ГУАП

КАФЕДРА № 14

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

ассистент		Н.Ю. Чумакова
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ

ПЕРЕГРУЗКА ОПЕРАТОРОВ

по курсу: ТЕХНОЛОГИЯ ПРОГРАММИРОВАНИЯ

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. № 1042		Д.А. Васейко
	подпись, дата	инициалы, фамилия

Санкт-Петербург 2021

1. Постановка задачи

Вариант 3:

• Задание 1 Унарная операция

Создать класс вещественных чисел. Необходимо перегрузить следующие операторы: оператор префиксного декрементирования как метод для уменьшения значения на 0,1; оператор префиксного инкрементирования как метод для увеличения значения на 0,5; оператор постфиксного декрементирования как дружественную функцию для уменьшения значения на 0,2; оператор постфиксного инкрементирования как дружественную функцию для увеличения значения на 2,5; операторы ввода и вывода в поток для вывода на экран и ввода с клавиатуры данных в объект соответственно.

• Задание 2. Бинарная операция

Создать класс"Стек", размер стека вводится с клавиатуры пользователем, стек заполняется случайными числами от 10 до 30. Требуется перегрузить арифметические операторы для работы с пользовательским числом (не с экземплярами вашего класса), вводимыми пользователем с клавиатуры: +, -, /, *, +=, -=, /=, *=. Для операторов +, -, * и / число может стоять как слева от оператора, так и справа. На усмотрение студента остается решение какой оператор какой перегрузки требует (метод или дружественная функция). Создать класс вещественных чисел. Определить оператор - как метод и + как дружественную функцию для работы с другими экземплярами вашего класса. Перегрузить операторы сравнения <, >, != и == для работы с другими экземплярами вашего класса.

2. Формализация задачи

Данная программа разбита на пять файлов: два файла первого задания (Stack.h и Stack.cpp), два файла второго задания (Real.h и Real.cpp) и один главный (main.cpp) – это файлы с объявлением, определением методов класса соответственно, а также файл с управляющей функцией.

2.1. Задание 1 – Унарный оператор

Описан класс вещественных чисел Real, приватным полем в нем является float numb, в публичном поле объявлены необходимые конструкторы согласно заданию, также перегружены операторы, добавлены геттеры и сеттеры, а также метод вывода на экран.

В главной функции main() объявляются и инициализируется с помощью конструктора с параметрами объекты класса Real: num1, num2. Реализовано также пользовательское меню, в котором можно выбрать необходимую операцию.

2.2.Задание 2 – Бинарный оператор

Элемент стека, реализованный с помощью структуры под именем Unit, состоит из данных целочисленного типа, а также указателя на следующий такой же элемент.

Число элементов стека заранее неизвестно и заполняется методом random. В программе создается объект класса Стек, инициализируемый конструктором без параметров.

В классе стека прописаны функции вставки, удаления, вывода на экран. Также написаны конструкторы согласно зданию: с параметром (задаем размер); без параметра; конструктор копирования, который реализован с помощью буферного массива; деструктор. Прочие же операторы реализованы согласно заданию.

В главной функции main() реализовано пользовательское меню с возможностью выбора операции над стеком: добавить элемент, удалить элемент, вывести стек на экран, провести ту или иную операцию, перейти ко второму заданию.

3. Исходный код

Файл Stack.h

```
#pragma once

typedef struct Unit
{
	int data;
	Unit* next;
} Unit; //элемент стека

//Unit* top; //вершина стека

class Stack
{
private:
	Unit* top = 0; //указатель на начальный элемент стека - инициализированный элемент
	int size = 0; //размер стека
public:
	//конструкторы
	Stack(); //конструктор без параметров
	explicit Stack(int size_st); //явный конструктор с параметрами (задаем размер стека)
	~Stack(); //деструктор
	Stack(const Stack& Stack_copy); //конструктор копирования

	//функции стека
	void push(int value); //добавление элемента в стек
	int pop(); //извлечение вершины
	void print(); //вывод стека на экран

	//геттеры и сеттеры
	Unit* get_top(); //взять элемент стека
	void set_size(int& value); //поместить значение в стек
	int get_size() const; //взять размер стека
	Stack& method(int l, Stack& r, int Var);
	//перегруженные операторы
	Stack& operator *(int r);
	friend Stack& operator *(int l, Stack& r);
	Stack& operator /(int r);
	friend Stack& operator /(int l, Stack& r);
	Stack& operator +(int r);
	friend Stack& operator +(int l, Stack& r);
	Stack& operator -(int r);
	friend Stack& operator -(int l, Stack& r);
	Stack& operator *=(int r);
	Stack& operator /=(int r);
	Stack& operator +=(int r);
	Stack& operator -=(int r);

};
```

