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. C++ intro

Студент гр.1304 _____ Арчибасов Е.О.

Преподаватель _____ Чайка К.В.

Санкт-Петербург

2022

Цель работы.

Изучить принципы работы с динамическими структурами и ознакомится с основами написания программы на языке C++.

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

Пример

Исходная последовательность: 1 -10 - 2 *

Результат: 22

Основные теоретические положения

Стек - это структура данных, в которой хранятся элементы в виде последовательности, организованной по принципу LIFO (Last In — First Out). Такую структуру данных можно сравнить со стопкой тарелок или магазином автомата. Стек не предполагает прямого доступа к элементам и список основных операций ограничивается операциями помещения элемента в стек и извлечения элемента из стека. Их принято называть PUSH и POP соответственно. Также, обычно есть возможность посмотреть на верхний элемент стека не извлекая его (TOP) и несколько других функций, таких как проверка на пустоту стека и некоторые другие.

Стек можно легко реализовать на основе массива. Для этого достаточно хранить индекс "верхнего" элемента в стеке. Операция добавления сопровождается инкрементом этого индекса и записью в соответствующую ячейку нового значения. Операция извлечения сопровождается декрементом этого индекса. Дополнительно, может потребоваться реализовать возможность увеличения и уменьшения размера массива.

Класс - это шаблон, по которому определяется форма объекта. В нем указываются данные и код, который будет оперировать этими данными

Так же, класс - это абстрактный тип данных, который может включать в себя не только данные, но и программный код в виде функций. Они реализуют в себе оба принципа, описанных выше следующим образом:

В классе могут размещаться как данные (их называют полями), так и функции (их называют методы) для обработки этих данных. Любой метод или поле класса имеет свой спецификатор доступа: public, private или protected (его мы не будем рассматривать).

Выполнение работы.

Класс ***CustomStack***:

Поля класса:

1. int mSize — количество переменных хранящихся в данный момент в стеке (mSize-1 также является индексом последнего добавленного элемента).
2. int mAvailable — максимально доступный размер стека в данный момент.
3. int* mData — указатель на массив данных.

Методы класса:

1. CustomStack() - конструктор класса в нем выделяется начальная память для массива mData.
2. ~CustomStack() - деструктор класса в нем очищается дин. память выделенная для массива mData.
3. void extend(int n) - расширяет исходный массив на n ячеек .
4. bool empty() - проверяет отсутствие элементов в стеке.
5. int size() - возвращает количество элементов в стеке.
6. int top() - возвращает значение верхнего элемента.
7. int pop() - возвращает значение верхнего элемента и удаляет его из стека.
8. void push(int val) - добавляет новый элемент в стек.
9. Функция try_stoi(const string &s, int &i):

Функция проверяет можно ли строку ***s*** привести к целочисленному значению. Если да то присваивает это значение в ***i*** и возвращает ***true,*** если нельзя привести, возвращает ***false***.

Ход работы:

Считываем строки данных разделенные пробелом в token, пока не закончится ввод.

Проверяем для каждого token можно ли привести его к int.

Если можно привести token к int то добавляем это значение в стек my_stack.

Если нет, тогда значит token хранит в себе арифметическое действие(«+», «-», «\», «*»). Пытаемся достать и my_stack два верхних значения и сохранить их в left_num и right_num.

При неудачи обрабатываем исключение. При удаче сохраняем в стек my_stack результат применения соответствующего арф. Действия.

В итоге если в стеке остался один элемент то выводим его , если

не один выводим ошибку

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	1 -10 - 2 *	22	Программа работает корректно
2.	1 2 + 3 4 - 5 * +	-2	Программа работает корректно

Выводы.

Была изучены принципы создания динамических структур и работы с ними. Также были изучены основы написания программы на языке C++.

Разработана программа, полностью выполняющая поставленную задачу, а именно последовательно выполняет подаваемые ей на вход арифметические операции над числами с помощью стека на базе массива.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла main.cpp

```
#include <cstring>
#include <iostream>

using namespace std;

class CustomStack {
public:
    CustomStack() {
        mData = new int[mAvailable];
    }
    ~CustomStack() {
        delete[] mData;
    }
    void extend(int n) {
        mAvailable += n;
        int *new_Data = new int[mAvailable];
        memcpy(new_Data, mData, mSize * sizeof(int));
        delete[] mData;
        mData = new_Data;
    }
    void push(int a) {
        if (mSize + 1 >= mAvailable) {
            extend(10);
        }
        mData[mSize] = a;
        mSize++;
    }
    int pop() {
        if (empty()) {
            throw "error";
        }
        mSize--;
        return mData[mSize];
    }
    bool empty() {
        if (mSize == 0) {
            return  true;
        }
        else {
```

```
            return false;
        }
    }
    int top() {
        if (empty()) {
            throw "error";
        }
        return mData[mSize - 1];
    }
    int size() {
        return mSize;
    }

private:
    int mSize = 0;
    int mAvailable = 100;

protected:
    int* mData;
};

bool tryStoi(const string &s, int &i){
    try {
        i = stoi(s);
        return true;
    }
    catch (const std::invalid_argument&) {
        return false;
    }
}

int main() {
    CustomStack myStack;
    string tok ;
    int value;
    int leftNum , rightNum;
    while (cin >> tok) {
        if (tryStoi(tok, value)) {
            myStack.push(value);
        } else {
            try{
                rightNum = myStack.pop();
                leftNum = myStack.pop();
            }
```

```
            catch(const char* error_str) {
                cout << error_str;
                return 0;
            }
            if (tok == "+")
                myStack.push(leftNum + rightNum);
            if (tok == "-")
                myStack.push(leftNum - rightNum);
            if (tok == "*")
                myStack.push(leftNum * rightNum);
            if (tok == "/")
                myStack.push(leftNum / rightNum);
        }
    }
    if (myStack.size() != 1) {
        cout << "error";
        return 0;
    } else {
        cout << myStack.pop();
        return 0;
    }
}
```