

Programare II

Limbajul C/C++

CURS 10



Curs anterior

❑ Supraîncărcarea operatorilor

- ❑ Funcții membre
- ❑ Funcții prietene

❑ Operatori

- ❑ Asignare
- ❑ Binari
- ❑ Prescurtați
- ❑ Unari
- ❑ Conversii de tip

CUPRINS

❑ Tipuri de date abstracte

❑ Tipuri de date generice

❑ Funcții şablon

❑ Clase şablon

Tipuri de date abstracte

- ❑ Tipuri de date abstracte
 - ❑ Realizarea de tipuri de date definite de utilizator care se comportă ca și tipurile implicite (build-in)
 - ❑ De ce proprietăți avem nevoie;
 - ❑ Implementarea unui set de operații pentru ele
- ❑ Tipuri generice de date
 - ❑ Parametrizare astfel încât să funcționeze **cu o mulțime** de date și structuri de date potrivite

Tipuri de date abstracte

PROBLEMA

- Creați o clasă care să permită lucrul cu o stivă de numere întregi

Dacă trebuie creată o stivă de numere reale?

SOLUȚIA

```
class Stiva{  
    int *tab;  
    int dim, index;  
  
public:  
    Stiva( int d ):index(0), dim(d) { tab = new int[dim]; }  
    bool isGoala() { return index == 0; }  
  
    bool isPlina() { return index == dim; }  
  
    void push( int x ) {  
        if (isPlina()) throw OutOfBounds(); tab[index++] = x;  
    }  
    int pop () {  
        if (isGoala()) throw OutOfBounds(); return tab[--index];  
    }  
    class OutOfBounds{};  
};
```

Tipuri de date abstracte

PROBLEMA

- Creați o clasă care să permită lucrul cu o stivă de numere ireale

Dacă trebuie creată o stivă de numere complexe?

SOLUȚIA

```
class StivaDouble{  
    double *tab;  
    int dim, index;  
  
public:  
    Stiva( int d ):index(0), dim(d) { tab = new double[dim]; }  
    bool isGoala() { return index == 0; }  
  
    bool isPlina() { return index == dim; }  
  
    void push( double x ) {  
        if (isPlina()) throw OutOfBounds(); tab[index++] = x;  
    }  
    double pop () {  
        if (isGoala()) throw OutOfBounds(); return tab[--index];  
    }  
    class OutOfBounds{};  
};
```

Care sunt diferențele între cele două clase?

STIVA NUMERE INTREGI

```
class Stiva{  
    int *tab;  
    int dim, index;  
  
public:  
    Stiva( int d ):index(0), dim(d) { tab = new int[dim]; }  
    bool isGoala() { return index == 0; }  
  
    bool isPlina() { return index == dim; }  
  
    void push( int x ) {  
        if (isPlina()) throw OutOfBounds(); tab[index++] = x;  
    }  
  
    int pop () {  
        if (isGoala()) throw OutOfBounds(); return tab[--index];  
    }  
    class OutOfBounds{};  
};
```

1. Numele
2. Tipul de date al tabloului de elemente
3. Diferențe la tipul de return sau tipul parametrilor unor funcții

STIVA NUMERE REALE

```
class StivaDouble{  
    double *tab;  
    int dim, index;  
  
public:  
    Stiva( int d ):index(0), dim(d) { tab = new double[dim]; }  
    bool isGoala() { return index == 0; }  
  
    bool isPlina() { return index == dim; }  
  
    void push( double x ) {  
        if (isPlina()) throw OutOfBounds(); tab[index++] = x;  
    }  
  
    double pop () {  
        if (isGoala()) throw OutOfBounds(); return tab[--index];  
    }  
    class OutOfBounds{};  
};
```

Generice

- ❑ Programare generică
 - ❑ Exprimă structuri de date/algoritmi independente(i) de detaliile de reprezentare
 - ❑ Structuri de date generice
 - ❑ Algoritmi generici
- ❑ În C++ se regăsesc sub numele de **template-uri**

