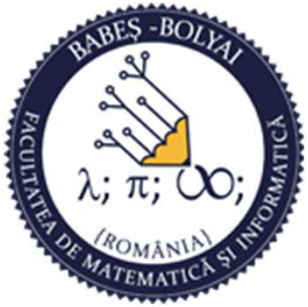
UNIVERSITATEA BABEŞ-BOLYAI
Facultatea de Matematică şi Informatică



Programare orientată obiect

Curs 05

Laura Dioşan

POO

- Clase
  - Şabloane (Template)
    - Funcţii
    - Clase

Re-utilizare

- La nivel de cod obiect
  - Relaţia de compoziţie
  - Relaţia de derivare
- Nivel de cod sursă (elemente generice)
  - prin void*
  - prin mecanismul *template*
    - funcţii template
    - clase template
    - clase cu membrii template
  - prin clase abstracte

Funcţii template

- Funcţii generice
- Lista prametrilor formali conşine parametri genericei care vor fi înlocuiţi, la apel, cu parametri particulari
- Legătura cu supraîncărcarea funcţiilor
- A se consulta exemplul din directorul *06/templateFunctions*

Funcţii template

- Declarare şi definire

```
template <class T> return_type fc_name(type param);
```

declarare template

argument template
(parametru substituibil)

- Exemplu

```
template <class T> T maxim(T a, T b){
            return (a > b ? a : b);
}
int x = 5;int y = 8;
int maxInt = maxim<int>(x, y) ;
cout << "integer maxim is " << maxInt << endl;

double z = 9.5;double t = 12.24;
double maxDouble = maxim<double>(z, t);
cout << "double maxim is " << maxDouble << endl;

Student student1(10);Student student2(9);
Student maxStudent = maxim<Student>(student1, student2);
cout << "Student maxim is " << maxStudent << endl;
```

instanţiere
template

Funcții template

- Funcţia *maxim* poate fi apelată pentru orice fel de argumente:
  - int
  - double
  - obiect (Flower, Student, etc) ⇔ pentru aceste clase trebuie supraîncărcat operatorul >

Clase template

- Elemente generice:
  - C : void*
  - Java, SmallTalk : o singură clasă de bază (Object ⇔ IE); din această clasă sunt derivate toate celelalte clase
  - C++
    - Pointeri la clase abstracte
    - Clase template

Clase template

- Un macro (framework, skeleton) care descrie o mulţime de clase similare
- Similar unei interfeţe
- **template**
  - Indică compilatorului că definiţia clasei va manipula unul sau mai multe tipuri de date nespecificate
  - La moment utilizării codul clasei generat pe baza template-ului trebuie să aibă acces la tipul de date respectiv pentru ca compilatorul să-l poată înlocui

Clase template - schelet

```
template <class T> class MyClass{
private:
  T data;
  static T staticData;
public:
  MyClass();
  MyClass(MyClass<T> &);
  void fc1(T);
  T fc2();
};

template <class T> T MyClass <T>::staticData = 0;

template <class T> MyClass<T>::MyClass(){…}
template <class T> MyClass<T>::MyClass(MyClass<T> &mc){…}
template <class T> void MyClass<T>::fc1(T x){…}
template <class T> T MyClass<T>::fc2(){…}

MyClass<int> mc1;
MyClass<Flower> mc2;
```

Clase template

- Întreaga *declarare* şi *definire în acelaşi fişier*
  - Compilatorul nu va aloca spaţiu pentru o astfel de clasă, ci va aştepta până la realizarea unei instanţieiri template
  - Există un mecanism, undeva între compilator şi linker, pentru înlăturarea definiţiilor multiple a unui template identic
- Dacă definirea se scrie în alt fişier (source file), atunci:
  - Programul de test trebuie să includă acest fişier sursă
  - Programul trebuie compilat într-un mod mai special

Parametrul template

- Poate fi:
  - orice tip de dată predefinit
    - int, bool, double, char, etc
  - orice tip de dată definit de user
    - Student, Flower, etc
  - expresii constante – folosite pentru a preciza capacitatea unei structuri
  - void
  - adrese ale variabilelor (obiectelor) ne-locale
  - funcţii
  - alte template-uri
- Nu poate fi:
  - expresii constante flotante
  - adrese ale elementelor dinţun vector
  - adrese ale variabilelor locale
  - tipuri de date locale
- A se consulta exemplul din directorul *05/templateParam*

Clasă template - exemplu

- Vector cu elemente generice
  - A se consulta exemplul din directorul *05/templateVector*
- Secvenţă ordonată cu elemente generice
  - A se consulta exemplul din directorul *05/templateOrdSeq*