Template-uri (şabloane)

Obiective:

- Sabloane;
- Clase sablon;
- Definitie;
- Parametri si argumentele sabloanelor;
- Instantierea claselor sablon.

Probleme:

1. a) Creati si implementati clasa generica Stack folosind un tip generic T.

Stack<T>
#ind:int #length:int #t: T*
+Stack() +Stack(dim:int) +~Stack() +add(elem:T) +list():void +empty():bool +full():bool

-atributul length reprezinta capacitatea maxima a stivei
-atributul ind reprezinta cursorul stivei, el ne indica pozitia pe care se afla ultimul element introdus
-atributul t este un T pointer, cu ajutorul lui vom aloca zona de memorie pe stiva
-metoda Stack() este constructorul implicit care initializeaza cele trei atribute:
-lungimea e initializata cu un numar (care vreti voi);
-ind va fi initializat cu -1;
-t va aloca zona de memorie in stiva.
-constructorul explicit Stack(int dim) face acelasi lucru ca si constructorul implicit, dar lungimea este initializata cu dim (cu parametrul primit de constructorul explicit).
-destructorul sterge zona de memorie alocata
-metoda add(T elem) adauga elemente de tipul T in stiva; daca stiva e plina avem eroare;
-metoda list() afiseaza elementele care se afla pe stiva;
-metoda empty() returneaza true daca stiva e goala si false altfel;
-metoda full() returneaza true daca stiva e plina, daca nu returneaza false.

Sintaxa pentru declararea clasei Stack este *template <class T> class Stack*
Cand desfasuram metodele din clasa utilizam: *template <class T> Stack<T>::Stack()*
Cand cream un obiect de tipul clasei Stack: *Stack<int> s=Stack<int>();*

b) Dupa ce ati adaugat elemente de tipul int adaugati elemente de alte tipuri diferite.

c) Apoi adaugati elemente de tipul Punct. Pentru acest lucru e nevoie sa implementati clasa Punct (atribute x si y-retin coordonatele punctului), un constructor care initializeaza x si y si supraincarcati operatorul <<. Iar in main veti avea in plus:

```
Stack<Punct> sp=Stack<Punct>();
Punct p1();
Punct p2(2,3);
Punct p3(3,3);
sp.add(p1);
sp.add(p2);
sp.add(p3);
```

2. Creati un program care sa functioneze corect pentru urmatoarea clasa si pentru urmatoarea functie main:

```
template <class T> class Stack {
public:
T pop();          // extract the element from stack's top
void push(T data);        // insert a new element on top
bool isEmpty();
Stack()
~Stack();
private:
   // specific implementation part
};

int main()
{
Stack <int> anIntegerStack;
anIntegerStack.push(5);
anIntegerStack.push(7);
if(anIntegerStack.isEmpty())
cout << "Stiva goala" << endl;
else
cout << anIntegerStack.pop() << endl;
Stack<char*> route;
route.push(„Timisoara”);
route.push(„Lugoj”);
route.push(„Deva”);
while(route.isEmpty())
cout << route.pop() << „ -> ”;
return 0;
    }
```

Tema

1. Pentru urmatoarea clasa si pentru urmatoarea functie main creati programul care sa functioneze corect:

```
template <class T> class List {
public:
void append (T data); // inserts a new element after the last one
void remove(); // removes the last element
List();
// List traversal opeations
class Iterator {
public:
Iterator();
int operator == (Iterator& x) const;
int operator != (Iterator& x) const;
T operator *() const;
Iterator& operator ++(int);
};
Iterator begin() const;
Iterator end() const;
private:
// list representation
};
int main(int, char*[]) {
List <Point> list;
list.append (Point(1, 1));
list.append (Point(3, 14));
List <Point>::Iterator index = list.begin(), end = list.end();
for(; index != end; index++)
cout << *index << " " << endl;
return 0;
    }
```

Pentru a obtine un iterator pentru inceputul si sfarsitul listei folositi doua metode list.begin() si list.end();
Pentru a obtine urmatorul element supraincarcati operatorul ++;
Pentru a obtine valoarea curenta supraincarcati operatorul *