МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ

“БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ”

КАФЕДРА ИИТ

ОТЧЁТ

по лабораторной работе №4

**«ШАБЛОНЫ ФУНКЦИЙ И КЛАССОВ»**

Выполнил:

студент 2 курса

группы ПО-9

Мисиюк Алексей Сергеевич

Проверил:

Козик И. Д.

Порядок выполнения работы.

1. Создать шаблон заданного класса. Определить конструкторы, деструктор, перегруженную операцию присваивания (“=”) и операции, заданные в варианте задания.

2. Написать программу тестирования, в которой проверяется использование шаблона для стандартных типов данных.

3. Выполнить тестирование.

4. Определить пользовательский класс, который будет использоваться в качестве параметра шаблона. Определить в классе необходимые функции и перегруженные операции.

5. Написать программу тестирования, в которой проверяется использование шаблона для пользовательского типа.

6. Выполнить тестирование.

**Вариант 15**

Класс − однонаправленный список list. Дополнительно перегрузить следующие операции:

[] − доступ к элементу в заданной позиции, например:

+ − объединить два списка;

!= − проверка на неравенство.

**Код программы**

**labrab4.cpp**

#include <iostream>

#include "List.h"

#include "Complex.h"

int main()

{

List<int> list1;

list1.push\_back(1);

list1.push\_back(2);

list1.push\_back(3);

List<int> list2;

list2.push\_back(4);

list2.push\_back(5);

List<int> list3 = list1 + list2;

std::cout << "List1: ";

for (int i = 0; i < list1.getSize(); i++) {

std::cout << list1[i] << " ";

}

std::cout << std::endl;

std::cout << "List2: ";

for (int i = 0; i < list2.getSize(); i++) {

std::cout << list2[i] << " ";

}

std::cout << std::endl;

std::cout << "List3: ";

for (int i = 0; i < list3.getSize(); i++) {

std::cout << list3[i] << " ";

}

std::cout << std::endl;

if (list1 != list1) {

std::cout << "List1 and List1 are not equal" << std::endl;

}

else {

std::cout << "List1 and List1 are equal" << std::endl;

}

if (list1 != list2) {

std::cout << "List1 and List2 are not equal" << std::endl;

}

else {

std::cout << "List1 and List2 are equal" << std::endl;

}

if (list1 != list3) {

std::cout << "List1 and List3 are not equal" << std::endl;

}

else {

std::cout << "List1 and List3 are equal" << std::endl;

}

List<Complex> list;

list.push\_back(Complex(1.0, 2.0));

list.push\_back(Complex(3.0, 4.0));

List<Complex> otherList;

otherList.push\_back(Complex(5.0, 6.0));

otherList.push\_back(Complex(7.0, 8.0));

List<Complex> combinedList = list + otherList;

std::cout << "List: ";

for (int i = 0; i < list.getSize(); i++) {

std::cout << list[i].getReal() << "+" << list[i].getImaginary() << "i ";

}

std::cout << std::endl;

std::cout << "Other List: ";

for (int i = 0; i < otherList.getSize(); i++) {

std::cout << otherList[i].getReal() << "+" << otherList[i].getImaginary() << "i ";

}

std::cout << std::endl;

std::cout << "Combined List: ";

for (int i = 0; i < combinedList.getSize(); i++) {

std::cout << combinedList[i].getReal() << "+" << combinedList[i].getImaginary() << "i ";

}

std::cout << std::endl;

if (list != otherList) {

std::cout << "List and Other List are not equal" << std::endl;

}

else {

std::cout << "List and Other List are equal" << std::endl;

}

return 0;

}

**List.h**

#pragma once

template <typename T>

class List {

private:

struct Node {

T data;

Node\* next;

Node(const T& value) : data(value), next(nullptr) {}

};

Node\* head;

Node\* tail;

int size;

public:

List() : head(nullptr), tail(nullptr), size(0) {}

List(const List& other) : head(nullptr), tail(nullptr), size(0) {

Node\* current = other.head;

while (current != nullptr) {

push\_back(current->data);

current = current->next;

}

}

~List() {

clear();

}

List& operator=(const List& other) {

if (this != &other) {

clear();

Node\* current = other.head;

while (current != nullptr) {

push\_back(current->data);

current = current->next;

}

}

return \*this;

}

T& operator[](int index) {

if (index < 0 || index >= size) {

throw std::out\_of\_range("Index out of range");

}

Node\* current = head;

for (int i = 0; i < index; i++) {

current = current->next;

}

return current->data;

}

List operator+(const List& other) {

List result(\*this);

Node\* current = other.head;

while (current != nullptr) {

result.push\_back(current->data);

current = current->next;

}

return result;

}

bool operator!=(const List& other) const {

if (size != other.size) {

return true;

}

Node\* current1 = head;

Node\* current2 = other.head;

while (current1 != nullptr && current2 != nullptr) {

if (current1->data != current2->data) {

return true;

}

current1 = current1->next;

current2 = current2->next;

}

return false;

}

void push\_back(const T& value) {

Node\* newNode = new Node(value);

if (head == nullptr) {

head = newNode;

tail = newNode;

}

else {

tail->next = newNode;

tail = newNode;

}

size++;

}

void clear() {

Node\* current = head;

while (current != nullptr) {

Node\* next = current->next;

delete current;

current = next;

}

head = nullptr;

tail = nullptr;

size = 0;

}

int getSize() const {

return size;

}

};

**Complex.h**

#pragma once

class Complex {

private:

double real;

double imaginary;

public:

Complex(double real = 0.0, double imaginary = 0.0) : real(real), imaginary(imaginary) {}

double getReal() const {

return real;

}

double getImaginary() const {

return imaginary;

}

Complex operator+(const Complex& other) const {

double newReal = real + other.real;

double newImaginary = imaginary + other.imaginary;

return Complex(newReal, newImaginary);

}

bool operator!=(const Complex& other) const {

return (real != other.real) || (imaginary != other.imaginary);

}

};

**Результат**