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. Вариант 1

Студент гр. 3343 _____ Коршков А.А.

Преподаватель _____ Государкин Я.С.

Санкт-Петербург
2024

Цель работы

Познакомиться с объективно-ориентированным языком программирования C++, научиться работать с конструкторами и деструкторами, создавать классы и объекты этих классов, узнать, что у него общего с процедурным языком программирования C, как создавать динамические массивы и очищать их, как работать со строковым типом string.

Задание

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:

Указатель на массив должен быть protected.

Подключать какие-то заголовочные файлы не требуется, всё необходимое подключено.

Предполагается, что пространство имен std уже доступно.

Использование ключевого слова using также не требуется.

Выполнение работы

Создание класса CustomStack. Создаём поля mData, mIndex и mCapacity, они будут доступны только внутри объекта нашего класса. mData будет содержать указатель на целочисленный массив (стек), mIndex – индекс последнего элемента, mCapacity – размер стека. Также были созданы функции getNewCapacity и resize. GetNewCapacity возвращает новый увеличенный размер стека (сам стек не расширяется), resize функция позволяет расширить на определённое количество элементов. Выполняем проверку, что новый размер больше, чем изначальный размер стека. Создаём новый динамический массив, копируем информацию со старого массива в новый, предыдущий очищаем, и в него записываем указатель на новый массив.

Если пользователь не указал размер стека, то автоматически он будет создан на 10 элементов.

Публичными будут только методы для работы со стеком.

Чтобы добавить (append) элемент в массив, нужно выполнить проверку, есть ли место для добавления элемента в стек. Если места не хватает (mIndex + 1 == mCapacity), то вызываются функции getNewCapacity и resize, чтобы увеличить массив. После чего увеличиваем переменную mIndex и записываем новый элемент.

Чтобы удалить (pop) последний элемент из стека, необходимо отнять у поля mIndex единицу (т.е. указывается новый предельный элемент стека).

Для получения верхнего (последнего) элемента (top) необходимо вернуть элемент mData[mSize-1].

Функция size вернёт размер стека (mIndex + 1).

Функция empty возвращает True (1), если его размер mIndex = -1 (не существует, пустой).

Чтобы расширить массив (extend) на n элементов необходимо воспользоваться функцией resize, в аргумент передаём mCapacity+n.

Обработка входных данных. Создаём объект класса stack, а также элементы s и word типа string. С помощью stringstream из функции <sstream> в

word будут записываться слова, отделённые пробелом. Если word равен знаку равно, плюс, умножение, деление, то берутся 2 числа из стека, удаляются, а затем в стек добавляется произведённая над ними операция. Если это число, то оно преобразуется в формат int и кладётся в стек.

Если размер стека равен 1 элементу (результат всех произведённых операций), то необходимо вывести этот элемент.

Тестирование

Таблица 1 – Результат тестирования

Входные данные	Выходные данные	Комментарий
1 -10 - 2 *	22	ОК
1 2 + 3 4 - 5 * +	-2	ОК
-6 14 12 19 -15 12 1 18 -4 -6 14 10 14 -16 17 -10 * * + - * * + * + - * + + * *	-894477608	ОК
4 3 - 4 +	5	ОК

Выводы

Была написана программа, которая реализует вычисление выражения в постфиксной форме записи. В ней реализована работа с массивами, классами. Создана модель стека на основе класса C++.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
class CustomStack
{
public:
    void push(int val)
    {
        if (!mData)
        {
            mData = new int[1];
            mSize = 1;
            mData[0] = val;
        }
        else
        {
            int *newData = new int[mSize + 1];
            memcpy(newData, mData, mSize * sizeof(int));
            delete[] mData;
            mData = newData;
            mSize++;
            newData[mSize - 1] = val;
        }
    };
    void pop()
    {
        if (mSize != 0)
        {
            int *NewData = new int[mSize - 1];
            memcpy(NewData, mData, (mSize - 1) * sizeof(int));
            delete[] mData;
            mData = NewData;
            mSize--;
        }
        else
        {
            cout << "error";
            exit(0);
        }
    };
    int top()
    {
        if (mSize != 0)
        {
            return mData[mSize - 1];
        }
        else
        {
            cout << "error";
            exit(0);
        }
    };

    size_t size()
    {
```

```
        return mSize;
    };
    bool empty()
    {
        return mSize == 0;
    };
    void extend(int n)
    {
        if (mData)
        {
            int *newData = new int[mSize + n];
            memcpy(newData, mData, mSize * sizeof(int));
            delete[] mData;
            mData = newData;
            mSize += n;
        }
    };
    ~CustomStack()
    {
        delete[] mData;
    }

protected:
    int *mData = NULL;
    int mSize = 0;
};

int main()
{
    CustomStack stack;
    string s;
    getline(cin, s);
    stringstream ss(s);
    string word;
    while (ss >> word)
    {
        if (word == "-" || word == "+" || word == "*" || word == "/")
        {
            int b = stack.top();
            stack.pop();
            int a = stack.top();
            stack.pop();
            if (word == "-")
            {
                stack.push(a - b);
            }
            else if (word == "+")
            {
                stack.push(a + b);
            }
            else if (word == "*")
            {
                stack.push(a * b);
            }
            else if (word == "/")
            {
                stack.push(a / b);
```

```
            }
        }
        else
        {
            int i = stoi(word);
            stack.push(i);
        }
    }
    if (stack.size() == 1)
        cout << stack.top() << endl;
    else
        cout << "error";
    return 0;
}
```