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

Студент гр. 3344 _____ Сербиновский Ю.М.
Преподаватель _____ Глазунов С.А.

Санкт-Петербург
2024

Цель работы

Получить представление о работе с ООП на языке C++. Научиться реализовывать стек при помощи класса.

Задание.

Стековая машина.

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

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

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 массив данных

    int* mData;
};
```

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
- **extend(int n)** - расширяет исходный массив на n ячеек

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 массив 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.

Выполнение работы

Был создан класс CustomStack. API класса описан в задании.

Сначала считывается строка входных данных, после чего разбивается на токены. Далее в цикле в зависимости от того, значения токена происхоидт следующее: если токен — арифметический знак, то удаляются два последних элемента списка заменяются на результат арифметической операции этих элементов, если токен — число, то элемент заносится в стек. Как результат работы программы выводится число, которое осталось в стеке, или же ошибка, если оно не единственное.

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	-

Выводы

Получен опыт работы с ООП на языке программирования C++. Был реализован класс CustomStack и API для него.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
class CustomStack {
public:
    CustomStack() {
        mHead = nullptr;
        len = 0;
    }

    CustomStack(int val) {
        mHead = new ListNode;
        mHead->mData = val;
        mHead->mNext = nullptr;
        len = 1;
    }

    ~CustomStack() {
        while(!empty()) {
            pop();
        }
    }

    void push(int val) {
        len++;
        ListNode* node = new ListNode;
        node->mData = val;
        node->mNext = mHead;
        mHead = node;
    }

    void pop() {
        if (!empty()) {
            len--;
            ListNode* p = mHead;
            mHead = mHead->mNext;
            delete p;
        } else {
            cout<<"error"<<endl;
            exit(0);
        }
    }

    int top() {
        if (!empty())
            return mHead->mData;
        else {
            cout<<"error"<<endl;
            exit(0);
        }
    }

    size_t size() {
```

```
        return len;
    }

    bool empty() {
        if (mHead == nullptr)
            return true;
        else
            return false;
    }


protected:
    ListNode* mHead;
private:
    size_t len;

};

int main() {
    string input;
    getline(cin, input);
    istringstream ss(input);
    string token;
    vector<string> tokens;

    while(getline(ss, token, ' ')) {
        tokens.push_back(token);
    }

    CustomStack stack;

    for(const string t : tokens) {
        if (string{"+-*/"}.find(t) != string::npos) {
            int arg2 = stack.top();
            stack.pop();
            int arg1 = stack.top();
            stack.pop();
            switch (t[0])
            {
            case '+':
                stack.push(arg1 + arg2);
                break;
            case '-':
                stack.push(arg1 - arg2);
                break;
            case '*':
                stack.push(arg1 * arg2);
                break;
            case '/':
                stack.push(arg1 / arg2);
                break;
            default:
                break;
            }
        }
        else {
            int arg = stoi(t);
            stack.push(arg);
```

```
        }
    }

    if (stack.size() != 1) {
        cout<<"error"<<endl;
        exit(0);
    }
    else {
        cout<<stack.top()<<endl;
    }

    return 0;
}
```