Tipuri de date generice

```
template <typename T> class Stiva {                                int main () {  
    T *tab;                                         Stiva <int> s(4);  
    int dim, index;                                 s.push(95);  
public:                                              s.push(7);  
    Stiva( int d ):index(0),dim(d) {tab = new T[ dim]; }  
    bool isGoal() { return index == 0; }  
    bool isPlina() { return index == dim; }  
    void push( T x) {  
        if (isPlina()) throw OutOfBounds();  
        tab[index++] = x; }  
    T pop () {  
        if (isGoal()) throw OutOfBounds();  
        return tab[--index]; }  
    class OutOfBounds();  
};  
  
template<typename T> void Stiva<T>::push(T el){ ....}
```

Template-uri

- ❑ Un template (șablon, tip de date parametrizat) reprezintă o familie de tipuri de date sau funcții, **parametrizeate cu un tip generic**
- ❑ Avantaje
 - ❑ Reutilizarea codului
 - ❑ Permite implementarea de **biblioteci cu scopuri generale**

Template-uri

❑ Sintaxă

```
template <listaDeParametri> declaratie;
```

❑ unde

❑ listaDeParametrii

❑ O listă de parametrii ai template-ului separată prin virgulă

❑ Parametrii de tip (typename T);

- ❑ T poate fi initializat cu un tip de bază (int, char, float),

- ❑ un tip de dată definit de utilizator (MyClass, complex, ...),

- ❑ un tip de pointer (void *, ...),

- ❑ un tip referință (int&, MyClass &, ...)

- ❑ o instanță a unui template

❑ Parametrii non-tip (int i);

- ❑ parametri non-tip pot fi instanțiați doar cu valori constante și sunt constanți în definirea/declararea clasei

Template-uri

❑ C++ permite definirea de

❑ Clase template

❑ Funcții template

Template-uri. Instantiere

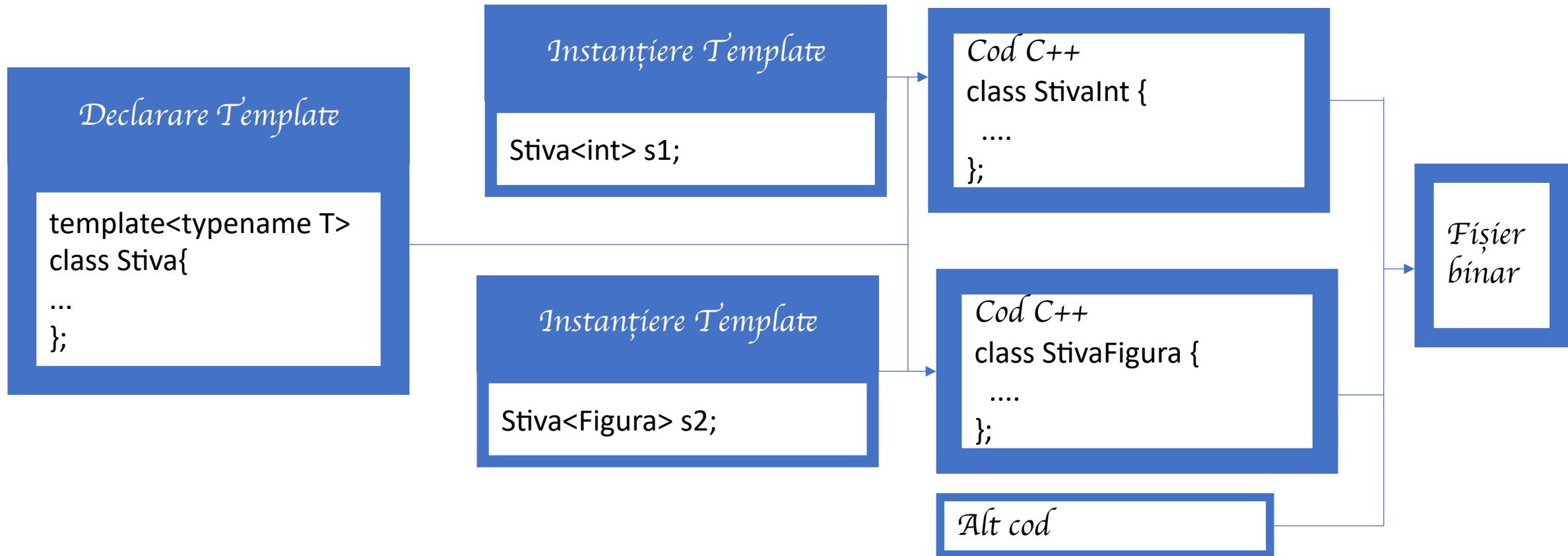
❑ Utilizare clase template

```
vector <int> d1;  
vector <double> d2;  
Buffer <char, 10> d3;  
MyClass <int, Employee, 10, 0.5>x;  
MyClass<Employee&, Manager*, 20-1, 103/7> y;
```

