egészmátrix létrehozása:

```
vector<vector<int> > m(10);
for (int i = 0; i < m.size(); i++)
    m[i].resize(5);
```

  - ezután az indexelés a megszokott módon, pl.: **`m[0][0]`**

Intelligens tömbök

Példa

Feladat: Adott 10 tanuló egyenként 5 jeggyel, állapítsuk meg a legjobb átlaggal rendelkező tanulót.

- ehhez használjunk egy mátrixot, kérjük be a jegyeket egyenként
- határozzuk meg minden hallgatóra az átlagot, majd abból keressük meg a maximumot

Specifikáció:

- bemenet: hallgatók jegyei egy mátrixban (*m*)
- kimenet: a legjobb átlagú hallgató indexe (*max*)

Intelligens tömbök

Példa

Megoldás:

```
…
int main(){
    vector<vector<int> > m(10);
    // sorok létrehozása
    for (int i = 0; i < m.size(); i++){
        m[i].resize(5); // oszlopok létrehozása

        for (int j = 0; j < m[i].size(); j++){
            cout << i+1 << ". tanuló " << j+1
               << ". jegye: ";
            cin >> m[i][j]; // jegyek beolvasása
        }
    }
```

Intelligens tömbök

Példa

Megoldás:

```
vector<float> avg(m.size());
// az átlagokat egy vektorba helyezzük

for (int i = 0; i < m.size(); i++){
   // összegzés végrehajtása minden sorra a
   // mátrixban
   avg[i] = 0;
   for (int j = 0; j < m[i].size(); j++)
      avg[i] += m[i][j];
   avg[i] /= m[i].size(); // átlagot számítunk
}
```

Intelligens tömbök

Példa

Megoldás:

```
    int max = 0;
    // maximumkeresés az átlagok vektorán
    for (int i = 1; i < avg.size(); i++){
        if (avg[i] > avg[max])
            max = i;
    }

    cout  << "Legjobban a(z) " << max+1
          << ". tanuló teljesített." << endl;
    return 0;
}
```

Intelligens tömbök

Feladatok

Az alábbi feladatok megoldásához használjunk hibaellenőrzést, valamint intelligens tömböket!

III. Programozási tételek:

10. Add meg egy tetszőleges egész számsorról, hogy hány eleme nagyobb ill. kisebb az átlagánál. (A számsorozat végét q jelöli.)
11. Számítsd ki két N dimenziós vektor skaláris szorzatát. (A felhasználótól kérjük be N értékét.)
12. Add meg egy tetszőleges egész számsorban a szomszédos elemek átlagos különbségét.

Intelligens tömbök

Feladatok

IV. Tömbös feladatok:

6. (*) Vektor permutálása, hogy végül monoton növekedő sorrendben legyenek az elemek a vektorban.
7. c) Mátrixban tároljuk egy osztály adatait, minden sora egy diák, minden oszlopa egy tantárgy, a mátrix értékei a jegyek. Melyik a legnehezebb tantárgy (legtöbb bukás).
8. a) Adott méretű mátrix feltöltése fájlból, majd (megfelelő méretek esetén) döntsd el, hogy szimmetrikus-e. (A fájl első két adata a mátrix mérete, majd utána jönnek az értékek sorfolytonosan).