Файл Stack.cpp

```
#include "Stack.h"
#include <iostream>


using namespace std;
```

```
/* КОНСТРУКТОРЫ */

Stack::Stack() : top(0), size(0) {} //конструктор без параметров (исп. список инициализации)

Stack::Stack(int size_st) : top(0)//конструктор с параметром, задает размер и инициализирует
элемент
{
	srand(time(NULL));
	for (int i = 0; i < size_st; i++)
		this->push(rand()%21+10);
}

Stack::~Stack()
{
	if ((top == 0) || (size == 0))
	{
		delete top; //удаляем вершину
		size = 0; //размер нулевой
	}
	else
	{
		while (top->next != 0) //пока не дойдем до нулевого указателя
		{
			Unit* temp = top; //временная переменная юнита, присваиваем ей значение
вершины
			top = temp->next; //теперь вершина указывает на следующий за ней
элемент,
			delete temp; //а временную - удаляеям
		}
		delete top; //удаляем вершину
		size = 0; //размер нулевой
	}
}

Stack::Stack(const Stack& Stack_copy) : size(Stack_copy.size)
{
	int* buff = new int[Stack_copy.size]; //вводим буферный массив, в котором будем
хранить наши числа нового стека
	Unit* el_stack = Stack_copy.top; //новая переменная-вершина будущего стека

	for (int i = 0; i <= Stack_copy.size - 1; i++)
	{	//пока счетчик не дойдет до конца стека
		buff[i] = el_stack->data; //заносим в массив значения из стека
		el_stack = el_stack->next; //ставим указатель на следующий элемент стека
	}

	for (int i = 0; i < Stack_copy.size; i++)
		this->push(buff[i]); //вносим в новый стек значения из массива

	delete[] buff; //удаляем буфер
}

/* МЕТОДЫ КЛАССА СТЕКА */

void Stack::push(int value)
{
	Unit* new_unit = new Unit; //инициализируем новый юнит стека и выделяем под него
память
	new_unit->next = top; //теперь новый юнит стал вершиной
	top = new_unit; //и вершина указывает на новый юнит
	top->data = value; //вносим значение в вершину
	size++;
}
```

```
int Stack::pop()
{
	setlocale(LC_ALL, "Rus");

	if (!this)
	{
		cout << "Стек пуст, удалять нечего." << endl;
	}

	Unit* temp = top; //временная переменная-вершина
	int deldata = 0; //место под удаленный элемент

	deldata = top->data; //присваиваем значение удаляемого элемента
	top = top->next; //вершина теперь тот юнит, который был под вершиной
	delete temp; //удаляем переменную-вершину
	size--;
	return deldata; //вернули удаленный элемент
}

void Stack::print()
{
	if (size == 0)
	{
		cout << "Стек пуст, выводить нечего." << endl;
		return;
	}
	else
	{
		Unit* temp = top;
		while (temp->next != nullptr)
		{
			cout << temp->data << " <-- ";
			temp = temp->next;
		}

		cout << temp->data << endl;
	}
}

Unit* Stack::get_top()
{
	return top;
}

int Stack::get_size() const
{
	return size;
}

void Stack::set_size(int& value)
{
	size = value;
}

Stack& Stack::operator *(int r)
{
	return method(r, *this, 0);
}
Stack& Stack::operator /(int r)
{
	return method(r, *this, 1);
}
Stack& Stack::operator +(int r)
{
```

```
	return method(r, *this, 2);
}
Stack& Stack::operator -(int r)
{
	return method(r, *this, 3);
}
Stack& Stack::method(int l, Stack& r, int Var)
{
	Unit* tmp = this->top;

	if (size == 0)
	{
		cout << "Стек пуст, невозможно провести операцию!" << endl;
	}
	else
	{
		while (tmp != nullptr)
		{
			tmp = tmp->next;
		}
	}
	int len = this->size; //переменная размера вызванного объекта
	int* buff = new int[len]; //буферный массив, куда занесем значения стека
	for (int i = 0; i < len; i++)
		buff[i] = this->pop();
	//вносим в массив значения стека, стек на время становится пустым
	for (int i = len - 1; i >= 0; i--)
		switch (Var)
		{
		case 0:
			this->push(buff[i] * l); //в обратном порядке (как было до
преобразования) заносим новые значения в стек
			break;
		case 1:
			this->push(buff[i] / l); //в обратном порядке (как было до
преобразования) заносим новые значения в стек
			break;
		case 2:
			this->push(buff[i] + l); //в обратном порядке (как было до
преобразования) заносим новые значения в стек
			break;
		case 3:
			this->push(buff[i] - l); //в обратном порядке (как было до
преобразования) заносим новые значения в стек
			break;
		/*
		* case 4:
		* this->push(buff[i]);
		* l-=buff[i]
		* break;
		* тожу самое только другой тип возвращаемого значения
		*/
		}



	delete[] buff;
	return *this;


}

Stack& operator *(int l, Stack& r)
{
```

```
	return r * l;
}
Stack& operator /(int l, Stack& r)
{
	cout << "U can't  num / stack, stack returned, check Stack .cpp" << endl;
	return r;// check method()
	//Еще был вариант поделить все число на каждый элемент стека, но это как-то странно
}
Stack& operator +(int l, Stack& r)
{
	return r + l;
}
Stack& operator -(int l, Stack& r)
{
	cout << "U can't  do num - stack, stack returned, check Stack .cpp" << endl;
	return r;
}


Stack& Stack::operator *=(int r)
{
	return method(r, *this, 0);
}
Stack& Stack::operator /=(int r)
{
	return method(r, *this, 1);
}
Stack& Stack::operator +=(int r)
{
	return method(r, *this, 2);
}
Stack& Stack::operator -=(int r)
{
	return method(r, *this, 3);

}
```

