Pázmány Péter Katolikus Egyetem
Információs Technológiai Kar

Bevezetés a programozásba I

7. gyakorlat

C++: szövegkezelés, szekvenciális fájlkezelés

© 2011.10.25. Giachetta Roberto
groberto@inf.elte.hu
http://people.inf.elte.hu/groberto

Szövegkezelés

Karakterkezelés

- A karakterek C++-ban ASCII kódjuk alapján vannak eltárolva, ezért amikor karakter típusú változót hozunk létre, akkor igazából egy bájtban tárolt számot ment el a gép
- Íj módon lehetőségünk van a számokra alkalmazott műveleteket karakterekre is alkalmazni, illetve értékül adni őket számoknak, és fordítva, pl.:

```
char ch1 = 'a';
int ch1i = ch1; // ch1i megkapja ’a’ ASCII kódját
int ch2i = 97; char ch2 = ch2i;
   // ch2 megkapja a 97-es ASCII kódú karaktert
```

- Továbbá lehetőségünk van az egész számra alkalmazható műveleteket használni, pl.:

```
char ch = 'a'; cout << ++ch; // kiírja a 'b'-t
```

Szövegkezelés

Karakterkezelés

- A PLanG-ban látott módon lehetőségünk van manipulálni karaktereket, illetve szövegeket
- A karaktereket manipuláló függvények többnyire a **`cctype`** (néha **`cctype.h`**) fájlban vannak, ezért ezt be kell raknunk a programunkba, az utasítások az **`std`** névtérben vannak
- A következő lehetőségeink vannak (a paraméterben konstanst vagy változót adhatunk meg, és visszakapjuk az átalakított karaktert):
  - kisbetűvé alakítás: **`tolower(<karakter>)`**
  - nagybetűvé alakítás: **`toupper(<karakter>)`**
  - betű-e a karakter: **`isalpha(<karakter>)`**
  - szám-e a karakter: **`isdigit(<karakter>)`**

Szövegkezelés

Példa

Feladat: Írjuk vissza a beírt 10 szót nagy kezdőbetűvel a kimenetre.

- elemenkénti feldolgozással végigmegyünk a szavakon
- használjuk a **`toupper()`** utasítást a szó első betűjén, amit indexeléssel le tudunk kérdezni, az átalakított betűt visszatesszük a szóba

Specifikáció:

- bemenet: szavak sorozata (*szo*)
- kimenet: a szavak nagybetűsen

Szövegkezelés

Példa

Megoldás:

```
#include <cctype>
…

int main(){
   string s;
   for (int i = 0; i < 10; i++){
      cin >> s;
      s[0] = toupper(s[0]);
         // nagybetűsítés, majd visszaírás
      cout << s << endl;
   }
   return 0;
}
```

Szövegkezelés

Szöveg műveletek

- A szöveget kezelő utasításokat egy adott szöveg típusú változóra kell meghívnunk **<*változónév*>.<*függvénynév*>()** formában a **string** fájl tartalmazza, ezért nem kell külön fájlt beillesztenünk a programba
- Jelentősebb műveletek:
  - adott karakter lekérdezése: **<*szöveg*>[<*index*>]**
  - szöveg összefűzése: **<*szöveg1*> + <*szöveg2*>**
  - szöveg hosszának lekérdezése: **<*szöveg*>.length()**
  - üres-e a szöveg: **<*szöveg*>.empty()**
  - szöveg törlése: **<*szöveg*>.erase()**
  - résszöveg lekérdezése: **<*szöveg*>.substr(<*kezdőindex*>,<*hossz*>)**

Szövegkezelés

Szöveg műveletek

- Jelentősebb műveletek:
  - karakter első előfordulásának helye (indexe, ha nem találja `string::npos` a visszatérési érték):
    `<szöveg>.find(<karakter>)`
  - karaktertömbbé konvertálás: **`<szöveg>.c_str()`**
  - szöveg hozzáfűzése:
    `<szöveg>.append(<új szövegrész>)`
  - új szövegrész beszúrása:
    `<szöveg>.insert(<index>, <új szövegrész>)`
  - szövegrész lecserélése:
    `<szöveg>.replace(<kezdőindex>, <hossz>, <új szövegrész>)`

