МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №4

по дисциплине «Программирование»

Тема: Динамические структуры данных.

Студентка гр. 1304 _____ Чернякова В.А.

Преподаватель _____ Чайка К.В.

Санкт-Петербург

2022

Цель работы.

Освоение работы с динамическими структурами данных. Научиться реализовывать классы.

Задание.

Стековая машина.

Требуется написать программу, которая последовательно выполняет подаваемые ей на вход арифметические операции над числами с помощью стека на базе **списка**.

1) Реализовать **класс** CustomStack, который будет содержать перечисленные ниже методы. Стек должен иметь возможность хранить и работать с типом данных **int**.

Структура класса узла списка:

```
struct ListNode {
   ListNode* mNext;
   int mData;
};
```

Объявление класса стека:

```
class CustomStack {
public:
// методы push, pop, size, empty, top + конструкторы, деструктор
private:
// поля класса, к которым не должно быть доступа извне
protected:  // в этом блоке должен быть указатель на голову
   ListNode* mHead;
};
```

Перечень методов класса стека, которые должны быть реализованы:

- **void push(int val)** - добавляет новый элемент в стек
- **void pop()** - удаляет из стека последний элемент
- **int top()** - доступ к верхнему элементу
- **size_t size()** - возвращает количество элементов в стеке

- **bool empty()** - проверяет отсутствие элементов в стеке

2) Обеспечить в программе считывание из потока **stdin** последовательности (не более 100 элементов) из чисел и арифметических операций (+, -, *, / (деление нацело)) разделенных пробелом, которые программа должна интерпретировать и выполнить по следующим правилам:

- Если очередной элемент входной последовательности - число, то положить его в стек,
- Если очередной элемент - знак операции, то применить эту операцию над двумя верхними элементами стека, а результат положить обратно в стек (следует считать, что левый операнд выражения лежит в стеке глубже),
- Если входная последовательность закончилась, то вывести результат (число в стеке).

Если в процессе вычисления возникает ошибка:

- например, вызов метода **pop** или **top** при пустом стеке (для операции в стеке не хватает аргументов),
- по завершении работы программы в стеке более одного элемента,

программа должна вывести "**error**" и завершиться.

Примечания:

1. Указатель на голову должен быть protected.
2. Подключать какие-то заголовочные файлы не требуется, всё необходимое подключено.
3. Предполагается, что пространство имен std уже доступно.
4. Использование ключевого слова using также не требуется.
5. Структуру **ListNode** реализовывать самому не надо, она уже реализована.

Пример:

*Исходная последовательность: 1 -10 - 2 **

Результат: 22

Выполнение работы.

Главная функция int main().

В основной функции программы для хранения объекта класса *CustomStack* выделяется память. Далее с помощью функции стандартной библиотеки *scanf()* считывается строка со значениями, которые необходимо обработать. Цикл *while()* осуществляет работу до тех пор, пока не будет достигнут конец строки. Далее в цикле осуществляется проверка, является ли текущей элемент строки числом. Если да, то данное число будет добавлено в стек, иначе с верху стека извлекаются при возможности первые два элемента и согласно данному символу-операции с помощью математических действий создается новый элемент, который будет добавлен в стек. Далее считывается новый элемент. По завершении цикла происходит проверка: сколько элементов осталось в стеке. Если число элементов не равно 1, на экран выводится сообщение об ошибке, в противном случае будет выведено значение элемента.

Класс.

Класс CustomStack().

В данном классе первыми объявляются методы со спецификатором *public* – то есть те методы и поля класса, которые будут доступны любым функциям, взаимодействующим с объектом данного класса. Метод *push()* позволяет добавлять новый элемент в стек. Метод *pop()* позволяет удалять элемент из стека, который является верхним. Метод *top()* позволяет получить значение самого верхнего элемента списка. С помощью метода *size()* можно узнать количество элементов, которые находятся в стеке. Проверку на пустоту стека осуществляет метод *empty()*. Чтобы узнать, содержится в строке число или математическая операция, реализован метод *check_number()*. В методе *check()* реализована проверка на наличие хотя бы одного элемента в стеке с помощью описанного ранее метода *empty()*, если нет, происходит вызов метода *appear_error()*, выводящий сообщение об ошибке и осуществляющий завершение работы программы. Удаление всех

элементов деструктором осуществляется методом *pop()*. В спецификаторе *protected*, в котором следует методы и классы поля, доступные как внутри класса, так и в наследуемых, содержит единственное поле указатель на голову списка *mHead*.

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	1 2 + 3 4 - 5 * +	-2	Программа работает корректно.
2.	1 + 5 3 -	error	Программа работает корректно.
3.	-12 -1 2 10 5 -14 17 17 * - - + - * +	304	Программа работает корректно.

Выводы.

В ходе лабораторной работы была освоена работа с динамическими структурами данных. Изучены классы и методы работы с ними. Написана программа, которая последовательно выполняет подаваемые ей на вход арифметические операции над числами с помощью стека на базе списка.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: Chernyakova_Valeria_lb4/main.c

```
class CustomStack{
    public:
        ~CustomStack(){
            while (mHead){
                pop();
            }
        }
        void appear_error(){
            printf("error\n");
            delete this;
            exit(0);
        }
        void check(){
            if (empty()){
                appear_error();
            }
        }
        void push(int val){
            ListNode* newe = new ListNode{mHead, val};
            mHead = newe;
        }
        void pop(){
            check();
            ListNode* t = mHead;
            mHead = mHead->mNext;
            delete t;
        }
        int top(){
            check();
            return mHead->mData;
        }
        size_t size(){
            ListNode* actual = mHead;
            size_t n;
            for (n = 0; actual; n++){
                actual = actual->mNext;
            }
            return n;
        }
        bool empty(){
            return !((bool)mHead);
        }
        int check_number(char* s){
            int l = strlen(s);
            if (l == 1 && !isdigit(s[0])){
```

```
                    return 0;
            }
            return 1;
        }
        protected:
            ListNode* mHead;
};

int main(){
    CustomStack* stack = new CustomStack();
    char actual[100];
    int f, s;
    scanf("%s", actual);
    while (!feof(stdin)){
        if (stack->check_number(actual)){
            stack->push(atoi(actual));
        }
        else {
            f = stack->top();
            stack->pop();
            s = stack->top();
            stack->pop();
            if (!strcmp(actual, "+")){
                    stack->push(f + s);
            }
            else if (!strcmp(actual, "-")){
                    stack->push(s - f);
            }
            else if (!strcmp(actual, "*")){
                    stack->push(f * s);
            }
            else if (!strcmp(actual, "/")){
                    stack->push(s / f);}
        }
        scanf("%s", actual);
    }
    if (stack->size() != 1){
        stack->appear_error();
    }
    f = stack->top();
    printf("%d\n", f);
    return 0;
}
```