Файл Real.h

```
#pragma once

class Real
{
private:
	float numb;
public:
	Real();
	explicit Real(float value);
	~Real();
	Real(const Real& real_copy);
	void set_num(float& num);
	float get_num();
	Real& operator=(float r);
	//Real& operator=(const Real& r);
	Real& operator --(); //префиксная форма
	friend Real& operator --(Real& r, int); //постфиксная форма
	Real& operator ++(); //префиксная форма
	friend Real& operator ++(Real& r, int); //постфиксная форма
	void print();

};
```

Файл Real.cpp

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

```
Real::Real() : numb(0.0){}
Real::Real(float num)
{
	numb = num;
}

Real::~Real() {}
Real::Real(const Real& real_copy)
{
	this->numb = real_copy.numb;
}

Real& Real::operator= (float r)
{
	set_num(r);
	return *this;
}

//Real& Real::operator= (const Real& r)
//{
//	if (this == &r) //проерка на самоприсваивание
//		return *this;
//	numb = r.numb;
//	return *this;
//}

Real& Real :: operator --()
{
	numb -= 0.1;
	return *this;
}
Real& Real:: operator ++()
{
	numb += 0.5;
	return *this;
}

Real& operator ++(Real& r, int) {
	Real temp(r);
	r.numb += 2.5;
	return temp;
}

Real& operator --(Real& r, int) {
	Real temp(r);
	r.numb -= 0.2;
	return temp;
}

void Real::set_num(float &value) { numb = value; }
void Real::print() { cout << numb << endl; }
float Real::get_num()
{
	return this->numb;

}
```

