Programare II
Programare Orientată Obiect

Curs 12

Curs anterior

- Stanard Template Library
  - Containere asociative
  - Algoritmi
  - Clasa string

Curs curent

- Fisiere

Procesarea fișierelor

- Fișier
  - Folosite pentru persistarea datelor
    - Stocarea permanentă a unor volume mari de date
  - Stocate pe device-uri secundare de stocare
    - discuri magnetice
    - discuri optice
    - casete

Procesarea fișierelor

- Tipuri de fișiere
  - Binare
    - stochează datele sub format binar, 0 sau 1
    - cel mai mic mod de stocare suportat
  - Text
    - stochează datele ca o secvență de caractere
    - pot fi citite în editoare

Ierarhia datelor

- Bit - cea mai mică unitate
  - valoare 0 sau 1
- Byte - 8 biţi
  - folosite pentru a stoca caractere
    - char
    - wchar_t - pentru caractere unicode
- Câmp - un grup de caractere care au un înţeles
  - de exemplu numele unei persoane

Ierarhia datelor

- Înregistrări - un grup de câmpuri relaţionate
  - reprezentate de structuri sau clase
  - Exemple
    - un sistem de plăţi, informaţii despre o persoană

- Fişiere
  - compuse dintr-un grup de înregistrări relaţionate
  - exemplu
    - un sistem de plaţi care conţine o înregistrare pentru fiecare angajat
  - pot fi stocate în multe feluri
    - fişiere secvenţiale
    - exemple XML, JSON

- Baze de date
  - compuse dintr-un grup de fişiere relaţionate
  - gestionate de programe de gestionare a sistemelor de baze de date (DataBaseManagementSystems)

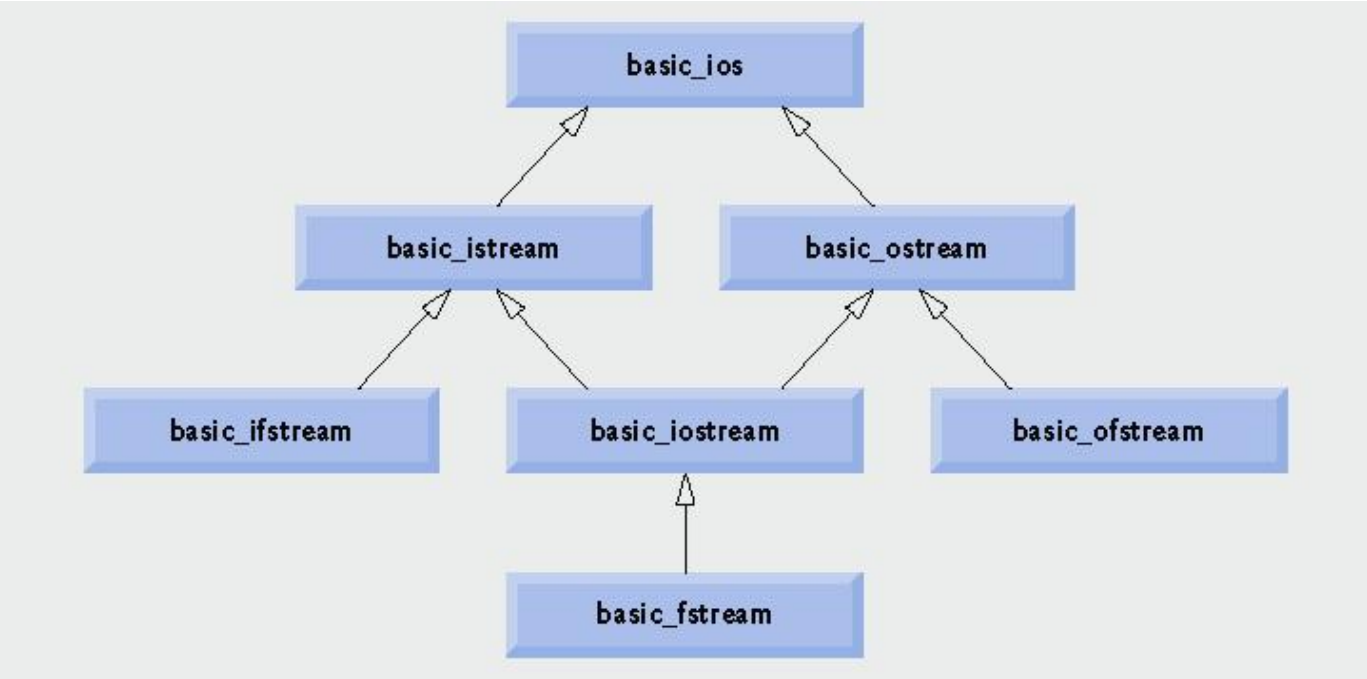
Fișiere și streamuri

- Ce este un fișier?
  - Este o colecție de date stocate
- Ce este un stream?
  - Este o conecxiune de la o sursă de date la o destinație de date
- C++ vede fișierele ca o secvență de bytes
  - se termină cu un marker de sfârșit de fișier (end-of-file)
- Comunicare dintre programe și fișiere se face prin intermediul streamurilor
  - fișiere header
    - iostream
      - cin/cout
    - fstream

