Templates

Exercises

1. a) Create and implement the class Stack using a generic type T.

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

-the atribute length represents the maximum capacity of the
-the atribute ind indicates the position of the last element added in the stack
-the atribute t is a T pointer, we use it in order to allocate memory on the stack
-the method Stack() is the default constructor which initializes the three atributes:

-the length is initialized with a value (that you choose);
-ind is initialized with -1;
-t will allocate memory on the stack.

-the explicit constructor Stack(int dim) similar with the default constructor, but the length is initialized with dim (the parameter of the explicit constructor).
-the destructor deletes the memory allocated
-the method add(T elem) add elements of the type T on the stack; if the stack is full, then we have an error;
-the method list() lists the elements of the stack;
-the method empty() returns true if the stack is empty and false otherwise;
-the method full() returns true if the stack is full and false otherwise.

The syntax for declaring the class Stack is *template <class T> class Stack*
When we define a method we use *template <class T> Stack<T>::Stack()*
When we take an object as an instance of the class *Stack<int> s=Stack<int>();*

b) After you add elements of the type int, add also elements of different types.

c) Add elements of the type Point. For this is necessary to implement the class Point (the atributes x and y are the coordinates of a point), one constructor which initializes x and y and overload the operator <<.

Add in the function main:

```
Stack<Point> sp=Stack<Point>();
Point p1();
```

```
      Point p2(2,3);
      Point p3(3,3);
      sp.add(p1);
      sp.add(p2);
      sp.add(p3);
```

2. **Create a corresponding program for the following class and function 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 << "Empty stack" << 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;
    }
```

Homework

1. **Create the corresponding program for the following class and function main:**

```
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 operations
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() {
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;
    }
```

In order to obtain an iterator for the beginning and for the end of a list use two methods list.begin() and list.end();
In order to obtain the next element overload the operator ++;
In order to obtain the current value overload the operator *