Файл main.cpp

```
#include "Stack.h"
#include "Real.h"
#include <iostream>
#include <locale>

using namespace std;
```

```
int main()
{
	setlocale(LC_ALL, "Rus");

	int stack_size = 0, value = 0; //переменные размера стека и вводимых данных
	int flag = 1, c;

	//////////////////////////////////////////////////////ЗАДАНИЕ №2

	cout << "Вас приветствует программа 2-ой лабораторной по ТП!" << endl;
	cout << "Задание №2 - Бинарные операции (Стек)\nВведите размер стека: ";
	cin >> stack_size;
	system("cls");

	Stack st;
	srand(time(NULL));
	for (int i = 0; i < stack_size; i++)
		st.push(rand() % 21 + 10);

	while (flag == 1)
	{
		cout << "Выберите, какие операции провести со стеком:" << endl;
		cout << "1 - Добавление нового элемента в стек" << endl;
		cout << "2 - Удаление элемента из стека" << endl;
		cout << "3 - Вывод стека на экран" << endl;
		cout << "4 - Оператор умножения (стек слева)" << endl;
		cout << "5 - Оператор умножения (стек справа)" << endl;
		cout << "6 - Оператор деления (стек слева)" << endl;
		cout << "7 - Оператор деления (стек справа)" << endl;
		cout << "8 - Оператор сложения (стек слева)" << endl;
		cout << "9 - Оператор сложения (стек справа)" << endl;
		cout << "10 - Оператор вычитания (стек слева)" << endl;
		cout << "11 - Оператор вычитания (стек справа)" << endl;
		cout << "12 - Остальные операторы в том же порядке (*=,/=,+=,-=)" << endl;
		cout << "0 - Переход ко второму заданию работы" << endl;
		cout << "--> ";
		cin >> c;

		switch (c)
		{
		case 1: //1 - Добавление нового элемента в стек
			system("cls");
			cout << "Введите значение: " << endl;
			cin >> value;
			st.push(value);
			cout << "Значение добавлено в стек\n " << endl;
			break;
		case 2: //2 - Удаление элемента из стека
			if (!st.get_size())
			{
				cout << "Стек пуст, извлекать нечего.\n " << endl;
				break;
			}
			else
			{
				system("cls");
				value = st.pop();
				cout << "Извлеченный элемент: " << value << endl;
			}
			break;
		case 3: //3 - Вывод стека на экран
			if (!st.get_size())
			{
```

```
                cout << "Стек пуст, выводить нечего.\n" << endl;
                break;
        }
        else
        {
                system("cls");
                st.print();
                cout << "\n";
        }
        break;
case 4: //4 - оператор умножения (стек слева)
        if (!st.get_size())
        {
                cout << "Стек пуст, невозможно провести операцию.\n" << endl;
                break;
        }
        else
        {
                system("cls");
                cout << "Введите число, на которое хотите умножить: ";
                cin >> value;
                st = st * value;
                cout << "Операция прошла успешно! Получившийся стек: " << endl;
                st.print();
                cout << "\n";
        }
        break;
case 5: //5 - оператор умножения (стек справа)
        if (!st.get_size())
        {
                cout << "Стек пуст, невозможно провести операцию.\n" << endl;
                break;
        }
        else
        {
                system("cls");
                cout << "Введите число, которое хотите умножить: ";
                cin >> value;
                st = value * st;
                cout << "Операция прошла успешно! Получившийся стек: " << endl;
                st.print();
                cout << "\n";
        }
        break;
case 6: //6 - оператор деления (стек слева)
        if (!st.get_size())
        {
                cout << "Стек пуст, невозможно провести операцию.\n" << endl;
                break;
        }
        else
        {
                system("cls");
                cout << "Введите число, на которое хотите поделить: ";
                cin >> value;
                st = st / value;
                cout << "Операция прошла успешно! Получившийся стек: " << endl;
                st.print();
                cout << "\n";
        }
        break;
case 7: //4 - оператор деления (стек справа)
        if (!st.get_size())
        {
```

```
                cout << "Стек пуст, невозможно провести операцию.\n" << endl;
                break;
        }
        else
        {