Szövegkezelés

Szöveg műveletek

- Pl.:

```
string s;
    // s.empty() igaz lenne
string s1 = "árvíztűrő";
string s2 = "tükörfúrógép";
s = s1 + " " + s2;
    // s = "árvíztűrő tükörfúrógép" lesz
    // s.empty() hamis lenne
s.append("!");
    // s = "árvíztűrő tükörfúrógép!" lesz
s.replace(0,1,"Á");
    // s = "Árvíztűrő tükörfúrógép!" lesz
int length = s.length(); // length = 23 lesz
char ch = s[2];          // ch = 'v' lesz
```

Szövegkezelés

Szöveg műveletek

- Pl.:

```
string sub1 = s.substr(0,5);
    // sub1 = "Árvíz" lesz
int index1 = s.find('ő'); // index1 = 8 lesz
int index2 = s.find('r');
    // index2 = 1 lesz, mindig az elsőt találja meg
string sub2 = s.substr(0,s.find('ő'));
    // sub2 = "Árvíztűrő" lesz
string sub3 = s.substr(s.find("fú"),
                       s.length() - s.find("fú"));
    // megkeresi az "fú"-t, annak a helye a 15
    // a szöveg hossza 23, tehát venni fogja a
    // 8 hosszan a szöveget, sub3 = "fúrógép!" lesz
s.erase(); // s = "" lesz
```

Szövegkezelés

Példa

Feladat: Olvassunk be szavakat a billentyűzetről, és ha írjuk vissza az első négy karakterét a képernyőre, vagy ha rövidebb, akkor a teljes szót.

- elemenként dolgozzunk fel a szavakat, amíg *-hoz nem érünk
- használjunk utántesztelő ciklust
- először kérdezzük le a hosszt, majd ennek megfelelően írjuk ki a teljes szót, vagy az elejét

Specifikáció:

- bemenet: szavak (*szo*) sorozata
- kimenet: szavak első négy betűje, vagy a teljes szó

Példa

Megoldás:

```
…
int main(){
   string s;
   do{
      cin >> s;
      if (s.length() < 4) // ha rövidebb
         cout << s << endl;
      else // ha nem, akkor csak az eleje kell
         cout << s.substr(0,4) << endl;
   } while (s != "*");
   return 0;
}
```

Szövegkezelés

Sorok olvasása

- Ha a **>>** operátort használjuk beolvasásra, akkor szöveg esetén csak az első szót olvashatjuk be a bemenetről, néha azonban szükségünk van arra, hogy nem csak egy szót, hanem egy sorban többet be tudjuk olvasni
- Erre a célra szolgál a **getline(<*forrás*>,<*változó*>)** utasítás, amely a forrás folyamról beolvas egy sort a szöveg típusú változóba
  - nálunk a forrás a képernyő bemenete, ezért **getline(cin, <*szöveg változó*>)** formában fogjuk használni
  - pl.:

```
string s;
getline(cin, s); // s-be bekerül a teljes sor
```

Példa

Feladat: Olvassunk be sorokat a képernyőről, és számoljuk meg, hány kezdődik szóközzel.

- számlálást alkalmazunk, dolgozzunk az üres sorig
- használjunk előreolvasást
- lekérdezzük a sor első karakterét, és megnézzük, hogy szóköz-e

Specifikáció:

- bemenet: sorok (*sor*) sorozata
- kimenet: szóközzel kezdődő sorok száma (*c*)

Példa

Megoldás:

```
…
int main(){
   string sor;
   int c = 0;
   getline(cin, sor); // sor beolvasása
   while (sor != ""){
      if (sor[0] == ' ') c++; // első karakter
      getline(cin, sor); // sor beolvasása
   }
   cout << "Szóközzel kezdődő sorok száma: "
        << c << endl;
   return 0;
}
```

Szövegkezelés

Példa

Feladat: Írjunk be minden sorba egy nevet és egy telefonszámot, és írjunk ki őket formázottan, ”Név: név, Szám: szám” formában.