Fișiere și streamuri

- Comunicare dintre programe și fișiere se face prin intermediul streamurilor
  - fișiere header
    - iostream
      - cin/cout
    - fstream
      - templaturi de clase
        - basic_ifstream (input)
        - basic_ofstream (output)
        - basic_fstream (I/O)
      - tipuri specializate pentru lucrul cu, caractere
        - ifstream (char input)
        - ofstream (char output)
        - fstream (char I/O)

Fișiere și streamuri



Crearea unui fișier secvențial

- C++ nu impune nici o structură a fişierelor
  - conceptul de ”înregistrare” trebuie implementat de programator

- Deschiderea fişierelor
  - tipuri de fişiere
    - ifstream (input only)
    - ofstream (output only)
    - fstream (I/O)
  - constructorul are ca parametrii numele fişierului şi mod în care se deschide
    - v1
      - ofstream outClientFile("filename", fileOpenMode);
    - v2
      - ofstream outClientFile;
      - outClientFile.open("filename", fileOpenMode);

Crearea unui fișier secvențial

- Moduri de deschidere a fișierelor

Mod	Descriere
ios::app	Scrie toate ieşirile la sfârşitul fişierului
ios::ate	Deschide un fişier pentru scriere şi se mută cursorul de scriere la sfârşitul fişierului. Datele pot fi scrise oriunde în fişier
ios::in	Deschide un fişier pentru citire
ios::out	Deschide un fişier pentru scriere
ios:trunc	Şterge conţinutul unui fişier dacă există (la fel se comportă şi ios::out)
ios::binary	Deschide un fişier binar pentru citire sau sciere

Exemplu

```
ofstream outClientFile( "clients.dat", ios::out );

ofstream outClientFile( "clients.dat");
```

Crearea unui fișier secvențial

- Operaţii
  - operatorul !
    - !outClientFile
    - returnează adevărat dacă fişierul nu a fost deschis cu succes
  - operatorul void *
    - converteşte streamul la un obiect de tip pointer
  - închidera unui fişier
    - outClientFile.close()
    - apelat automat de destructor
  - Scrierea într-un fişier
    - outClientFile << myVariable
- Exercitiu
  - cititi urmoarele date de la tastatura si scrietile intr-un fisier
    - datele sunt data sub forma unui record care contine numar cont, nume, suma

Crearea unui fișier secvențial. Rezolvare exercitiu

```
#include <iostream>
#include <fstream>

using namespace std;

int main()
{
   ofstream
outClientFile("Client.dat",
ios::out);

   if (!outClientFile){
      cerr << "Fisierul nu a putut fi
deschis" <<endl;
   }
   cout << "Introdu un cont, nume si
o suma" << endl
      <<" Introdu sfarsitul de fisier
cand s-a terminat citirea ";
   int cont;
   char nume[30];
   double suma;

   while(cin >> cont >> nume >>
suma)
   {
      outClientFile << cont << ' ' <<
nume << ' '<<suma<<endl;
      cout<<'?';
   }
   return 0;
}
```

Citireaunui fișier secvențial

- Operaţii
  - operatorul !
    - !inClientFile
    - returnează adevărat dacă fişierul nu a fost deschis cu succes
  - operatorul void *
    - converteşte streamul la un obiect de tip pointer
  - închidera unui fişier
    - inClientFile.close()
    - apelat automat de destructor
  - citirea dintr-un fişier
    - inClientFile >> myVariable
- Exercitiu
  - cititi datelin fişierul creat la exemplul anterior

Citireaunui fișier secvențial. Exemplu

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

int main()
{
   ifstream inClientFile("Client.dat", ios::in);

   if (!inClientFile)
   {
      cerr << "Fisierul nu a putut fi deschis" <<endl;
   }
   int cont;
   char nume[30];
   double suma;

