Programare orientată obiect

Cursul 10

Sumar

- Intrări/ieșiri în C++ (3) – cont.
  - Formatarea conținutului
    - Manipulatori de format
    - Metode
  - Intrări/ieșiri nestandard
    - Fișiere
    - Memorie
- Funcții generice și clase generice
  - Definire
  - Instanțiere
  - Speciaizare

3. Indicatori de format

- ***ios::left***, ***ios::right***, ***ios::internal***
- ***ios::scientific***, ***ios::fixed***
- ***ios::hex***, ***ios::dec***, ***ios::oct***
- **ios::showpos**
- **ios::showbase**
- **ios::skipws**
- **ios::uppercase**
- **ios::showpoint**

- ios::adjustfield
- ios::floatfield
- ios::basefield

Indicatori de format - Metode

- Setarea indicatorilor precizați cu returnarea stării indicatorilor de format
  - unsingned int **setf**(unsingned int)
- Setarea indicatorului precizat din cîmpul de indicatori (masca); returnează starea indicatorilor de format
  - unsingned int **setf**(unsingned int, unsingned int)
- Resetarea indicatorior precizați
  - **unsetf**(unsingned int)
- Obținerea/Modificarea indicatorilor și resetarea celor neincluși ca parametri
  - unsingned int **flags**(unsingned int)
  - unsingned int **flags**()

Indicatori de format - Metode

- Exemplu
  - int x = 200;
  - unsigned int indFmt = cout.setf(ios::hex, std::ios::basefield);
  - cout<<x<<endl;
  - cout.setf(indFmt);

Indicatori de format - Manipulatori

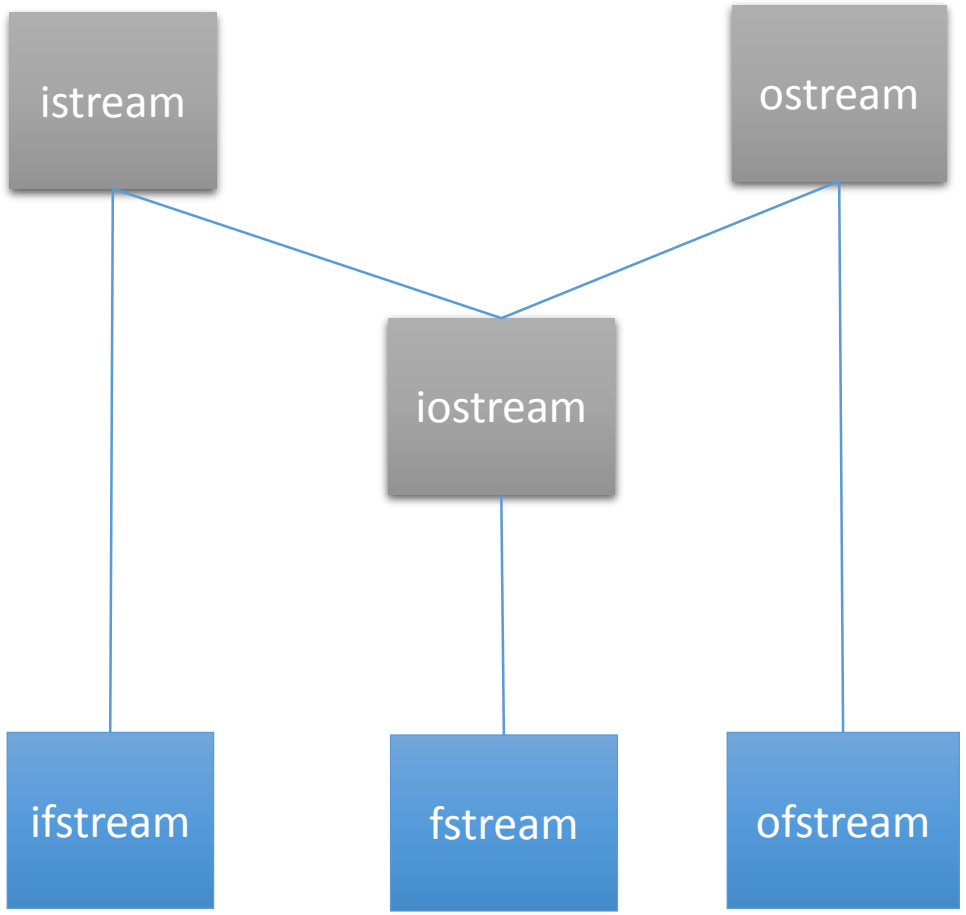
- Setarea indicatorilor precizați
  - **setiosflags**(unsingned int)
- Resetarea indicatorior precizați
  - **resetiosflags**(unsingned int)

Indicatori de format - Manipulatori

- Exemplu:

```
int x = 200;
cout<<resetiosflags(ios::dec);
cout<<setiosflags(ios::hex);
cout<<x<<endl;
cout<<resetiosflags(ios::hex);
cout<<x<<endl;
```

I/O C++ nestandard (fișiere)