                system("cls");
                cout << "Введите число,  которое хотите поделить: ";
                cin >> value;
                st = value / st;
                st.print();
                cout << "\n";
        }
        break;
case 8: //8- оператор сложения (стек слева)
        if (!st.get_size())
        {
                cout << "Стек пуст, невозможно провести операцию.\n" << endl;
                break;
        }
        else
        {
                system("cls");
                cout << "Введите число, с которым хотите сложить: ";
                cin >> value;
                st = st + value;
                cout << "Операция прошла успешно! Получившийся стек: " << endl;
                st.print();
                cout << "\n";
        }
        break;
case 9: //9 - оператор умножения (стек справа)
        if (!st.get_size())
        {
                cout << "Стек пуст, невозможно провести операцию.\n" << endl;
                break;
        }
        else
        {
                system("cls");
                cout << "Введите число, с которым хотите сложить: ";
                cin >> value;
                st = value + st;
                cout << "Операция прошла успешно! Получившийся стек: " << endl;
                st.print();
                cout << "\n";
        }
        break;
case 10: //10 - оператор вычитания (стек слева)
        if (!st.get_size())
        {
                cout << "Стек пуст, невозможно провести операцию.\n" << endl;
                break;
        }
        else
        {
                system("cls");
                cout << "Введите число, на которое хотите умножить: ";
                cin >> value;
                st = st - value;
                cout << "Операция прошла успешно! Получившийся стек: " << endl;
                st.print();
                cout << "\n";
        }
        break;
```

```
	case 11: //10 - оператор вычитания (стек справа)
		if (!st.get_size())
		{
			cout << "Стек пуст, невозможно провести операцию.\n" << endl;
			break;
		}
		else
		{
			system("cls");
			cout << "Введите число, на которое хотите умножить: ";
			cin >> value;
			st = value - st;
			st.print();
			cout << "\n";
		}
		break;
	case 12: //12 - остальное (=)
		if (!st.get_size())
		{
			cout << "Стек пуст, невозможно провести операцию.\n" << endl;
			break;
		}
		else
		{
			system("cls");
			cout << "Введите число, с которым хотите провести действия: ";
			cin >> value;
			st *= value;
			st.print();
			st /= value;
			st.print();
			st += value;
			st.print();
			st -= value;
			st.print();
			cout << "Операция прошла успешно!" << endl;
			cout << "\n";
		}
		break;
	case 0: //0 - Выход
		cout << "\n";
		flag = 0;
		break;
	default:
		cout << "\n";
		flag = 0;
		break;
	}
}

//////////////////////////////////////////////////////////ЗАДАНИЕ №1
system("cls");

float input_num1;
Real num1, num2;
num2 = 0;

cout << "Задание №1 - Унарные операторы\n" << endl;
flag = 1;
cout << "Введите свое число:";
cin >> input_num1;
num1 = input_num1;
while (flag == 1)
{
```

```
		cout << "Выберите, какие операции провести с классом вещественных чисел:" <<
endl;
		cout << "1 - ++ префиксная (+=0.5)" << endl;
		cout << "2 - ++ постфиксная (+=2,5)" << endl;
		cout << "3 - -- префиксная (-=0.1)" << endl;
		cout << "4 - -- постфиксная (-=0,2)" << endl;
		cout << "0 - Выход" << endl;
		cout << "--> ";
		cin >> c;

		switch (c)
		{
		case 1: //1 - ++num
			system("cls");
			num2 = ++num1;
			cout << "num2 = " << num2.get_num() << ", num1 = " << num1.get_num() <<
endl;
			break;
		case 2: //2 -num++
			system("cls");
			num2 = num1++;
			cout << "num2 = " << num2.get_num() << ", num1 = " << num1.get_num() <<
endl;
			break;
		case 3: //3 - --num
			system("cls");
			num2 = --num1;
			cout << "num2 = " << num2.get_num() << ", num1 = " << num1.get_num() <<
endl;
			break;
		case 4: //4 - num--
			system("cls");
			num2 = num1--;
			cout << "num2 = " << num2.get_num() << ", num1 = " << num1.get_num() <<
endl;
			break;
		case 0: //0 - Выход
			cout << "\n";
			flag = 0;
			break;
		default:
			cout << "\n";
			flag = 0;
			break;
		}
	}

	return 0;
}
```

