Fişiere

Noțiuni generale

Intrările şi ieşirile în limbajul C++ sunt implementate cu ajutorul fluxurilor (*stream*). Fluxurile sunt obiecte care transportă şi formatează şiruri de bytes. Fluxurile pot fi unidirecționale (de intrare sau de ieşire) sau bidirecționale (permit şi intrări şi ieşiri).

Clasele care reprezintă fluxuri asociate fişierelor sunt definite în biblioteca *fstream*; aceste clase sunt:

- *ifstream*: flux unidirecțional care permite citirea de date dintr-un fişier
- *ofstream*: flux unidirecțional care permite scrierea de date dintr-un fişier
- *fstream*: flux bidirecțional care permite citiri şi scrieri în/din fişier

Operații de bază

Deschiderea fişierelor

Deschiderea fişierelor se poate face în două moduri: prin constructor sau prin apelarea funcției membru *open*. Ambele variante primesc ca parametri numele fişierului şi opțiunile de deschidere:

Varianta folosind constructorul:

fstream fisier("nume_fisier", optiuni);

Varianta folosind funcția *open*:

fstream fisier;
fisier.open("nume_fisier", optiuni);

Partea cu opțiunile poate lipsi. Pentru clasele *istream* şi *ostream*, operațiunea de deschidere se efectuează similar. Opțiunile la deschiderea fişierelor se dau prin intermediul următoarelor constante:

Constantă	Efect
`ios::in`	Deschide un fişier pentru citire. Opțiunea este folosită şi pentru a evita suprascrierea fişierului în cazul în care acesta există.
`ios::out`	Deschide un fişier pentru scriere.
`ios::app`	Deschide fişierul pentru adăugare. Toate scrierile se vor face la sfârşitul fişierului.
`ios::ate`	Deschide fişierul şi se poziționează la sfârşit.
`ios::trunc`	Şterge conținutul fişierului la deschidere (dacă acesta există).
`ios::binary`	Deschide fişierul în mod binar (implicit este text).

Opțiunile se pot combina folosind operatorul „|”.În cazul în care opțiunile lipsesc, valorile implicite sunt:

Clasa	Opțiuni

ifstream	`ios::in`
ofstream	`ios::out`
fstream	`ios::in | ios::out`

Exemple de utilizare:

```
// creaza obiectul fisier pentru citire/scriere
// si deschide fisierul "test.txt" folosind
// constructorul si optiunile implicite
fstream fisier("test.txt");

// deschide fisierul pentru citire in mod binar
ifstream fcitire;
fcitire.open("date.dat", ios::in | ios::binary);

// deschide fisierul "studenti.txt" in mod text pentru
// scriere la sfarsitul fisierului
ofstream fscriere("studenti.txt", ios::out | ios::app);
```

Verificarea faptului că un fişier este deschis se poate face folosind funcția membru *is_open()*.

Închiderea fişierelor

Fişierele deschise folosind clasele din *fstream* sunt închise automat de către destructorul clasei. Închiderea fişierului se poate face explicit prin apelul funcției membru *close*; după apelul funcției, obiectul poate fi refolosit pentru a accesa alt obiect. Exemplu:

```
void main()
{
	// deschidem fisierul binar "studenti.dat"
	fstream fisier("studenti.dat", ios::in | ios::binary);

	// operatii ...

	// inchidem fisierul
	fisier.close();

	// deschidem fisierul "lista.txt"
	fisier.open("lista.txt", ios::out);

	// operatii ...

} // fisierul "lista.txt" este inchis automat la
  // distrugerea obiectului "fisier"
```

Detectarea erorilor

Detectarea erorilor apărute se poate face prin intermediul unor funcții membre ale claselor:

Funcția	Descriere
`bad`	Întoarce **true** dacă a apărut o eroare fatală; în acest caz, fluxul nu mai este utilizabil.
`fail` sau `operator!`	Ultima operație de I/E a eşuat (eroare de conversie, sfârşitul şirului, fişierul nu poate fi deschis…). Fluxul poate fi utilizat în continuare după apelarea funcției membru *clear()*.
`good`	Întoarce **true** dacă nu a apărut nici o eroare şi fişierul nu este la sfârşit.