Operații cu fișiere

- #include <fstream>
- namespace std;
- Fișiere de intrare
  - istream
- Fișiere de ieșire
  - ostream
- Fișiere de intrare/ieșire
  - fstream

Operații cu fișiere

- Deschiderea fișierelor
  - Constructor
  - Metoda **open**(char *, openmode={ios::in, ios::out, ios::in | ios::out})
  - Rezultatul operatiei de deschidere
    - bool is_open()
- Închiderea unui fișier
  - automat, la ieșirea din bloc
  - metoda **close**()

Moduri de deschidere

- Intrare:
  - **ios::in**,
- Ieșire:
  - **ios::out,**
- Binar:
  - **ios::binary**
- Poziționare la sfîrșit pentru scriere
  - **ios::app**
- Poziționare la sfîrșit
  - **ios::ate**
- Trunchiere
  - **ios::trunc**
- Fără creare
  - **ios::nocreate**
- Fără suprascriere
  - i**os::noreplace**

Fișiere de intrare

- Deschiderea unui fișier
  - ifstream fisIn("nume fisier");
  - ifstream fisIn; fisIn.**open**("nume fisier");
  - Implicit cu ios::in și modul text; pentru alte moduri se adaugă parametrul corespunzător
- Citirea din fișier
  - Cu formatare
    - Operatorul **>>**
  - Fără formatare
    - Metodele **read**(), **get**()

Fișiere de ieșire

- Deschiderea unui fișier
  - ofstream fisO("nume fisier");
  - ofstream fisO; fisO.**open**("nume fisier");
  - Implicit cu ios::out și modul text; pentru alte moduri se adaugă parametrul corespunzător
- Scriere în fișier
  - Cu formatare
    - Operatorul **<<**
  - Fără formatare
    - Metodele **write**(), **put**()

Fișiere de intrare/ieșire

- Deschiderea unui fișier
  - fstream fisIO("nume fisier");
  - fstream fisIO; fisIO.**open**("nume fisier");
  - Implicit cu ios::in| ios::out și modul text; pentru alte moduri se adaugă parametrul corespunzător
- Citire/Scriere în fișier
  - Operatorii și metodele din clasele de bază

Poziționarea în fișiere (citire = g/scriere = p)

- n octeți de la începutul fișierului:
  - fis.seekg(n, ios::beg); //sau
  - fis.seekg(n);
- n octeți de la poziția curentă în fișier:
  - fis.seekg(n, ios::cur);
- n octeți de la sfîrșitul fișierului:
  - fis.seekg(n, ios::end);
- La sfîrștul fișierului:
  - fis.seekg(0, ios::end);
- Numărul de octeți pînă la poziția curentă
  - fis.tellg()

Scrierea/citirea de tipuri complexe

```
struct Produs
{
        //
};

Produs p;
fstream fis("io.dat", ios::binary| ios::out);
fis.write((char *)&p, sizeof(Produs));
fis.close();
```

Scrierea/citirea de tipuri complexe

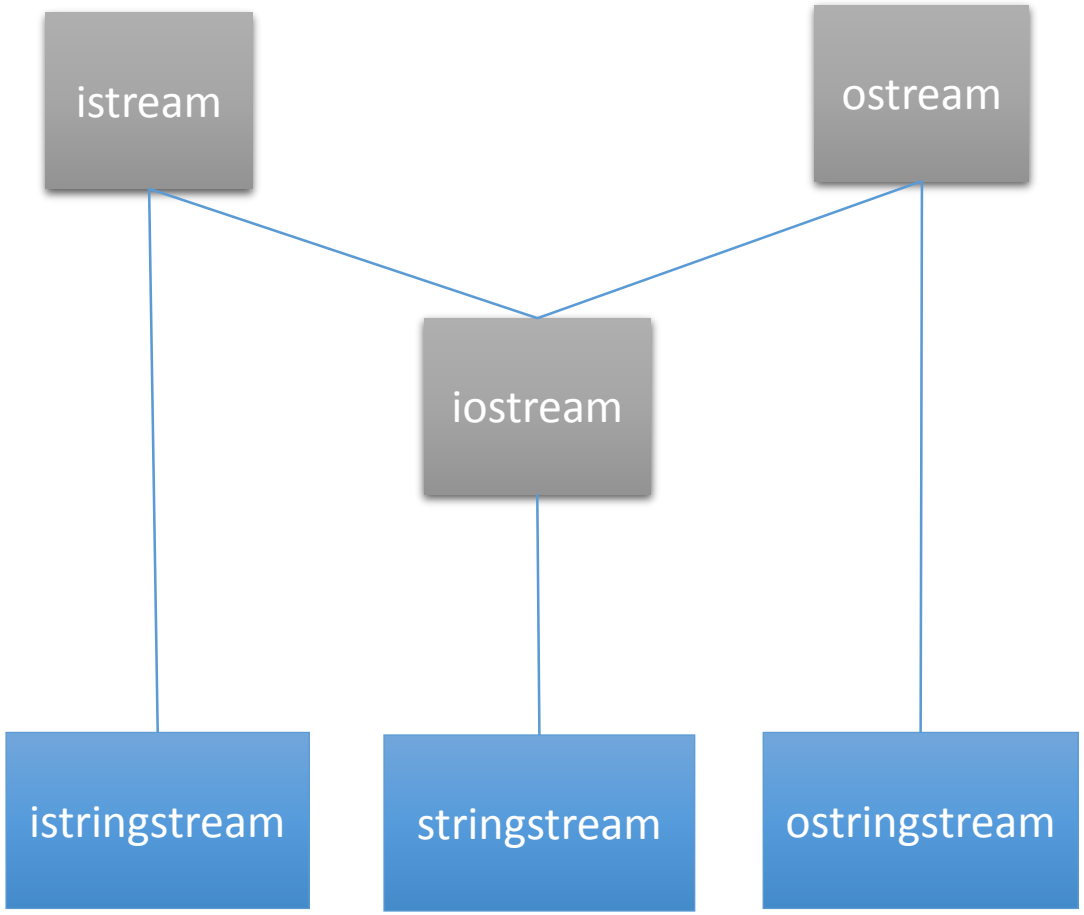
```
//…
fis.open("io.dat", ios::in | ios::binary);
Produs p2;

fis.read((char *)&p2, sizeof(Produs));
fis.close();
```