4. Результаты работы программы

4.1. Задание 2 – Бинарные операторы

При включении программы, нам предлагают ввести количество элементов в стеке. Их будет пять. (Рис. 1)

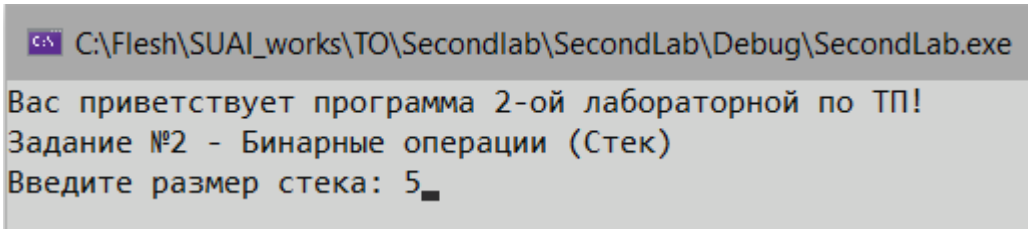

Рисунок 1 – Выбор количества элементов.

Затем перед нами появится основное меню работы. Выведем стек на экран (Рис. 2)

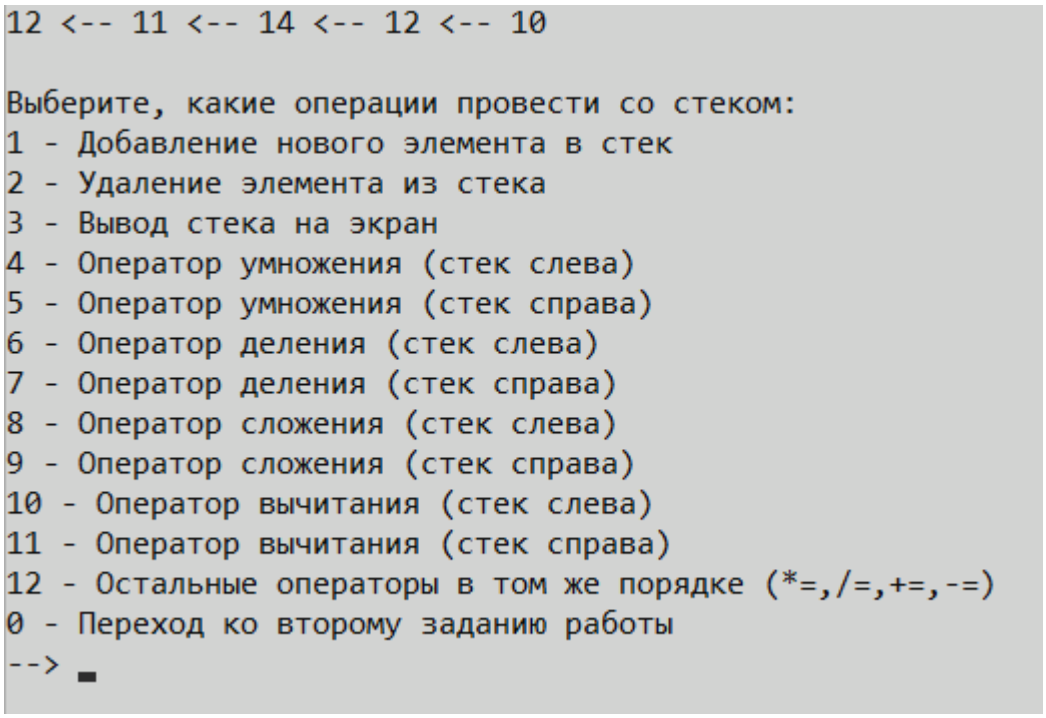

Рисунок 2 – Вывод стека.

Удалим последний элемент для продолжения работы (Рис. 3)