   while(inClientFile >> cont >> nume >>suma)
   {
      cout  << left << setw( 10 ) << cont << setw(10) << nume << setw(10)<< setprecision( 4 ) << right <<suma<<endl;
   }
   return 0;
}
```

Citireaunui fișier secvențial. Poziționarea în fișiere

- Poziționarea pointerului de citire în fișier
  - numărul de bytes care se citesc/scriu
  - Funcţii de repoziţionare
    - seekg (seek get pentru clasa istream)
    - seekp (seek put pentru clasa ostream)
    - Au doi parametrii offset şi direcţie
      - offset - numărul de bytes
      - direcţia
        - ios::beg - relativ la începutul streamului
        - ios::cur - relativ la poziţia curentă
        - ios::end - relativ la sfârşitul streamului

Citireaunui fișier secvențial. Poziționarea în fișiere

- Recitirea unui fișier
  - //citeste fisierul
  - //resetare eofbit pentru urmatoarea citire
  - inClientFile.clear();
  - //mutare la începutul fișierului
  - inClientFile.seekg(0);
  - //recitire fișier de la început

Modificarea unui fișier secvențial

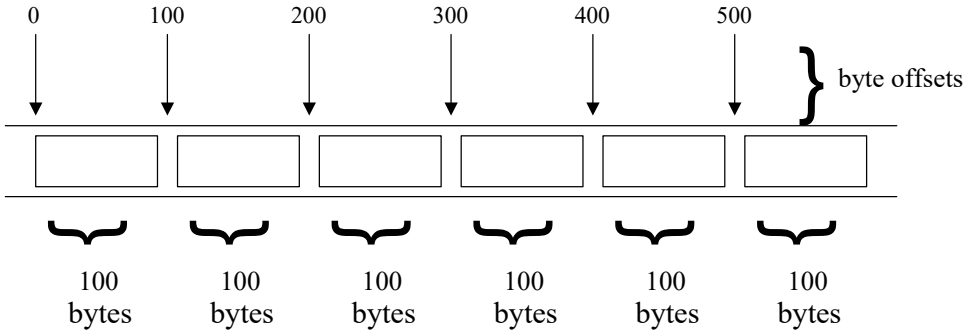
- Noua înregistrare nu trebuie să fie mai lungă decât vechea înregistrare
  - Nu ar trebui să suprascrie următoarea înregistrare
  - Ar trebui să rescriem fiecare înregistrare într-un nou fișier
    - se copie toate înregistrările înaintea celei care se va modifica
    - se scrie o nouă versiune a înregistrări
    - se copie toate înregistrările după înregistrarea curentă
  - Poate fi acceptabilă dacă se modifică multe înregistrări

Exemplu

```
#include <iostream>
#include <string>
#include <vector>
#include <fstream>
#include <algorithm>
using namespace std;
int main(){
  string from, to;
  cout << "Fisierul de unde se citeste : "; cin >>
from;
  cout << "Fisierul de unde se scrie : "; cin >>
to;
//deschidere fisier
  ifstream is(from.c_str());
//definire iteratori fisier
  istream_iterator <string > isi(is);
  istream_iterator <string > eos; //end of
stream
//definire vector cu continut fisier
  vector <string> v(isi, eos);

//sortare vector de cuvinte
  sort(v.begin(), v.end());

//deschide fisier pentru scriere
  ofstream os(to.c_str());

//definire iterator
  ostream_iterator <string> osi(os, "\n");

//scriere in fisier fara duplicate
  unique_copy(v.begin(), v.end(), osi);

  return (!is.eof() && !os);
}
```

Fișiere cu acces aleatoriu

- Random Access Files
  - programatorii ar trebui să creeze fișiere cu acces aleatoriu
  - un mod simlu de a stoca înregistrări care pot fi adăugate/modificate/șterse ușor
    - se calculează poziția în fișier în funcție de dimensiunea înregistrării
    - se găsește poziția în fișier în funcție de cheia înregistrării



Fișiere cu acces aleatoriu vs. fișiere text

- ”1234567” (char *) vs 1234567
  - scrie raw bytes in fișier
  - char * ocupă 8 bytes (unul pentru fiecare caracter)
  - int se reprezintă pe un număr fix de bytes
    - 123 are aceeaaşi lungime ca 1234567
  - operatirul << şi funcţia write()
    - outfile << numar
      - Afişează numerele (int) ca şi caracter
      - număr variabil de bytes
    - outfile.write(const char *, int)
      - ia pinteri din memorie şi numărul de bytes care îl scrie
        - copie direct în fişier din memorie

Fișiere cu acces aleatoriu vs. fișiere text

Exemplu

- int i = 5567;
- int*p = &i;
- file.seekp(8);
- file.write((char*) p , sizeof(int));

- outFile.write((char*)(&number), sizeof(number));
  - &number este un int*
    - Convertit la char*
  - sizeof(number)
    - Dimensiunea variabilei number (un int) în bytes
  - funcţia de citire este similară

Scrierea de date într-un fișier cu acces aleatoriu

- Folosire seekp pentru a scrie la o locație exactă
  - prima înregistrare începe la byteul 0
  - a doua înregistrare începe la bytelul 0 + sizeof(obiect)
  - orice înregistrare (NrInregistrare -1) * sizofObject

- Exemplu
  - inClientFile.seekp((i-1) * sizeof(Client));
  - inClientFile.write((char*)(&client), sizeof(client));

Citirea dintr-un fișier cu acces aleatoriu

- Citirea este similară cu scrierea
  - Citește raw bytes din fișier în memmorie
  - inFile.read((char*)(&number), sizeof(int));
    - &number: locaţia datelor care vor fi citite
    - sizeof(int): câţi bytes vor fi citiţi
  - Nu folosiți inFile >> number cu raw bytes
    - >> converteşte numerele la text
    - de ex 123 nu va ocupa aceeaşi numar de bytes cu 1234

Serializare

- Procesul de scriere/citire automata a unui obiect
- Supraîncarcarea operatoriilor <<, >>
- Se pierde informatia despre tipul obiectului
  - dacă programul ştie tipul obiectului poate citi tipul corect
  - dacă un fişier conţine tipuri diferite de obiecte
    - adaugarea unui cod pentru fiecare tip
    - folosirea unui switch pentru citirea tipului corect