Laborator 3

Paradigme de programare

Programarea OuOaPe

Primire teme si pus note (10-15 minute)

Testare cunostiintelor de la curs prin intrebari rapide (5 minute)

Task 1 Sa se implementeze cu template o stiva(20 min)

```
#ifndef STACK_H_
#define STACK_H_

template<class T>
class Stack {

private:
	T* buffer;
	int first;
	int capacity;
public:
	Stack(int cap = 10) {
		….
	}

	~Stack() {
		…
	}

	void push(T elem) {
		…
	}

	T pop() {
		…
	}

	T top() {
		…
	}

	bool isEmpty() {
		…
	}
};

#endif /* STACK_H_ */


#include <iostream>
#include "Stack.h"
using namespace std;
```

```
int main() {
	Stack<int> stiva(5);
	stiva.push(1);
	stiva.push(2);
	stiva.push(3);
	cout << stiva.pop()<< endl;
	cout << stiva.top()<< endl;
	cout << stiva.pop()<< endl;
	cout << stiva.isEmpty()<< endl;
	cout << stiva.pop()<< endl;
	cout << stiva.isEmpty()<< endl;
	return 0;
}
```

Task 2 Sa se implementeze cu template lista circulară (20 min)

```
#ifndef LIST_H_
#define LIST_H_

template<class T>
class List {

private:
	template<class E>
	struct Node {
		E data;
		Node * next;
		Node(E data) :
				data(data), next(0) {
		}
	};

	Node<T> *head;
public:
	List() :
			head(0) {
	}

	~List() {
		…
		}
	}

	void addNode(T data) {
		Node<T> * t = new Node<T>(data);
		…
	}
};

#endif /* LIST_H_ */
```

Task 3 Sa se implementeze cu template lista cu numere complexe si operatori supraincarcati ca in curs (20 min)

```
#ifndef COMPLEX_H_
#define COMPLEX_H_

#include <iostream>

using namespace std;

class Complex {
private:
	double re, im;
public:
	Complex() :
			re(0), im(0) {

	}
	Complex(double x, double y) :
			re(x), im(y) {

	}
	Complex & operator +=(const Complex & rhs) {
		…
	}
	friend Complex operator +(const Complex & scr1, const Complex & scr2);
    friend ostream & operator << (ostream & out, const Complex & rhs);
};

Complex operator +(const Complex & src1, const Complex & src2) {
	…
}

ostream & operator << (ostream & out, const Complex & rhs) {
	…
}


#endif /* COMPLEX_H_ */



	//
	List<int> l;
	l.addNode(4);
	l.addNode(5);
	//
	Complex c;
	Complex z1, z2(1, 2), z3(-2, 4);
	z1 = z2 + z3;
	z2 = z2 + z3;
	cout << z2 + z3 << endl;
	cout << z1 << " + " << z2 << " = " << z1 + z2 << endl;
	//
	List<Complex> l2;
	l2.addNode(z1);
	l2.addNode(z2);
	l2.addNode(z3);
```

Task 4 Sa se implementeze hambarul din curs, cateva reguli de interactiune intre elementele din hambar din curs, instantieri aleatoare a unui numar si tip de animale se executa/interactioneaza in clasa curte pe baza regulilor (20 min)

```
#ifndef ANIMALS_H_
#define ANIMALS_H_

#include <string.h>
#include <iostream>
using namespace std;

class Animal {
…
};

#endif /* ANIMALS_H_ */
```

```
	Animal * theBarn[4];
	Animal MickeyMouse("Mickey");
	Cat Sly("Sylvester");
	Dog Fred;
	Dog Spot("Spot");
	theBarn[0] = &MickeyMouse;
	theBarn[1] = &Sly;
	theBarn[2] = &Fred;
	theBarn[3] = &Spot;
	for (int i = 0; i < 4; i++) {
		theBarn[i]->speak();
	}
```

Tema pe acasa: pornind de la model hambar curte interactiuni sa se proiecteze un joc cu nave in spatiu (navele diferite(putere de foc, rezistenta la atac, viteza, manevrabilitate etc) in loc de actiuni grafice se vor afisa mesaje descriptive a ceea c ear trebui sa se vada/intample grafic