Supraîncărcarea operatorilor pentru I/O cu fișiere

```
class Produs
{
        //…
        friend ofstream & operator<<(ofstream &o, Produs p);
        friend ifstream & operator>>(ifstream &i, Produs &p);
}
…
Produs p;
ofstream fiso("produse");
fiso<<p;
fiso.close();
```

Fluxuri în memorie



Fluxuri în memorie

- Utile la conversii
- #include <sstream>
- namespace std;
- Funcția **str**()
  - Forme pentru set și get
- Șiruri în alte tipuri
- Alte tipuri către șiruri
- Sînt supraîncărcați operatorii **<<** și **>>**

istringstream

- Conversie șiruri în alte tipuri
- Flux de intrare
- Exemplu
  - istringstream iss("a 10.20 nota");
  - char a; double b; char d[10];
  - iss>>a>>b>>d;
  - cout<<a<<b<<d<<endl;

ostringstream

- Flux de ieșire
- Conversie alte tipuri în șiruri
- Exemplu:
  - ostringstream oss;
  - char a; double b; char d[10];
  - oss<<a<<" "<<b<<" "<<d;
  - cout<<oss.str();

stringstream

- Fluxuri de intrare/ieșire

Funcții și clase generice

Funcții și clase generice

- Definirea de metafuncții și/sau metaclase (șabloane, clase parametrizate, template)
- În locul tipurilor concrete (fundamentale sau utilizator) se utilizează o denumire generică
- Tipul concret este precizat ulterior în cod la apel sau definire/declarare
- Funcțiile/clasele vor fi instanțiate de compilator prin înlocuirea tipurilor generice cu tipurile concrete de date
- Funcțiile/clasele generice sînt introduse prin cuvîntul cheie **template**
- Tipurile generice sînt introduse prin cuvintele cheie **typename** sau **class**

Funcții și clase generice

Funcții generice

- Prototip

```
template <lista_tipuri_generice>
tip functie(param)
```

- unde:
  - `lista_tipuri_generice` poate include (separate prin virgulă):
    - `typename TIP_GENERIC` și/sau tipuri fundamentale/utilizator
      - acestea pot fi inițializate cu valori implicite (tipuri concrete/constante)
  - tip/param pot fi tipuri:
    - Generice (TIP_GENERIC etc.)
      - ar trebui să apară cel puțin o dată!
    - Fundamentale
    - Definite de utilizator

Funcții generice – Apel

- Selecția funcțiilor:
  - Tipul parametrilor
  - Apel cu tip explicit
- Au prioritate funcțiile non-parametrice, fără conversii
- Sînt căutate funcțiie generice specializate
- Urmează funcțiile generice
- Alte funcții (conversii)/eroare de compilare/editare de legături

Funcții generice

```
template <typename T>
void minMax(T a, T b, T &min, T &max)
{
    if (a <b)
    {
        min = a; max = b;
    }
    else
    {
        min = b; max = a;
    }
}
```

Funcții generice

- double d1, d2, dm, dM;
- int i1, i2, im, iM;

- minMax(d1, d2, dm, dM);//instanțiere cu double
- minMax<double>(d1, d2, dm, dM);//instanțiere explicită cu double
- minMax(i1, i2, im, iM);//instanțiere cu int
- minMax(d1, i1, dm, dM);//eroare!
- minMax<double>(d1, i1, dm, dM);//instanțiere explicită cu double+conversie

Instanțiere și specializare

- Definire

  template <typename T>
  T aduna(T &a, T &b)
  { … }

- Instanțiere explicită

  template **int** aduna<**int**>(**int** &a, **int** &b);

- Specializare

  template **<> int** aduna<**int**>(**int** &a, **int** &b)
  { … }