`eof`	Întoarce **true** dacă fişierul este la sfârşit.

În cazul în care se doreşte ştergerea indicatorilor de eroare se poate folosi funcția **clear**.

Exemplu de utilizare:

```
// deschidere fisier
fstream fisier("studenti.dat", ios::in | ios::binary);

// verificare
if (!fisier)
{
	cerr << "Eroare la deschiderea fisierului!" << endl;
	return;
}
```

Fişiere text şi formatarea datelor

Citirea şi scrierea din/în fişiere text se face folosind operatorii „<<” şi „>>” (ca şi în cazul obiectelor *cin* şi *cout*).

Exemplu:

```
#include <iostream>		// obiectele pentru lucrul cu consola
#include <fstream>		// obiectele pentru lucrul cu fisiere
using namespace std;

// Afiseaza vectorul pe ecran
void AfisareVector(int vector[], int n)
{
	cout << "Vector cu " << n << " elemente:" << endl;
	for (int i = 0; i < n; i++)
		cout << vector[i] << " ";
	cout << endl;
}

// Salveaza vectorul intr-un fisier in formatul:
// Linia 1: numarul de elemente
// Linia 2: elementele separate prin spatiu
void SalvareVector(int vector[], int n, char* numeFisier)
{
	// deschidem fisierul text pentru scriere
	// (si il suprascriem daca este cazul)
	ofstream fisier(numeFisier, ios::out | ios::trunc);

	// daca fisierul a fost deschis
	if (fisier)
	{
		// scriem numarul de elemente pe prima linie
		fisier << n << endl;

		// scriem elementele separate prin spatiu
		for (int i = 0; i < n; i++)
			fisier << vector[i] << " ";
	}
	else
	{
		// altfel efisam un mesaj de eroare
		cerr << "EROARE: Fisierul '" << numeFisier
			<< "' nu a putut fi deschis pentru scriere." << endl;
	}
```

```
	// fisierul este inchis automat de constructorul
	// clasei ofstream
}

// Incarca din fisier un vector salvat de functia
// SalvareVector. Presupune ca vectorul primit ca
// parametru este alocat.
void IncarcareVector(int vector[], int& n, char* numeFisier)
{
	// deschidem fisierul pentru citire
	ifstream fisier(numeFisier);

	// daca fisierul a fost deschis
	if (fisier)
	{
		// citim numarul de elemente din fisier
		fisier >> n;

		// citim elementele vectorului
		for (int i = 0; i < n; i++)
			fisier >> vector[i];
	}
	else
	{
		// afisam un mesaj de eroare
		cerr << "EROARE: Fisierul '" << numeFisier
			<< "' nu a putut fi deschis pentru citire." << endl;
	}

	// fisierul este inchis automat de constructorul
	// clasei ifstream
}

void main()
{
	// construim un vetor de test
	int vector[] = {45, 34, 234, 3, 66};
	int n = 5;

	// afisam vectorul pe ecran
	AfisareVector(vector, n);

	// salvam vectorul in fisier
	SalvareVector(vector, n, "vector.txt");

	// construim un nou vector
	int vector2[5], n2;

	// citim datele din fisier in noul vector
	IncarcareVector(vector2, n2, "vector.txt");

	// afisam noul vector
	AfisareVector(vector2, n2);
}
```

În afară de cei doi operatori se mai pot folosi şi următoarele funcţii pentru lucrul cu fişiere text:

Funcția	Descriere
`getline(char*sir,int n, char delim='\n')`	Citeşte din fişier până se ajunge la numărul de caractere specificate sau până se întâlneşte delimitatorul.

`get()`	Citeşte un caracter din fişier.
`peek()`	Întoarce următorul caracter din fişier fără să mute indicatorul de poziționare..
`put(char)`	Scrie un caracter în fişier.

