MINISTERUL EDUCAȚIEI, CULTURII ȘI CERCETĂRII AL REPUBLICII MOLDOVA
Universitatea Tehnică a Moldovei
Facultatea Calculatoare, Informatică și Microelectronică
Departamentul Informatică și Ingineria Sistemelor

Raport

pentru lucrarea de laborator Nr.2

la cursul de "Supraîncărcarea operatorilor"

Efectuat: Studentul gr. SI-191 **Comanac Artiom**
Verificat: **Mititelu Vitalie**

Chișinău – 2020

LUCRARE DE LABORATOR NR. 2

Tema: Constructorul – funcție de inițializare a obiectelor clasei

Scopul lucrării:

- Studierea necesității supraîncărcării operatorilor
- Studierea sintaxei de definire a operatorilor
- Studierea tipurilor de operatori
- Studierea formelor de supraîncărcare

Varianta 9

a) Să se creeze clasa *Time* – timpul, care conține câmpurile: ore, minute, secunde. Să se definească operatorii "+" și "-", ca funcții prietene, iar "++" și "--" în ambele forme (prefixă și postfixă) ca metode ale clasei. Operatorii trebuie să permită realizarea operațiilor atât cu variabilele clasei date, cât și cu variabilele de tip predefinit int (desemnează secundele).

b) Să se creeze clasa *Queue* – coadă, utilizând memoria dinamică. Să se definească operatorii "+" – de adunare a două cozi, "=" – de atribuire ca metode ale clasei. Să se definească operatorii de comparare "==", "!=", "<", ">" ca funcții prietene. Pentru realizarea ultimilor doi operatori să se definească funcția de calcul a normei elementelor cozii. Să se definească operatorii "<<" și ">>" pentru ieșiri/intrări de obiecte, precum și pentru inserarea/eliminarea elementelor în/din coadă.

Realizarea punctului *a*

main.cpp

```
/*
	Comanac Artiom		SI-191
	LAB #3, a
*/


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

int main() {
	Time t1(20, 00, 00);
	Time t2(01, 30, 00);

	cout << "Standart" << endl;
	cout << "Time A = " << t1.stringify() << endl;
	cout << "Time B = " << t2.stringify() << endl;
	cout << endl;

	Time t1_ = t1++;
```

```
	Time t2_ = t2++;
	cout << "Time++" << endl;
	cout << "Time (return) A = " << t1_.stringify() << ", current A = " <<
t1.stringify() << endl;
	cout << "Time (return) B = " << t2_.stringify() << ", current B = " <<
t2.stringify() << endl;
	cout << endl;


	t1_ = ++t1;
	t2_ = ++t2;
	cout << "++Time" << endl;
	cout << "Time (return) A = " << t1_.stringify() << ", current A = " <<
t1.stringify() << endl;
	cout << "Time (return) B = " << t2_.stringify() << ", current B = " <<
t2.stringify() << endl;
	cout << endl;


	t1_ = t1 + t2;
	cout << "Time + Time" << endl;
	cout << "Time C = " << t1_.stringify() << endl;
	cout << endl;

	t1_ = t1 + 5;
	cout << "Time + Int (5)" << endl;
	cout << "Time C = " << t1_.stringify() << endl;
	cout << endl;

	t1_ = t1 - t2;
	cout << "Time - Time" << endl;
	cout << "Time C = " << t1_.stringify() << endl;
	cout << endl;

	t1_ = t1 - 5;
	cout << "Time - Int (5)" << endl;
	cout << "Time C = " << t1_.stringify() << endl;
	cout << endl;
}
```

Time.h

```
#pragma once

#include <string>
using namespace std;

class Time {
private:
	int hours, minutes, seconds;

public:
	Time(int, int, int);

	Time& operator++(); //++Time
	Time operator++(int); //Time++

	Time& operator--(); //--Time
	Time operator--(int); //Time--

	friend Time operator+(Time&, Time&);  //Time + Time
	friend Time operator+(Time&, int); //Time + int
```

```
        friend Time operator-(Time&, Time&); //Time - Time
        friend Time operator-(Time&, int); //Time - int

        string stringify();

        int toSeconds();
        int toSeconds(Time&);
        static Time toTime(int);
        void setTime(int);
};
```

