Objektum orientált programozás C++ nyelven 8. Fájlok kezelése

PEKÁRDY MILÁN – PANNON EGYETEM

PEKARDY@DCS.UNI-PANNON.HU

Fájlok kezelése

- C++-ban karakterek fájlba írására, illetve fájlból beolvasására a következő osztályokat használhatjuk:
  - ofstream: folyam a fájlba íráshoz,
  - ifstream: folyam a fájlból való beolvasáshoz,
  - fstream: folyam az íráshoz/olvasáshoz.
- A fenti folyamok az istream/istream osztályokból származnak, hasonlóan mint a cin/cout folyamok, ennek megfelelően a működésük is hasonló, azzal a különbséggel, hogy a fájl folyamok esetén szükséges egy fizikai fájl hozzárendelése a folyamokhoz.

Fájlok kezelése

```cpp
// basic file operations
#include <iostream>
#include <fstream>
using namespace std;

int main () {
  ofstream myfile;
  myfile.open ("example.txt");
  myfile << "Writing this to a file.\n";
  myfile.close();
  return 0;
}
```

Fájl megnyitása

- Fájl kezeléséhez első lépésben a fájl folyamot hozzá kell rendelni egy fizikai fájlhoz, azaz meg kell nyitni a fájlt a program számára,
- A megnyitott fájlt a hozzárendelt folyam objektum reprezentálja. A folyamon végzett műveletek érvényesülnek a hozzárendelt fizikai fájlon,
- Fájl megnyitásához az open(filename, mode) metódust használjuk:
  - filename: a megnyitandó fájl neve (abszolút elérési út vagy relatív a futtatható állományhoz képest),
  - mode: fájlmegnyitás módja (opcionális).

Fájl megnyitása – megnyitási módok

Mód	Leírás
ios::in	megnyitás olvasásra
ios::out	megnyitás írásra
ios:binary	megnyitás bináris módban
ios::ate	a fájlban a kezdő pozíciót a fájl végére állítja (alapértelmezetten a fájl elején vagyunk)
ios::app	minden írási művelet a fájl végén történik, azaz hozzáfűzünk a meglévő tartalomhoz
ios:trunc	ha a fájl már létezett és meg van nyitva írási műveletekre, akkor a meglévő tartalom törlődik és az új tartalomra cserélődik

Fájl megnyitása – megnyitási módok

- A megnyitási módok tetszőlegesen kombinálhatók a bitenkénti VAGY (|) operátor segítségével:

```
ofstream myfile;
myfile.open ("example.bin", ios::out | ios::app | ios::binary);
```

- A fájl folyamok esetén az alábbi alapértelmezett megnyitási módok állítódnak be:

class	default mode parameter
ofstream	ios::out
ifstream	ios::in
fstream	ios::in | ios::out

- ifstream/ofstream esetén az alapértelmezett módok akkor is beállítódnak ha explicit adunk más egyéb módokat is, fstream esetén viszont a megadott módok felülírják az alapértelmezett módokat.

Fájlok megnyitása – megnyitási módok

- Bináris módban megnyitott fájlok esetén az írási/olvasási műveletek formátum nélkül hajtódnak végre: a bináris adatok egyszerűen kiírásra/beolvasásra kerülnek,
- Nem bináris fájlok esetén (pl. szöveges fájlok) bizonyos formátum átalakítások automatikusan előfordulhatnak: speciális karakterek (új sor, kurzor sor elejére, stb.) formázása,
- A fájlok megnyitásának megkönnyítésére a fájl folyamok konstruktorainak bizonyos verziói automatikusan megnyitják a fájlt: a konstruktor a fájl nevét és az opcionális megnyitási módokat várja:

```
ofstream myfile ("example.bin", ios::out | ios::app | ios::binary);
```

- A fájl folyamok esetén mindig érdemes ellenőrizni, hogy sikerült-e megnyitni a fájlt az is_open metódus hívásával. Sikeres megnyitás esetén logikai igaz értékkel tér vissza, ellenkező esetben hamissal:

```
if (myfile.is_open()) { /* ok, proceed with output */ }
```

