Universitatea Politehnica Timișoara

Programare orientată pe obiecte

Curs 2

Programarea în stilul C++

Programul 1

- IOSTREAM este pentru C++ ceea ce este STDIO.H
- using namespace std; - toate elementele bibliotecii standard C++ sunt declarate într-un spațiu de nume (namespace) și anume în namespace-ul cu titlul std.
- **cout << "Iata iesirea.\n"; // un comentariu de o linie**
- functie de iesire noua
- modul de realizare a comentariilor
- **cin >> i;** - functie de intrare noua
- **>>** - semnificaţie şi de deplasare dreapta (*shift*) şi de **flux** de informaţii (*stream*)
- **cout << i << " la patrat este " << i*i << "\n";** - asocierea mai multor operaţii de intrare/ieşire
- **return 0;** - returnarea valorii zero indică terminarea normală a programului
- **main()** este declarată uzual într- un program în **C++** :
- nedeclarând nimic (ceea ce înseamnă că tipul returnat este void) sau returnând o valoare întreagă.
- Nu este obligatoriu ca **main()** să returneze o valoare.

Comparație C – C++

C	C++

Ex. 1
typedef struct CLS {
int a,b;
} cls;
cls oc_c;
ob_c.a=1; ob_c.b=1;
	`class CLS{`
	`int a,b;`
	`public:`
	`CLS (int z=0) {`
	`a=b=z;`
	`}`
	`CLS ob_c(1);`

Ex. 2 text=malloc(strlen(s)) free(text)	`text=new char [strlen(s)]` `delete text`

Operatori de intrare/ieșire

- Cînd sunt folosiţi pentru operaţii de intrare/ieşire, operatorii << şi >> sunt capabilli să manevreze orice tip de date încorporat în **C++.**

  Atenţie ! Operatorul >> lucrează cu şirurile în acelaşi fel în care o face şi specificatorul ***%s*** *pentru* ***scanf()****, adică încheie citirea intrării cînd întîlneşte primul caracter de spaţiu.*

Declararea variabilelor locale

```
/*   incorect în C, permis în C++  */
int f()
{
   int i,k;
   for (i=0; i <10; i++) {
    k=i+1;
    int j; /*  da eroare in C  */
    j=i*2;
    ...
   }
}
```

- Se pot declara variabile în **C++** în orice punct dintr-un bloc, nu doar la început.

Prezentarea claselor în C++

- **Cea mai importantă caracteristică din C++:**

  - ***clasele :*** pentru a crea un obiect în **C++** trebuie să îi definiţi mai întâi forma sa generală folosind cuvântul cheie **class**. Aceasta este similară sintactic cu o structură.

Exemplu (pentru o clasă care implementează o stivă) :

```
# define SIZE 100
//  Aici se defineste clasa numita stiva
class stiva {
   int stiv[SIZE];
   int virf_stiva;
public:
   void init();
   void pune(int );
   int scoate();
```

Clasa

- secţiuni *private* (particulare), *publice*
- *funcţii membre, variabile membre*
- ***C++ class*** *creează un nou tip de date care pot fi folosite pentru a construi obiecte de acel tip.*
- *un obiect este o* ***instanţiere*** *(un exemplar) a unei clase exact în acelaşi fel în care altă variabilă este de exemplu un exemplar al tipului de date* ***int*** *(adică o clasă este o* ***abstractizare logică****, iar un obiect este* ***real****, clasa constituie* ***modelul*** *după care se generează* ***obiecte concrete*** *(care există în memoria calculatorului)*

Clasa – continuare 1

- **Forma generală** a declarării unei clase este :

```
class nume_clasa {
        date si functii private
  public:
        date si functii publice
  } lista de obiecte;
```

- Acolo unde trebuie introdus codul pentru ***funcţiile membre*** ale unei clase, trebuie specificată apartenenţa la clasă prin scrierea numelui clasei urmat de :: înainte de numele funcţiei. Exemplu :

```
void stiva::pune(int i);
{
  if(virf_stiva==SIZE) {
        cout << "Stiva este plina.";
        return;
  }
  stiv[virf_stiva]=i;
  virf_stiva++;
}
```

:: este operatorul de specificare a domeniului (operator de ***rezoluţie***)

Clasa – continuare 2

- Mai multe clase pot folosi **acelaşi nume de funcţie**, iar compilatorul ştie care funcţie aparţine fiecărei clase datorită operatorului de rezoluţie (***polimorfism***).
- **Referirea** la un membru al clasei dintr-o secţiune de cod care nu face parte din acea clasă se face prin *numele obiectului* cu care facem legătura, *urmat de punct*, astfel :

```
stiva stiva1, stiva2;
stiva1.init();
```

Important ! în exemplul de mai sus **stiva1** şi **stiva2** sunt obiecte distincte, iar iniţializarea obiectului **stiva1** nu provoacă iniţializarea obiectului **stiva2**. Singura legătură dintre ele este că ambele sînt obiecte de acelaşi tip (tipul **stiva**).