Time.cpp

```
#include "Time.h"

/*
        Constructor
*/
Time::Time(int hours, int minutes, int seconds) {
        this->hours = hours;
        this->minutes = minutes;
        this->seconds = seconds;
}

/*
        Convert current time (hh:mm:ss) to seconds
*/
int Time::toSeconds() {
        return this->hours * 60 * 60 + this->minutes * 60 + this->seconds;
}

/*
        Convert var time (hh:mm:ss) to seconds
*/
int Time::toSeconds(Time& t) {
        return t.hours * 60 * 60 + t.minutes * 60 + t.seconds;
}

/*
        Convert seconds to time (hh:mm:ss)
*/
Time Time::toTime(int s) {
        if (s < 0)
                s = 0;

        return Time(s / 3600, (s % 3600) / 60, s % 60);
}

/*
        Set current time from seconds
*/
void Time::setTime(int s) {
        if (s < 0)
                s = 0;

        this->hours = s / 3600;
        this->minutes = (s % 3600) / 60;
        this->seconds = s % 60;
}

/*
```

```
	++Time
*/
Time& Time::operator++() {
	int stamp = this->toSeconds();

	this->setTime(++stamp);

	Time _new = this->toTime(stamp);

	return _new;
}

/*
	Time++
*/
Time Time::operator++(int i) {
	int stamp = this->toSeconds();

	Time _old = this->toTime(stamp);
	Time _new = this->toTime(++stamp);

	this->hours = _new.hours;
	this->minutes = _new.minutes;
	this->seconds = _new.seconds;

	return _old;
}

/*
	--Time
*/
Time& Time::operator--() {
	int stamp = this->toSeconds();

	this->setTime(--stamp);

	Time _new = this->toTime(--stamp);

	return _new;
}

/*
	Time--
*/
Time Time::operator--(int i) {
	int stamp = this->toSeconds();

	Time _old = this->toTime(stamp);
	Time _new = this->toTime(--stamp);

	this->hours = _new.hours;
	this->minutes = _new.minutes;
	this->seconds = _new.seconds;

	return _old;
}

/*
	Convert time (hh:mm:ss) to string "H:i:s"
*/
string Time::stringify() {
	//format 0:0:0 to 0:00:00
	return
```

```
            to_string(this->hours)
            + ":" +
            string(2 - to_string(this->minutes).length(), '0') + to_string(this-
>minutes)
            + ":" +
            string(2 - to_string(this->seconds).length(), '0') + to_string(this-
>seconds);
}

/*
      Time + Time
*/
Time operator+(Time& a, Time& b) {
      int stampA = a.toSeconds();
      int stampB = b.toSeconds();

      return Time::toTime(stampA + stampB);
}

/*
      Time + int
*/
Time operator+(Time& a, int offset) {
      int stampA = a.toSeconds();

      return Time::toTime(stampA + offset);
}

/*
      Time - Time
*/
Time operator-(Time& a, Time& b) {
      int stampA = a.toSeconds();
      int stampB = b.toSeconds();

      return Time::toTime(stampA - stampB);
}

/*
      Time - int
*/
Time operator-(Time& a, int offset) {
      int stampA = a.toSeconds();

      return Time::toTime(stampA - offset);
}
```

Demonstrarea

```
Standart
Time A = 20:00:00
Time B = 1:30:00

Time++
Time (return) A = 20:00:00, current A = 20:00:01
Time (return) B = 1:30:00, current B = 1:30:01

++Time
Time (return) A = 20:00:02, current A = 20:00:02
Time (return) B = 1:30:02, current B = 1:30:02

Time + Time
Time C = 21:30:04

Time + Int (5)
Time C = 20:00:07

Time - Time
Time C = 18:30:00

Time - Int (5)
Time C = 19:59:57
```

Realizarea punctului *b*

main.cpp

```
/*
	Comanac Artiom		SI-191
	LAB #3, b
*/


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

int main() {
	Queue q1;
	Queue q2;
	Queue q3;

	q1 << 1 << 2 << 3;
	cout << "Q1: " << q1 << endl;

	int el = 0;
	q1 >> el;
	cout << "Get from Queue: " << el << endl;
```

```
	cout << "Input new element to Q1: ";
	cin >> q1;

	q2 << 4 << 5 << 6;
	cout << "Q2: " << q2 << endl;

	q3 = q1 + q2;
	cout << "Q3 (Q1 + Q2): " << q3 << endl;

	cout << "Q1 == Q2 --- ";

	if (q1 == q2)
		cout << "true" << endl;
	else
		cout << "false" << endl;

	cout << "Q1 != Q2 --- ";

	if (q1 != q2)
		cout << "true" << endl;
	else
		cout << "false" << endl;

	cout << "Q1 > Q2 --- ";

	if (q1 > q2)
		cout << "true" << endl;
	else
		cout << "false" << endl;

	cout << "Q1 < Q2 --- ";

	if (q1 < q2)
		cout << "true" << endl;
	else
		cout << "false" << endl;
}
```