Fájlok bezására

- Az írási/olvasási folyamatok végeztével szükséges a fájl bezárása, hogy az operációs rendszer értesítve legyen a fájl erőforrások felszabadításáról,
- A bezárást explicit módon a close függvény hívásával végezzük, ilyenkor az esetlegesen pufferelt módosítások megtörténnek (flush) majd bezárja a fájlt,
- Bezárás után a folyam objektum újra hasznosítható másik fájl kezelésére,
- Ha az objektum felszabadításra kerül miközben a fájl még meg van nyitva, akkor a destruktor automatikusan meghívja close függvényt.

```
myfile.close();
```

Szövegfájlok kezelése

- Szöveg fájlok esetén az ios::binary megnyitási mód nincs beállítva,
- Az ilyen folyamok esetén szöveges adatokat kezelünk és a kapcsolódó fizikai fájlok tárolják ezeket a szövegeket,
- Az írási művelet a cout-nál megmutatott móddal analóg történik:

```
// writing on a text file
#include <iostream>
#include <fstream>
using namespace std;

int main () {
  ofstream myfile ("example.txt");
  if (myfile.is_open())
  {
    myfile << "This is a line.\n";
    myfile << "This is another line.\n";
    myfile.close();
  }
  else cout << "Unable to open file";
  return 0;
}
```

Szövegfájlok kezelése

- Beolvasás a cin-nél látott módon történik,
- A fájlban lévő sorok beolvasására a getline függvényt használjuk, ami a folyam objektumra mutató referenciával tér vissza, ami logikai igazzá értékelődik ki ha lehet még beolvasni a fájlból, logikai hamissá ha már nincs mit beolvasni vagy valami hiba történt:

```
// reading a text file
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

int main () {
  string line;
  ifstream myfile ("example.txt");
  if (myfile.is_open())
  {
    while ( getline (myfile,line) )
    {
      cout << line << '\n';
    }
    myfile.close();
  }

  else cout << "Unable to open file";

  return 0;
}
```

Állapotjelölők

- A fájl folyam különböző állapotainak ellenőrzésére az alábbi függvényeket alkalmazhatjuk:

bad()	igazzal tér vissza ha egy írási/olvasási művelet sikertelen, pl.: olyan fájlba szeretnénk írni, ami nem olvasásra lett megnyitva, vagy nincs elegendő hely az írás elvégzésére, stb.
fail()	igazzal tér vissza azokban az esetekben, amikor a bad(), illetve akkor is ha formázási hiba történik, pl.: egy betű karakter jön a folyamból, de mi egy számot szeretnénk beolvasni
eof()	igazzal tár vissza ha a beolvasásra megnyitott fájl végére értünk
good()	általános állapotjelölő függvény: logikai hamissal tér vissza, akkor amikor az összes fenti függvény igazzal tér vissza (a good és a bad függvények nem teljesen ellentétei egymásnak, a good több jelölőt ellenőriz)

- Mindegyik függvény logikai értékkel tér vissza,
- A clear() tagfüggvény segítségével az összes állapotjelölő visszaállítható az alap értékekere

Folyam pozíciók

- Minden fájl folyam nyilvántart legalább egy belső pozíciót:
  - ifstream: egy get pozíciót tart nyilván, ami a beolvasás helyét jelzi a következő olvasás műveletnél,
  - ofstream: egy put pozíciót tart nyilván, ami a beírás helyét jelzi a következő írási műveletnél,
  - fstream: get és put pozíciót is nyilvántart,
- A belső pozíciók lekérdezhetők, illetve beállíthatók a következő függvényekkel:
  - tellg(), tellp(): bemeneti paraméterük nincs, visszatérési értékük streampos típus és az aktuális get/put pozíciót adják vissza,
  - seekg(), seekp(): a get/put pozíciók beállítására szolgálnak, bemeneti paraméter lehet egy streampos típusú változó, ami a változó által reprezentált abszolút pozícióra állítja a jelölőt, illetve megadható egy offset és egy irány (direction), ami a megadott pozícióhoz képest relatív állítja a jelölőt a megadott offsettel. Az irány típusa seekdir, az offset típusa pedig streamoff. A direction a következő enumerált értékeket veheti fel:

ios::beg	offset counted from the beginning of the stream
ios::cur	offset counted from the current position
ios::end	offset counted from the end of the stream