Funcția *getline* este utilă în special în cazul în care trebuie citite şiruri de caractere care conțin spații (de exemplu nume de persoane):

```
#include <iostream>        // obiectele pentru lucrul cu consola
#include <fstream>         // obiectele pentru lucrul cu fisiere
using namespace std;

// numarul maxim de caractere pentru un nume
const int DIM_MAX_NUME = 1024;

// numarul maxim de persoane din lista
const int NR_MAX_NUME = 1024;

// Citeste o lista de persoane dintr-un fisier
// (cate un nume pe fiecare linie) si intoarce
// un vector de siruri de caractere. Vectorul se
// presupune neinitializat.
void CitireListaPersoane(char**& lista, int&n, char* numeFisier)
{
	// deschidem fisierul pentru citire
	ifstream fisier(numeFisier);

	// daca fisierul a fost deschis
	if (fisier)
	{
		// alocam spatiu pentru stocarea temporara a numelui
		char *numeTemp = new char[DIM_MAX_NUME];

		// alocam spatiu pentru stocarea temporara a vectorului
		char **listaTemp = new char*[NR_MAX_NUME];

		n = 0;
		while(!fisier.eof())
		{
			// citim numele din fisier; se foloseste separatorul
			// implicit ('\n');
			fisier.getline(numeTemp, DIM_MAX_NUME);

			// alocam spatiul necesar si il copiem in vector
			int dimNume = (int)strlen(numeTemp);
			listaTemp[n] = new char[dimNume + 1];
			strcpy(listaTemp[n], numeTemp);

			n++;   // incrementam numarul de persoane
		}

		// copiem pointerii din vectorul temporar in rezultat
		lista = new char*[n];
		for (int i = 0; i < n; i++)
			lista[i] = listaTemp[i];

		// dealocam spatiul temporar
		delete [] numeTemp;
		delete [] listaTemp;
	}
	else
	{
		// afisam un mesaj de eroare
		cerr << "EROARE: Fisierul '" << numeFisier << "' nu a putut fi
deschis pentru citire." << endl;
```

```
	}
}

void main()
{
	// declaram variabilele necesare
	char **listaPersoane;
	int n;

	// citim lista de persoane din fisier
	CitireListaPersoane(listaPersoane, n, "persoane.txt");

	// afisam lista de persoane
	for (int i = 0; i < n; i++)
		cout << listaPersoane[i] << endl;
}
```

Fişiere binare

Pentru manipularea fişierelor binare se folosesc următoarele funcţii:

Funcţia	Descriere
`read(char*,int)`	Citeşte numărul maxim de octeţi specificaţi şi îi salvează în memorie la adresa indicată.
`gcount()`	Întoarce numărul de octeţi citiţi de ultima operaţie.
`write(char*,int)`	Scrie în fişier zona de memorie indicată prin adresă şi număr de elemente.
`tellg()/tellp()`	Întoarce poziţia curentă în fişier pentru *ifstream/ofstream*.
`seekg(int, ios::beg/cur/end)/seekp(int, ios::beg/cur/end)`	Setează poziţia curentă în fişier faţă de început/poziţia curentă/sfârşit pentru *ifstream/ofstream*.

Exemplu de utilizare – funcţie pentru copierea de fişiere:

```
void CopiereFisiere(char* numeSursa, char* numeDestinatie)
{
	// deschidem fisierele in modul binar
	ifstream sursa(numeSursa, ios::in | ios::binary);
	ofstream destinatie(numeDestinatie,
		ios::out | ios::trunc | ios::binary);

	// verificam daca fisierele sunt deschise corect
	if (!sursa || !destinatie)
	{
		cerr << "EROARE la deschiderea fisierelor." << endl;
		return;
	}

	// declaram un buffer de 1k pentru copierea fisierelor
	const int DIM_BUFFER = 1024;
	int buffer[DIM_BUFFER];

	// cat timp mai avem de copiat
	while (!sursa.eof())
```

```
	{
		// citim un bloc din fisierul sursa
		sursa.read((char*)buffer, DIM_BUFFER);

		// verificam cat a fost citit efectiv
		int nrOctetiCititi = sursa.gcount();

		// scriem in fisierul destinatie
		destinatie.write((char*)buffer, nrOctetiCititi);
	}

	// fisierele sunt inchise automat de catre destructori
}
```

Supraîncărcarea operatorilor

Supraîncărcarea operatorilor se face la fel ca în cazul operatorilor de I/E pentru consolă, cu mențiunea că se vor folosi clasele *ifstream* şi *ofstream* în locul claselor *istream* respectiv *ostream*.