Clasa – continuare 3

Atenţie ! Variabilele private ale unui obiect sunt accesibile doar funcţiilor membre ale obiectului. În exemplu precedent, nu putem avea în programul principal:

```
stiva1.virf_stiva=0;
```

(am încerca să accesăm variabile din domeniul privat şi se generează un mesaj de eroare)

- Uzual funcţiile membre ale claselor sunt definite înainte de main() iar funcţiile obişnuite (nemembre) sunt definite după main() - dar nu e obligatoriu. În aplicaţiile uzuale, se foloseşte includerea claselor asociate unui program în fişiere antet.

Supraîncărcarea funcțiilor

- ***supraîncărcarea funcţiilor*** (***overloading***): în **C++**, două sau mai multe funcţii pot să aibă acelaşi nume atât timp cât declaraţiile lor de parametri sunt diferite.
- Se spune că funcţiile cu acelaşi nume sunt ***supraîncărcate*** iar procesul este numit ***supraîncărcarea funcţiilor***.

Supraîncărcarea funcțiilor

- Exemplu : ***Programul 1*** *(funcțiile vabs(): prima pentru valoarea absolută a unui întreg, iar următorul, pentru valoarea absolută a unui număr în virgulă mobilă, în precizie dublă (double). Chiar dacă aceste funcţii efectuează acţiuni aproape identice, în C trebuie folosite nume diferite pentru a diferenţia funcţiile între ele.*

```
#include <iostream>
#include <conio.h>
using namespace std;

int vabs(int);
double vabs(double);

int main()
{
cout << vabs(-10) << endl;
cout << vabs(-22.5) << endl;
_getch();
return 0;
}
```

```
int vabs(int i)
{
   cout << "ABS pentru intregi“ <<endl;
   return i<0 ? -i : i;
}

double vabs(double d)
{
  cout << "ABS pentru double “ <<endl;
  return d<0 ? -d : d;
}
```

```
ABS pentru intregi
10
ABS pentru double
22.5
```

Moștenirea

- **Moştenirea** - una dintre caracteristicile cele mai importante ale unui limbaj de programare OO.
- În **C++** - realizată prin acceptarea ca ***o clasă să încorporeze în declararea sa altă clasă***.
- **Permite** construirea unei ***ierarhii de clase***, trecerea de la cele mai generale la cele mai particulare :
  - definirea ***clasei de bază***, care stabileşte calităţile comune ale tuturor obiectelor ce vor deriva de aici; reprezintă ***cea mai generală descriere***.
  - clasele derivate (obţinute prin moştenire) din clasa de bază se numesc chiar ***clase derivate*** ; o clasă derivată include toate caracteristicile clasei de bază şi în plus calităţi proprii acelei clase

Moștenirea (continuare 1)

- **Exemplu (pentru clădiri)**

Întâi declarăm clasa de bază clădire :

```
//clasa de baza

class cladire {
  int camere;
  int etaje;
  int supraf;
public:
  void nr_camere(int);
  int cite_camere();
  void nr_etaje(int);
  int cite_etaje();
  void nr_supraf(int);
  int cite_supraf();
};
```

În această clasă, clădirile sunt caracterizate prin proprietăţi comune : număr de etaje, număr de camere, suprafaţă. Funcţiile membre care încep cu **nr** iniţializează valorile datelor particulare iar cele care încep cu **câte** returnează acele valori.

Moștenirea (continuare 2)

Mai departe, putem folosi clasa aceasta pentru a deriva **apartament și scoala**

- ***sintaxa pentru moştenire :***

class nume_clasa_derivata: ***acces*** **nume_clasa_mostenita {**

// aici este corpul noii clase

...

};

```
// casa este derivat din cladire
class casa : public cladire {
   int dormitoare;
   int bai;
public:
 void nr_dormitoare(int);
  int cite_dormitoare();
  void nr_bai(int);
  int cite_bai();
};
```

```
// scoala este derivat tot din cladire

class scoala : public cladire {
  int sali_clasa;
  int laboratoare;
public:
  void nr_sali_clasa(int);
  int cite_sali_clasa();
  void nr_laboratoare(int);
  int cite_laboratoare();
};
```

Moștenirea (continuare 3)

- **Accesul poate fi public, private sau protected :**
  - **public - toate elementele publice ale clasei de bază vor fi publice, de asemenea, şi în clasa derivată care o moşteneşte (de aceea, în exemplul prezentat toţi membrii clasei apartament au acces la funcţiile membre din clasa cladire ca şi cum ar fi fost declarate în interiorul clasei apartament *(cum ar fi să ai cheia de la apartamentul părinţilor)***
  - **Totuşi, funcţiile membre ale clasei apartament nu au acces la părţile private din cladire, în acest fel moştenirea neîncălcînd principiul încapsulării, necesar în POO *(cum ar fi să ai cheia de la apartamentul părinţilor dar să nu ai acces peste tot în casă, unele camere să fie încuiate pentru tine)***

Moștenirea (continuare 3)

- ***În concluzie, o clasă derivată are acces atât la propriii membri cât şi la membrii publici ai clasei de bază.***
- ***Exemplu*: un program simplu, care evidenţiază principiile şi regulile enunţate şi foloseşte clasa cladire ca şi clasă de bază şi clasele apartament şi scoala ca şi clase derivate - *Programul 2***
- **Avantaje ale moştenirii : utilizarea *ierarhiilor de clase*, suport pentru *polimorfismul din timpul rulării* prin mecanismul funcţiilor virtuale**