❑ Cum tratează compilatorul instantierea claselor template?

Template-uri. Instantiere. Generarea Codului

- Compilatorul C++ generează cod doar pentru clasele/funcțiile utilizate.



Generarea codului

- ❑ Compilatorul va genera declarații de clase doar pentru template-urile instanțiate
- ❑ În clasa template se generează funcțiile membre ale template-ului utilizare
- ❑ Dacă template-ul nu este instantiat nu se generează cod

Generarea codului

EXEMPLU

```
int main (int, char* [] ) {  
    vector <int>v0, v1;  
    v.add(1) ;  
    v.add(100);  
    cout << v.get (0);  
    v1 = v0;  
    vector <float>fv;  
    return 0;  
}
```

CE SE VA GENERA?

- ❑ Pentru instanțierea vector <int>
 - ❑ declarația clasei (include funcțiile inline)
 - ❑ operatorul de atribuire
 - ❑ funcția add ()
 - ❑ funcția get ()
- ❑ Pentru instanțierea vector <float>
 - ❑ declarația clasei (include funcțiile inline)

Verificarea tipului

❑ Erori în definirea unui template

- ❑ Care pot fi determinate la [compilare](#), exemplu punct și virgulă sau cuvinte cheie scrise greșit

❑ Care pot fi identificate la [instantierea](#) template-ului

- ❑ Prima instantiere a unui template, utilă pentru detectarea și rezolvarea erorilor din template
- ❑ Pentru depanare se utilizează tipurile cele mai frecvente

❑ Care pot fi identificate la [execuție](#)

Verificarea tipului

- ☐ Argumentele pasate la template-uri trebuie să aibă operațiile cerute de template

```
class X { };  
  
template<typename T> class vector{  
    ...  
    void add(T);  
};  
  
template<typename T> void  
vector<T>::add(T x) {  
    // add e to the vector v  
    std::cout << "Added element " << x;  
}
```

```
void f1() {  
    vector vi; // instantiere  
    vi.add(100);  
    // => OK!  
}  
  
void f2() {  
    vector vX; // instantiere  
    vX.add(X(7));  
    // => eroare! De ce este eroare?  
}
```

Functii template

```
int min( int x, int y) { return x<y?x:y; }
```

```
float min( float x, float y) { return x<y?x:y; }
```

```
complex& min(complex& x, complex& y) { return x<y?x:y; }
```

```
void f() {
    complex c1(1,1), c2(2,3);
    min(0,1);
    min(6.5, 3);
    min(c1, c2);
}
```

Cum putem grupa
cele 3 functii?