Folyam pozíciók

- streampos típus puffer és fájl pozíciók kezelésére használhatjuk,
- a pozíciókat kivonhatjuk egymásból, így pl.: megkaphatjuk egy fájl méretét,
- streampos/streamoff típusok a stream osztályban is definiálva vannak:

```cpp
// obtaining file size
#include <iostream>
#include <fstream>
using namespace std;

int main () {
  streampos begin,end;
  ifstream myfile ("example.bin", ios::binary);
  begin = myfile.tellg();
  myfile.seekg (0, ios::end);
  end = myfile.tellg();
  myfile.close();
  cout << "size is: " << (end-begin) << " bytes.\n";
  return 0;
}
```

Type	Member type	Description
streampos	ios::pos_type	Defined as fpos<mbstate_t>. It can be converted to/from streamoff and can be added or subtracted values of these types.
streamoff	ios::off_type	It is an alias of one of the fundamental integral types (such as int or long long).

Bináris fájlok

- Bináris fájlok esetén a olvasó/író operátorok (>>, <<), illetve a getline függvény nem hatékonyak, mivel nincs szükség az adatok formázására (nem szövegként tároljuk az adatokat),
- Bináris adatok kezeléséhez használjuk a read/write függvényeket. A write az ostream, a read pedig az istream osztály tagfüggvénye,
- Mindegyik első paraméterben egy char* típust vár, ami egy bájt tömbnek azt a címét reprezentálja ahol a beolvasott adatokat tároljuk, illetve ahonnan a kiírandó adatokat vesszük.

Bináris fájlok

- A példában a memóriába beolvassuk egy bináris fájl teljes tartalmát:
  - megnyitjuk a fájlt úgy, hogy a get jelölő a fájl végére mutasson,
  - lekérjük a fájl méretét a tellg() függvénnyel,
  - létrehozunk egy a méretnek megfelelő nagyságú char tömböt,
  - a read függvénnyel beolvassuk a fájlt a tömbbe,
  - felszabadítjuk az erőforrásokat.

```
// reading an entire binary file
#include <iostream>
#include <fstream>
using namespace std;

int main () {
  streampos size;
  char * memblock;

  ifstream file ("example.bin", ios::in|ios::binary|ios::ate);
  if (file.is_open())
  {
    size = file.tellg();
    memblock = new char [size];
    file.seekg (0, ios::beg);
    file.read (memblock, size);
    file.close();

    cout << "the entire file content is in memory";

    delete[] memblock;
  }
  else cout << "Unable to open file";
  return 0;
}
```

Bináris fájlok

```cpp
// Copy a file
#include <fstream>      // std::ifstream, std::ofstream

int main () {
  std::ifstream infile ("test.txt",std::ifstream::binary);
  std::ofstream outfile ("new.txt",std::ofstream::binary);

  // get size of file
  infile.seekg (0,infile.end);
  long size = infile.tellg();
  infile.seekg (0);

  // allocate memory for file content
  char* buffer = new char[size];

  // read content of infile
  infile.read (buffer,size);

  // write to outfile
  outfile.write (buffer,size);

  // release dynamically-allocated memory
  delete[] buffer;

  outfile.close();
  infile.close();
  return 0;
}
```

Pufferek és szinkronizáció

- A fájl folyamokhoz tartozik általában egy belső puffer is, aminek a típusa streambuf,
- A puffer egy köztes réteg a folyam és a fizikai fájl között. Pl.: fájlba íráskor a pufferbe szúrja be a folyama az adatokat és csak egy későbbi időpontban írja ki őket a fájlba (az operációs rendszer további puffer rétegeket is használhat),
- Amikor a puffert kiürítjük (flush), akkor a tárolt tartalom kiírásra kerül a fizikai fájlba, ez a művelet a szinkronizáció és a következő esetekben történik:
  - fájl bezárásakor: bezárás előtt minden puffer szinkronizálódik a fizikai fájllal,
  - puffer telítődésekor: a puffereknek meghatározott méretei vannak, amikor megtelik egy puffer, akkor automatikusan szinkronizáció történik,
  - explicit (manipulátorokkal): bizonyos manipulátorok (flush, endl) alkalmazása esetén a folyamokon automatikus szinkronizáció történik,
  - explicit (sync() tagfüggvény): a folyam sync() függvényének hívásakor explicit szinkronizáció történik. A függvény -1-el tér vissza ha nincs puffer rendelve a folyamhoz, illetve ha valami hiba történik egyébként sikeres szinkronizáció esetén 0 a visszatérési érték