- elemenként dolgozzunk fel 10 sort a bementről
- tegyük fel, hogy csak keresztneveket írunk, és szóköz után jön a telefonszám, ekkor a szóköz előtt a név lesz, utána a telefonszám, ezért itt szét kell vágnunk a sort
- a **`find()`** segítségével megkapjuk a szóköz helyét, a **`substr()`** segítségével megvághatjuk a sort

Szövegkezelés

Példa

Specifikáció:

- bemenet: sorokban (*sor*) tárolt adatok
- kimenet: a sorok kiírása formázottan

Megoldás:

```
…
int main(){
    string sor, nev, szam;
    for (int i = 0; i < 10; i++){
       getline(cin, sor); // sor beolvasása
```

Példa

Megoldás:

```
        nev = sor.substr(0, sor.find(' '));
         // levágjuk a sor első részét, a szóközig
         szam = sor.substr(sor.find(' ') + 1,
                sor.length() - sor.find(' ') - 1);
         // megkeressük a szóköz utáni karaktert
         cout << "Név: " << nev << ", Szám: "
              << szam << endl;
         // formázott kiíratás
      }
      return 0;
  }
```

Szekvenciális fájlkezelés

Elvei

- 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 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, ezért még óvatosabbnak kell lennünk a kezelésükkel

Szekvenciális fájlkezelés

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

Szekvenciális fájlkezelés

Fájlok megnyitása

- A logikai és fizikai fájl társítását kétféleképpen végezhetjük:
  - a logikai fájlnév deklarációját követően a **<*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,
// innentől az f az adatok.txt fájlon dolgozik
```

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

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

Szekvenciális fájlkezelés

Fájlok elérése

- Fontos, hogy a paraméterben megadott fájlnév helyileg a generált futtatható fájl könyvtárában kell legyen
- Ha nem ott van, az elérési útját is meg kell adnunk (persze akkor is megadhatjuk az elérési utat, ha ugyanabban a könyvtárban dolgozunk)
  - az elérési utat az operációs rendszer elérési útjának megfelelően kell megadnunk
  - pl.:

```
f.open("c:\\doksik\\inf.dat");
        // Windows alatt (a \ védett karakter)
f.open("/home/groberto/inf.dat");
        // Linux alatt
```

Szekvenciális fájlkezelés

Fájlok elérése

- 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 szövegtípusban található egy olyan függvény, amely karaktertömbbé alakítja a szöveget, ezt használjuk: **`<`*`változónév`*`>.c_str()`**
  - pl.:

```
ifstream file;  // logikai fájl
string fname;   // egy string
cin >> fname;   // beolvassuk a stringet
file.open(fname.c_str());
// a beolvasott fájlnevet próbáljuk megnyitni
```

Szekvenciális fájlkezelés

Megnyitás ellenőrzés

- 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.:

```
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
}
```

Szekvenciális fájlkezelés

Megnyitás ellenőrzés

- 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
  - ehhez újra kell inicializálni a fájlt a **<*fájlnév*>.clear()** utasítással, 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()); // újbóli megnyitás
}
```

Szekvenciális fájlkezelés

Fájlok bezárása

- 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
  - 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.open("adat2.txt");
// adat2.txt megnyitása
```

Szekvenciális fájlkezelés

Beolvasás, kiírás

- 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();
```