Exemplu de supraîncărcare pentru clasa Persoana (pentru fişiere binare):

```
#include <iostream>		// obiectele pentru lucrul cu consola
#include <fstream>		// obiectele pentru lucrul cu fisiere
using namespace std;

// numarul maxim de caractere pentru un nume
const int DIM_MAX_NUME = 200;

// Clasa Persoana: Contine datele referitoare la o persoana.
class Persoana
{
public:

	// constructor
	Persoana(int cod = 0, char* nume = "Anonim")
	{
		// copiem datele primite ca parametri
		_cod = cod;
		strcpy(_nume, nume);
	}

	// functii de acces
	int GetCod()		const { return _cod;  }
	const char* GetNume() const { return _nume; }

private:
	// date membre
	int _cod;
	char _nume[DIM_MAX_NUME];
};

// Operatori de I/E pentru fisiere binare

// Salvam datele in fisier in formatul: Cod|NrCaractereNume|Nume
ofstream& operator << (ofstream& out, const Persoana& persoana)
{
	// obtinem codul persoanei
	int cod = persoana.GetCod();

	// obtinem numarul de caractere din nume
	int dimNume = (int)strlen(persoana.GetNume());

	// scriem codul in fisier
	out.write((char*)&cod, sizeof(cod));
```

```
	// scriem numarul de caractere in fisier
	out.write((char*)&dimNume, sizeof(dimNume));

	// scriem numele in fisier
	out.write(persoana.GetNume(), dimNume);

	return out;
}

// Salvam datele din fisier in formatul: Cod|NrCaractereNume|Nume
ifstream& operator >> (ifstream& in, Persoana& persoana)
{
	// citim codul, numarul de caractere din fisier
	int cod, dimNume;
	in.read((char*)&cod, sizeof(int));
	in.read((char*)&dimNume, sizeof(dimNume));

	// alocam spatiu pentru nume si il citim din fisier
	char *nume = new char[dimNume + 1];
	in.read(nume, dimNume);

	// adaugam terminatorul de sir
	nume[dimNume] = '\0';

	// punem datele in obiect
	persoana = Persoana(cod, nume);

	// dealocam spatiul temporar pentru nume
	delete [] nume;

	return in;
}

// Operatori de I/E pentru consola

// Operator pentru afisarea pe monitor
ostream& operator << (ostream& out, const Persoana& persoana)
{
	// afisam datele persoanei
	out << persoana.GetCod() << " " << persoana.GetNume() << endl;
	return out;
}

// Operator pentru citirea de la consola
istream& operator >> (istream& in, Persoana& persoana)
{
	// citim codul
	int cod;
	cout << "Cod:";
	cin >> cod;

	// citim numele
	char *nume = new char[DIM_MAX_NUME];
	cout << "Nume:";
	in >> ws; // eliminam spatiile de dinainte
	in.getline(nume, DIM_MAX_NUME);

	// construim obiectul persoana pe baza datelor citite
	persoana = Persoana(cod, nume);

	// dealocam spatiul temporar pentru nume
	delete [] nume;

	return in;
}

void main()
{
```

```
        // cream un obiect Persoana si citim datele de la tastatura
        Persoana pers;
        cin >> pers;

        // salvam obiectul intr-un fisier binar folosind operatorul
        ofstream fisOut("persoana.bin", ios::out | ios::trunc | ios::binary);
        fisOut << pers;
        fisOut.close();

        // construim un nou obiect Persoana si citim datele din fisierul binar
        Persoana pers2;
        ifstream fisIn("persoana.bin", ios::in | ios::binary);
        fisIn >> pers2;

        // afisam datele citite din fisier
        cout << pers2;
}
```

Supraîncărcarea pentru fişiere text se face la fel ca pentru consolă.