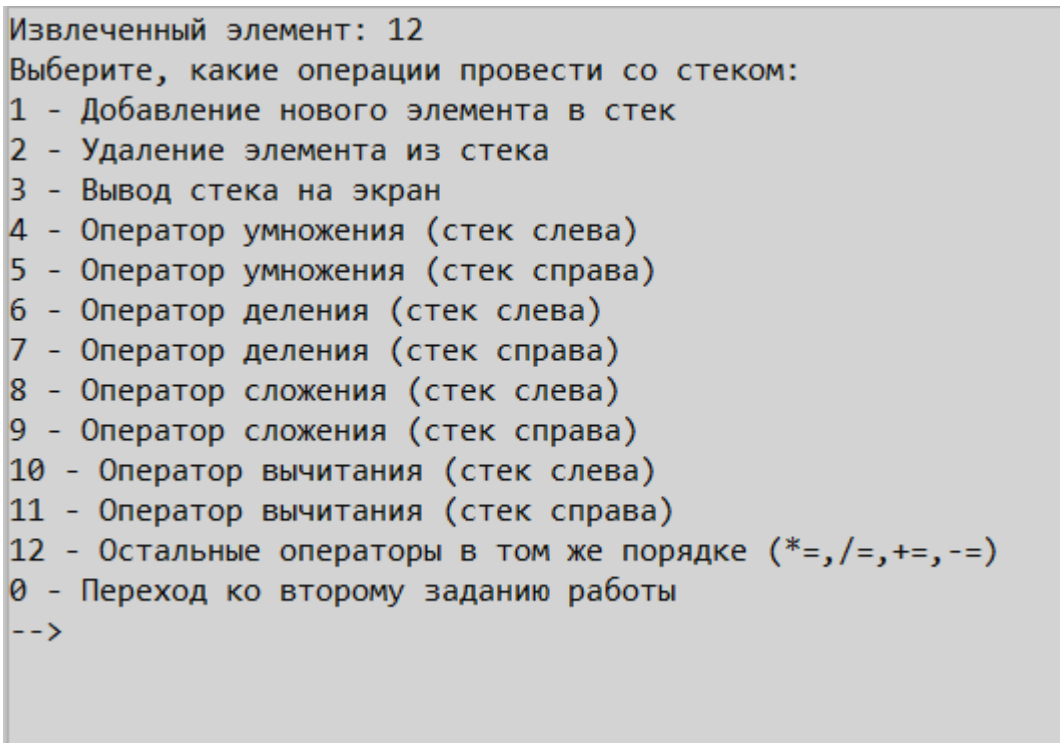

Рисунок 3 – Извлеченный элемент.

Давайте проверим, как работает перегрузка операторов. Проведем со стеком `11 <-- 14 <-- 12 <-- 10` операцию под номером 4 – умножение (стек слева). (рис.4)

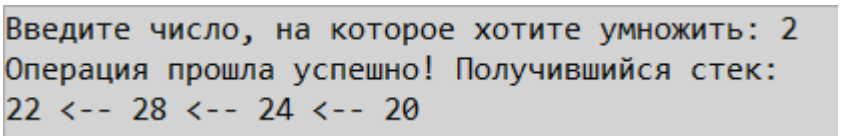

Рисунок 4 – умножение (стек слева).

Проведем с получившимся стеком операцию под номером 5 – умножение (стек справа)(Рис. 5)

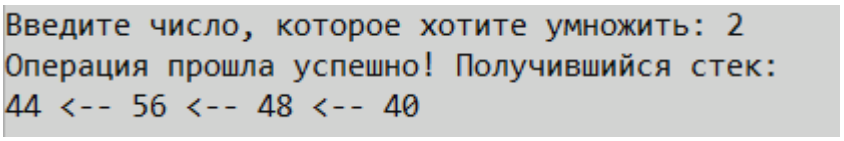

Рисунок 5 – умножение (стек справа).

Проведем с получившимся стеком операцию под номером 6 – деление (стек слева). (Рис. 6)

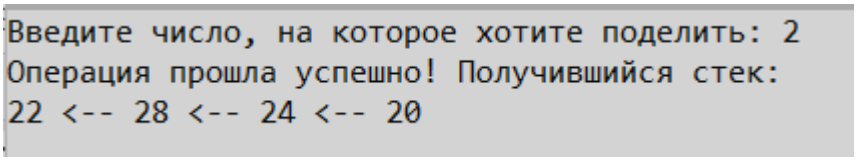

Рисунок 6 – деление (стек слева).

Проведем с получившимся стеком операцию под номером 7 – Оператор деления (стек справа). (Рис. 7)

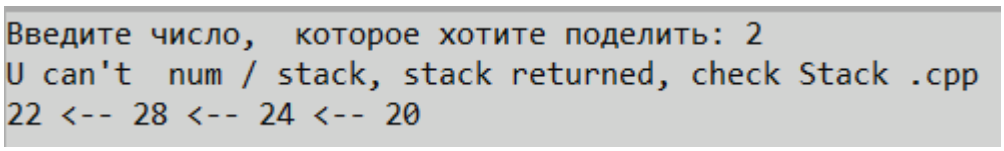

Рисунок 7 – Оператор деления (стек справа).

Проведем с получившимся стеком операцию под номером 8 – Оператор сложения (стек слева). (Рис. 8)

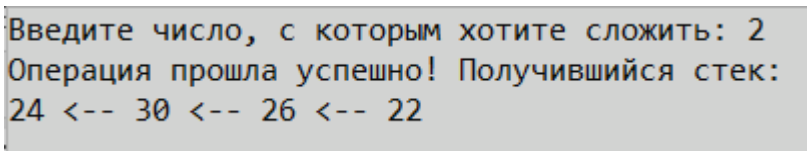

Рисунок 8 – Оператор сложения (стек слева).

Проведем с получившимся стеком операцию под номером 9 – Оператор сложения (стек справа). (Рис. 9)

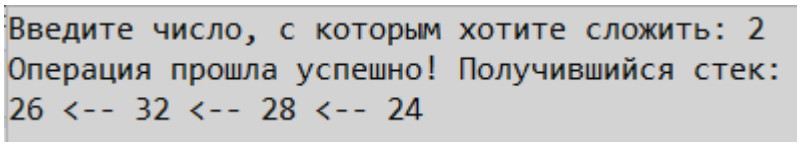

Рисунок 9 – Оператор сложения (стек справа).