Szekvenciális fájlkezelés

Fájl vége jel

- 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.:

```
ofstream f; int data;
f.open("adatok.txt");
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

Szekvenciális fájlkezelés

Megjegyzések

- Amire érdemes odafigyelni:
  - mielőtt elkezdünk beolvasni egy fájlból, nézzük meg, hogy sikerült-e megnyitni
  - a megnyitott fájlt mindig zárjuk be
  - ne olvassunk tovább, ha már elértük az EOF jelet, ha nem előreolvasást használunk és nem vagyunk biztosak a fájl tartalmában, akkor ellenőrizzük le, nem-e értünk 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 korábbi tartalma törlődni fog a megnyitáskor

Szekvenciális fájlkezelés

Példa

Feladat: Írjuk ki a képernyőre a **`szamok.txt`** fájlban tárolt számok összegét.

- bementi fájlt feldolgozzuk összegzés tételével
- bármennyi szám lehet a fájlban, ezért EOF jelig dolgozunk előreolvasással

Specifikáció:

- bemenet: fájlban (f) számok (szam) sorozata
- kimenet: számok összege (s)

Szekvenciális fájlkezelés

Példa

Megoldás:

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

int main(){
   int szam, s = 0;

   ifstream f("szamok.txt"); // megnyitás
   if (f.fail()){       // ha nem sikerült
      cout << "Rossz fájlnév!";
   }
   else {               // ha sikerült
```

Szekvenciális fájlkezelés

Példa

Megoldás:

```
        f >> szam;       // előreolvasás
        while (!f.eof()){ // amíg nincs vége
           s += szam;
           f >> szam;
        }
        f.close();       // fájl bezárása
        cout << "A számok összege: " << s;
    }
    return 0;
}
```

Szekvenciális fájlkezelés

Példa

Feladat: Olvassunk be egy fájlból szavakat, és írjuk ki egy kimeneti fájlba az ‘a’ betűt tartalmazó szavakat.

- használjunk elemenkénti feldolgozást, a **`find()`** függvénnyel nézzük meg, hogy van-e benne ‘a’ betű
- a fájlok neveit kérjük be a felhasználótól

Specifikáció:

- bemenet: szavak (*szo*) sorozata egy bemeneti fájlban (*bf*)
- kimenet: ‘a’ betűs szavak egy kimeneti fájlban (*kf*)

Szekvenciális fájlkezelés

Példa

Megoldás:

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

int main(){
   string fajl_nev, szo;
   ifstream bf; // bemenő fájl
   ofstream kf; // kimenő fájl

   cout << "Bemenő adatok fájlja: ";
   cin >> fajl_nev;
```

Szekvenciális fájlkezelés

Példa

Megoldás:

```
bf.open(fajl_nev.c_str()); // fájl megnyitás
while(bf.fail()){ // ha sikertelen
   cout << "Nem található a fájl!" << endl
        << "Bemenő adatok fájlja: ";
   cin >> fajl_nev; // újra bekérés

   bf.clear();
   bf.open(fajl_nev.c_str());
}

cout << "Kimenő adatok fájlja: ";
cin >> fajl_nev;
```

Szekvenciális fájlkezelés

Példa

Megoldás:

```
    kf.open(fajl_nev.c_str());
    while (bf >> szo){
       // beolvasással egybekötött ciklusfeltétel
       if (szo.find('a') < szo.length())
          // ha tartalmaz a-t, kiírjuk
          kf << szo << endl;
    }

    bf.close(); // fájlok bezárása
    kf.close();
    return 0;
}
```

Szekvenciális fájlkezelés

Sorok olvasása

- 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("input.txt");
string s;
getline(bemenet, s); // előreolvasás
while (!f.eof()) // amíg nincs vége a fájlnak
{
   // ... műveletek s-sel
   getline(bemenet, s);
   // következő sor beolvasása
}
```

Szekvenciális fájlkezelés

Példa

Feladat: Határozzuk meg egy fájlban tárolt soroknak a hosszát, és írjuk a képernyőre.