Functii template

FARA FOLOSIRE TEMPLATE

```
int min( int x, int y) { return x<y?x:y; }

float min( float x, float y) { return
x<y?x:y; }

complex& min(complex& x, complex& y) {
return x<y?x:y; }

void f() {
    complex c1(1,1), c2(2,3);

    min(0,1);
    min(6.5, 3);

    min(c1, c2);

}
```

FOLOSIRE TEMPLATE

```
template <typename T> T min( T x, T y) {
    return x<y?x:y;
}

void f() {
    complex c1(1,1), c2(2,3);

    min(0,1);
    min<float>(6.5, 3);

    min(c1, c2); // min<complex>(c1,c2);

}
```

Functii template

□ Sintaxă

```
template < listaTipuriParametrii > numeFunctie( lista de  
parametrii);
```

□ Apelarea funcțiilor template

- min <int>(0,1)
- min <complex>(complex(2,3), complex(3,4))

□ Dacă argumentele corespund tipurilor de date ale template-ului (ex. pentru funcția min ambele argumente sunt de același tip) compilatorul va instanția automat funcția fără a mai fi nevoie ca utilizatorul să specifice explicit tipurile parametrilor

- Exemplu: min(7.8, 7.8)

Functii template

❑ AMBIGUITĂȚI

min (0,1); //OK

min (2.5, 4); //ambiguu; este nevoie de un apel explicit min<double> (2.5, 4)

min (2.5, 4.8); //OK

❑ Rezolvarea ambiguităților

❑ Apelarea explicită

❑ supraîncărcarea / specializarea prin adăugarea unor noi funcții care să se potrivească cu apelul (ex. double min(double, int))

Parametri multipli

❑ Template-urile pot accepta mai multe tipuri generice

❑ Sintaxă

```
template <typename T1, [typename T2[, ... [typename tipN]]]> numeFunctie  
        (listaDeParametrii);  
template < typename T1 [, typename T2[, ... ]]> class nume_clasa { ... }
```

❑ Un număr mare de parametrii poate produce confuzii

❑ Tipul de return dacă este generic trebuie să se regăsească în lista de tipuri

❑ Exemplu

```
template <typename T1, typemane T2> T1 add( T1 a, T2 b);
```

Separarea codului

- ❑ Declararea clasei și definirea metodelor se realizează în fișierele header (.h)

❑ Cauză: **mecanismul de expandare a template-urilor**

- ❑ Declarație metode în fișier header

```
template <typename T> class MyClass {  
    // Folosirea lui T ca un tip obișnuit  
    bool test(T item);  
};
```

- ❑ Definire metode în fișier header

```
template <typename T> bool MyClass<T>::test(T item) {  
    // Folosirea lui T ca un tip obișnuit  
}
```

Clase generice

- ❑ Moștenire
 - ❑ Moștenirea funcționează la fel ca în cazul claselor ‘obișnuite’
- ❑ Parametrii implicați (default)
 - ❑ O valoare implicită poate fi specificată din în definiția template-ului
- ❑ Exemplu

```
template <typename T1, typename T2 = int> class MyClass
{
    ...
}
```



ÎNTREBĂRI

Întrebări- Q1

Care este rezultatul executiei programului?

```
template <class T> void f(T &i) {  
    std::cout << 1;  
}  
template <> void f(const int &i) {  
    std::cout << 2;  
}  
int main() {  
    int i = 42;  
    f(i);  
}  
a) 1  
b) 2  
c) Eroare de compilare  
d) Eroare de executie
```

Întrebări – Q2

Care este rezultatul execuției programului?

```
#include <iostream>
using namespace std;
template <class T> class Test {
private:
    T val;
public:
    static int count;
    Test() { count++; }
};
template<class T> int Test<T>::count = 0;
int main() {
    Test<int> a;
    Test<int> b;
    Test<double> c;
    cout << Test<int>::count << " ";
    cout << Test<double>::count << endl;
    return 0;
}
```

- a) 0 1; b) 0 0; c) 1 1; d) 2 1; e) 1 2

Întrebări – Q3

- Ce cuvânt cheie poate fi folosit ca parametru al unui template?
 - class
 - typename
 - Function
 - Variantele A și B