Queue.h

```
#pragma once
#include <iostream>
using namespace std;

class Queue {
public:
	int* elements = nullptr;
	int count = 0;

	Queue();
	Queue(int*, int);

	void push(int);
	int pop();

	Queue operator+(const Queue&); //Queue + Queue
	Queue& operator=(const Queue&); //Queue = Queue

	friend ostream& operator<<(ostream&, Queue&); //cout << Queue
	friend istream& operator>>(istream&, Queue&); //cin >> Queue

	friend Queue& operator<<(Queue&, int); //Queue << int
```

```
	friend Queue& operator>>(Queue&, int&); //Queue >> int

	friend bool operator==(Queue&, Queue&); //Queue == Queue
	friend bool operator!=(Queue&, Queue&); //Queue != Queue
	friend bool operator<(Queue&, Queue&); //Queue < Queue
	friend bool operator>(Queue&, Queue&); //Queue > Queue
};
```

Queue.cpp

```
#include "Queue.h"

/*
	Constructor
*/
Queue::Queue() {
	this->elements = nullptr;
	this->count = 0;
}

/*
	Constructor
*/
Queue::Queue(int* e, int c) {
	this->elements = e;
	this->count = c;
}

/*
	Add to Queue
*/
void Queue::push(int el) {
	int* _new = new int[this->count + 1];
	for (int i = 0; i < this->count; i++) {
		_new[i] = this->elements[i];
	}

	_new[this->count] = el;

	this->elements = _new;
	this->count++;
}

/*
	Get from Queue
*/
int Queue::pop() {
	if (this->count == 0)
		return 0;

	int* _new = new int[this->count - 1];
	for (int i = 1; i < this->count; i++) {
		_new[i-1] = this->elements[i];
	}

	int el = this->elements[0];

	this->elements = _new;
	this->count--;

	return el;
}
```

```
/*
	Queue + Queue
*/
Queue Queue::operator+(const Queue& q) {
	int* _new = new int[this->count + q.count];
	for (int i = 0; i < this->count; i++) {
		_new[i] = this->elements[i];
	}

	for (int i = 0; i < q.count; i++) {
		_new[i + this->count] = q.elements[i];
	}

	return Queue(
		_new,
		this->count + q.count
	);
}

/*
	Queue = Queue
*/
Queue& Queue::operator=(const Queue& q) {
	this->elements = q.elements;
	this->count = q.count;
	return *this;
}

/*
	Queue == Queue
*/
bool operator==(Queue& q1, Queue& q2) {
	if (q1.count != q2.count)
		return false;

	for (int i = 0; i < q1.count; i++) {
		if (q1.elements[i] != q2.elements[i])
			return false;
	}

	return true;
}

/*
	Queue != Queue
*/
bool operator!=(Queue& q1, Queue& q2) {
	return !(q1 == q2);
}

/*
	Queue < Queue
*/
bool operator<(Queue& q1, Queue& q2) {
	return q1.count < q2.count;
}

/*
	Queue > Queue
*/
bool operator>(Queue& q1, Queue& q2) {
	return q1.count > q2.count;
}
```

```
/*
	cout << Queue
*/
ostream& operator<<(ostream& out, Queue& q) {
	if (q.count == 0)
		return out;

	for (int i = 0; i < q.count - 1; i++) {
		out << q.elements[i] << ", ";
	}

	out << q.elements[q.count - 1];

	return out;
}

/*
	cin >> Queue
*/
istream& operator>>(istream& in, Queue& q) {
	int temp;
	in >> temp;

	q.push(temp);

	return in;
}

/*
	Queue >> int
*/
Queue& operator<<(Queue& q, int el) {
	q.push(el);

	return q;
}

/*
	Queue << int
*/
Queue& operator>>(Queue& q, int& el) {
	el = q.pop();

	return q;
}
```

Demonstrarea

```
Q1: 1, 2, 3
Get from Queue: 1
Input new element to Q1: 8
Q2: 4, 5, 6
Q3 (Q1 + Q2): 2, 3, 8, 4, 5, 6
Q1 == Q2 --- false
Q1 != Q2 --- true
Q1 > Q2 --- false
Q1 < Q2 --- false
```

Concluzii

Efectuând aceasta lucrare de laborator au fost obținute cunoștințele in diferite domenii limbajului C++: utilizarea operatorilor, supraîncărcarea operatorilor +, -, ++, --, =, ==, !=, >, <, >>, <<, utilizarea specificatorului friend.