- a fájlnevet kérjük be a felhasználótól, sikertelen esetben lépjünk ki a programból

Specifikáció:

- bemenet: sorok (*sor*) egy bemeneti fájlban (*f*)
- kimenet: a sorok karaktereinek száma

Szekvenciális fájlkezelés

Példa

Megoldás:

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

int main(){
   string fajl_nev, sor; ifstream f;
   cout << "Bemenő adatok fájlja: ";
   cin >> fajl_nev; f.open(fajl_nev.c_str());
   if (f.fail()){ // ha sikertelen
      cout << "Nem található a fájl!" << endl;
      return 1;  // kilépés
   }
```

Szekvenciális fájlkezelés

Példa

Megoldás:

```
    getline(f, sor);   // sor beolvasása
    while(!f.eof()){
       cout << sor.length() << endl;
       getline(f, sor);
    }
    f.close(); return 0;
}
```

Szekvenciális fájlkezelés

Sorok olvasása

- A sorbeolvasás egy speciális változata, amikor nem a teljes sort olvassuk be, csak egy részét
- Lehetőségünk van egy adott karakterig olvasni a sort:
  `getline(<`*`logikai fájlnév`*`>, <`*`változó`*`>, <`*`karakter`*`>)`
  - ekkor a harmadik helyen megadjuk azt a karaktert, aminek első előfordulásánál megáll az olvasás
  - a határoló karaktert az utasítás eldobja, tehát nem kerül be a változóba, ugyanakkor a következő olvasás utána fog kezdődni
  - a karakter lehet vezérlőkarakter is (pl. tabulátor)
  - kiegészítve az előző változattal, több lépésben olvashatjuk be a teljes sort

Szekvenciális fájlkezelés

Példa

Feladat: Egy **`telkonyv.txt`** fájlban a következő formátumban vannak a sorok: vezetéknév keresztnév,cím,telefonszám. Írjuk ki az adatokat az **`uj_telkonyv.txt`** fájlba a következő formátumba: keresztnév vezetéknév,telefonszám,cím.

- feltételezzük, hogy a fájl létezik, és a formátuma helyes
- négy lépésben olvassuk be a sort négy szöveg változóba
- előreolvasást használunk

Specifikáció:

- bemenet: a szöveg (*veznev*, *kernev*, *cim*, *szam*) egy bemeneti fájlban (*bf*)
- kimenet: aformázott szöveg a kimeneti fájlban (*kf*)

Szekvenciális fájlkezelés

Példa

Megoldás:

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

int main(){
   string veznev, kernev, cim, szam;

   ifstream bf("telkonyv.txt");
   ofstream kf("uj_telkonyv.txt");
```

Szekvenciális fájlkezelés

Példa

Megoldás:

```
// sor beolvasása 4 lépésben:
getline(bf, veznev, ' ');
// olvasás az első szóközig, a szóköz karakter
// elveszik
getline(bf, kernev, ',');
// olvasás a szóköztől a vesszőig, a vessző
// elveszik
getline(bf, cim, ',');
getline(bf, szam); // olvasás a végéig
```

Szekvenciális fájlkezelés

Példa

Megoldás:

```
    while(!bf.eof()){
       kf << kernev << " " << veznev << ","
          << szam << "," << cim << endl;
       // következő sor beolvasása:
       getline(f, veznev, ' ');
       getline(f, kernev, ',');
       getline(f, cim, ',');
       getline(f, szam);
    }
    bf.close(); kf.close();
    return 0;
}
```

II. Vegyes feladatok:

14. a) Egy tetszőleges szövegben keresd meg az "alma" szó első előfordulását, és cseréld le a "körte" szóra.
    b) Az "alma" és "körte" szavak helyett tetszőleges szót lehessen megadni.
    c) Az összes előfordulást cseréld le.
24. Egy fájlból beolvasott tetszőleges szöveget módosíts úgy egy másik fájlba, hogy a sorai elé írod az adott sor sorszámát.

III. Programozási tételek:

9. b) Tetszőleges, fájlból beolvasott egész számsor átlagát add meg.

28. Egy fájlban tárolt, több soros szövegben add meg, hányszor található meg benne egy tetszőleges szó.

38. Olvass egy fájlból mozifeliratokat sub formátumban.
    a) (*) Melyik sor látszik legtöbb ideig a képernyőn?
    c) (*) Hol hadartak a leggyorsabban?