



UNIVERSITATEA BABEŞ-BOLYAI
Facultatea de Matematică și Informatică



Programare orientată obiect

Curs 05

Laura Dioşan

POO

- Clase

- Şabloane (Template)

- Funcții

- Clase

Re-utilizare

- La nivel de cod obiect
 - Relația de compoziție
 - Relația de derivare
- Nivel de cod sursă (elemente generice)
 - prin void*
 - prin mecanismul *template*
 - funcții template
 - clase template
 - clase cu membrii template
 - prin clase abstracte

Funcții template

- Funcții generice
- Lista prametrilor formali conține parametri generice care vor fi înlocuiți, la apel, cu parametri particulari
- Legătura cu supraîncărcarea funcțiilor
- A se consulta exemplul din directorul *06/templateFunctions*

Funcții template

□ Declaraire și 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;
```

instanțiere
template

Funcții template

- Funcția *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 mulțime de clase similare
- Similar unei interfețe
- **template**
 - Indică compilatorului că definiția 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

Class 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 și definire în același fișier*
 - Compilatorul nu va alocă spațiu pentru o astfel de clasă, ci va aștepta până la realizarea unei instanțieri template
 - Există un mecanism, undeva între compilator și linker, pentru înlăturarea definițiilor multiple a unui template identic
- Dacă definirea se scrie în alt fișier (source file), atunci:
 - Programul de test trebuie să includă acest fișier 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
 - funcții
 - alte template-uri

- Nu poate fi:
 - expresii constante flotante
 - adrese ale elementelor dintr-un 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*

- Secvență ordonată cu elemente generice
 - A se consulta exemplul din directorul *05/templateOrdSeq*