

PCLP2

Laboratorul 11

1. **Clase template.** *Template-urile (șabloanele) de clase* se mai numesc și *tipuri parametrizate* și se folosesc pentru crearea claselor generice. Pentru adaptarea unei astfel de clase, este nevoie de unul sau mai mulți parametri de tip care transformă clasa dintr-una generică în una particulară. Exemplul de mai jos prezintă implementarea unei structuri de date de tip *stivă* care permite stocarea unor valori, ordinea de citire a acestora fiind inversă ordinii în care au fost înscrise în colecție. Regula *last-in-first-out* specifică stivelor este valabilă indiferent de tipul elementelor care sunt încărcate. Când se pune problema instanțierii unei stive, trebuie specificat tipul de dată al elementelor. Template-urile de clase oferă posibilitatea implementării unei stive generice care poate fi particularizată pentru un tip specific de dată. Studiați programul care implementează o stivă cu elemente de tip `double`. Clasele template folosesc un mecanism particular prin care codul compilat este generat doar în momentul instanțierii, nu se poate separa interfața clasei de implementarea sa în două fișiere `.h` și `.cpp` așa cum procedăm de obicei. Din acest motiv, cele două secțiuni se scriu într-un singur fișier header.

```
stack.h
#ifndef STACK_H
#define STACK_H

template <class T>
class Stack
{
public:
    Stack(int = 10); //stiva are dimensiunea implicita 10
    ~Stack() { delete[] stackPtr; } //destructor
    bool push(const T&); //insereaza un element in stiva
    bool pop(T&); //extrage un element din stiva
private:
    int size; //numarul de elemente din stiva
    int top; //localizarea elementului din varful stivei
    T* stackPtr; //pointer la stiva
    bool isEmpty() const {return top == -1;} //functii
    bool isFull() const {return top == size-1;} //utilitare
};

template<class T>
Stack<T>::Stack(int s)
{
    size = s > 0 ? s : 10;
    top = -1; //initial stiva este goala
    stackPtr = new T[size]; //alocarea spatiului pentru elemente
}
//Introduce un element in stiva
//Intoarce 1 daca s-a putut face inserarea si 0 in caz contrar
template<class T>
bool Stack<T>::push(const T& pushValue)
{
    if(!isFull())
    {
        stackPtr[++top] = pushValue; //plaseaza elementul in stiva
        return true; //inserare realizata cu succes
    }
    return false; //inserarea nu s-a putut realiza
}

//Extrage un element din stiva
```

Programarea calculatoarelor și limbaje de programare 2

```
template<class T>
bool Stack<T>::pop(T& popValue)
{
    if(!isEmpty())
    {
        popValue = stackPtr[top--]; //sterge elementul din stiva
        return true; //extragere realizata cu succes
    }
    return false; //extragerea nu s-a putut realiza
}
#endif
```

test_stack.cpp

```
#include <iostream>

using std::cout;
using std::cin;
using std::endl;

#include "stack.h"

int main()
{
    Stack<double> doubleStack(5);
    double f = 1.1;
    cout << "Inserarea elementelor in doubleStack\n";

    while(doubleStack.push(f))
    {
        cout << f << ' ';
        f += 1.1;
    }

    cout << "\nStiva este plina. "
         << "Nu se mai poate insera elementul " << f
         << "\n\nExtragerea elementelor din doubleStack\n";

    while(doubleStack.pop(f))
        cout << f << ' ';

    cout << "\nStiva este goala. "
         << "Nu se mai pot extrage elemente\n";

    return 0;
}
```

2. Scrieți un program care implementează o nouă structură de date numită *queue* (coadă), asemănător programului de mai sus. Regula după care funcționează o coadă este *first-in-first-out*, adică funcția `pop` va extrage primul element introdus, și nu ultimul, așa cum se întâmpla în cazul stivei.