Проведем с получившимся стеком операцию под номером 10 – Оператор вычитания (стек слева). (Рис. 10)

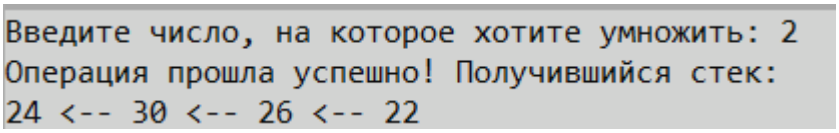

Рисунок 10 – Оператор вычитания (стек слева).

Проведем с получившимся стеком операцию под номером 11 – Оператор вычитания (стек справа). (Рис. 11)

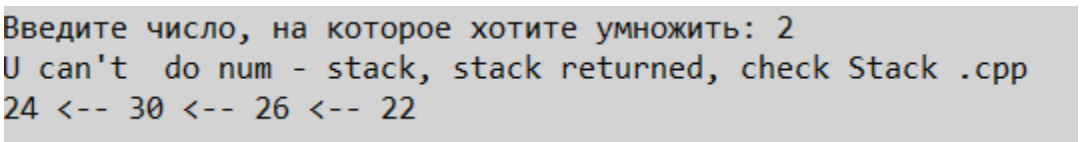

Рисунок 11 – Оператор вычитания (стек справа).

Проведем с получившимся стеком операцию под номером 12 – Остальные операторы в том же порядке (*=,/=,+=,-=). (Рис. 12)

```
Введите число, с которым хотите провести действия: 2
48 <-- 60 <-- 52 <-- 44
24 <-- 30 <-- 26 <-- 22
26 <-- 32 <-- 28 <-- 24
24 <-- 30 <-- 26 <-- 22
Операция прошла успешно!
```

Рисунок 12 – Остальные операторы в том же порядке (*=,/=,+=,-=). (Рис. 12)

4.2. Задание 1 – Унарный оператор

Нажмем ноль и перейдем к выполнению второго задания (Рис. 13)

```
Задание №1 - Унарные операторы

Введите свое число:7.1
Выберите, какие операции провести с классом вещественных чисел:
1 - ++ префиксная (+=0.5)
2 - ++ постфиксная (+=2,5)
3 - -- префиксная (-=0.1)
4 - -- постфиксная (-=0,2)
0 - Выход
-->
```

Рисунок 13 – Второе задание.

Поочередно проведем все операции. Префиксное сложение (Рис. 14)

```
num2 = 7.6, num1 = 7.6
Выберите, какие операции провести с классом вещественных чисел:
1 - ++ префиксная (+=0.5)
2 - ++ постфиксная (+=2,5)
3 - -- префиксная (-=0.1)
4 - -- постфиксная (-=0,2)
0 - Выход
-->
```

Рисунок 14 – Префиксное сложение.

Постфиксное сложение (Рис. 15)

```
num2 = 7.6, num1 = 10.1
Выберите, какие операции провести с классом вещественных чисел:
1 - ++ префиксная (+=0.5)
2 - ++ постфиксная (+=2,5)
3 - -- префиксная (-=0.1)
4 - -- постфиксная (-=0,2)
0 - Выход
-->
```

Рисунок 15 – Постфиксное сложение.

Префиксное вычитание (Рис. 16)

```
num2 = 10, num1 = 10
Выберите, какие операции провести с классом вещественных чисел:
1 - ++ префиксная (+=0.5)
2 - ++ постфиксная (+=2,5)
3 - -- префиксная (-=0.1)
4 - -- постфиксная (-=0,2)
0 - Выход
-->
```

Рисунок 16 – Префиксное вычитание.

Постфиксное вычитание (Рис. 17)

```
num2 = 10, num1 = 9.8
Выберите, какие операции провести с классом вещественных чисел:
1 - ++ префиксная (+=0.5)
2 - ++ постфиксная (+=2,5)
3 - -- префиксная (-=0.1)
4 - -- постфиксная (-=0,2)
0 - Выход
-->
```

Рисунок 17 – Постфиксное вычитание.

5. Выводы

В процессе выполнения лабораторной работы мы изучили перегрузку операторов унарных и бинарных при помощи методов класса и дружественных функций. Были описаны классы стека и вещественных чисел. Программа работает исправно согласно введенным примерам.