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: ВВЕДЕНИЕ В ЯЗЫК C++

Студент гр. 3341 _____ Трофимов В.О.

Преподаватель _____ Глазунов С.А.

Санкт-Петербург
2024

Цель работы

Целью работы является изучение основных механизмов языка C++ путем разработки структур данных стека и очереди на основе динамической памяти.

Для достижения поставленной цели требуется решить следующие задачи:

– ознакомиться со структурами данных стека и очереди, особенностями их реализации;

– изучить и использовать базовые механизмы языка C++, необходимые для реализации стека и очереди;

– реализовать индивидуальный вариант стека в виде C++ класса, его операции в виде функций этого класса, ввод и вывод данных программы

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.

Пример:

Исходная последовательность: 1 -10 - 2 *

Результат: 22

Основные теоретические положения

Стек (Stack): Стек - это абстрактная структура данных, организованная по принципу LIFO (Last In, First Out), что означает, что последний добавленный элемент будет первым удаленным. Это подобно стопке тарелок: чтобы взять верхнюю тарелку, нужно сначала снять верхние тарелки.

Операции со стеком: Стек поддерживает две основные операции:

push(): добавляет элемент в верхушку стека.

pop(): удаляет элемент с верхушки стека.

Вершина стека: Это место, где происходят все операции добавления и удаления элементов. Новый элемент всегда добавляется на вершину стека, а при удалении элемента удаляется элемент с вершины.

Ограничение доступа: В стеке нет прямого доступа к произвольным элементам. Элементы могут быть добавлены или удалены только с вершины стека. Это делает стек простым в использовании, но ограничивает его функциональность.

Стековая память: В некоторых языках программирования (например, C++), локальные переменные и вызовы функций хранятся в стековой памяти. Это связано с тем, что операции push() и pop() используются для управления контекстом выполнения программы.

Применения стека:

Стеки широко используются в алгоритмах, таких как обход деревьев (например, обход в глубину). В парсинге арифметических выражений стек используется для хранения операторов и операндов при их вычислении. Механизм вызова функций в большинстве языков программирования реализуется с помощью стека вызовов. В обработке рекурсивных алгоритмов стек используется для хранения промежуточных результатов вызовов функций.

Реализация стека: Стек может быть реализован с использованием различных структур данных, например, массивов или связанных списков. Каждая реализация имеет свои преимущества и недостатки в зависимости от требований к производительности и используемой памяти.

Выполнение работы

В начале идет включение необходимых заголовочных файлов: <iostream>, <cstring> и <cstdlib>. Эти файлы нужны для использования стандартных потоков ввода/вывода (cin, cout), функций работы со строками (strlen, strtok) и функции преобразования строки в целое число (atoi).

Объявляется макрос BUF, который устанавливает размер буфера для строки ввода равным 100.

Объявляется класс CustomStack, реализующий стек целых чисел. В нем определены следующие методы:

CustomStack() - конструктор класса, инициализирующий указатель mHead на nullptr.

~CustomStack() - деструктор класса, освобождающий память, выделенную для элементов стека.

push(int val): - Метод добавляет новый элемент в стек. Он создает новый узел типа ListNode, присваивает ему значение val и помещает этот узел на вершину стека.

pop(): - Метод удаляет верхний элемент из стека. Он освобождает память, занимаемую верхним узлом, и перемещает указатель mHead на следующий элемент стека.

top(): - Метод возвращает значение верхнего элемента стека, но не удаляет его из стека. Если стек пуст, возвращается значение -1.

size(): - метод возвращает текущий размер стека, то есть количество элементов в нем. Он проходит по всем элементам стека, начиная с вершины, и подсчитывает их количество.

empty(): - Метод проверяет, пуст ли стек. Если mHead равен nullptr, то стек считается пустым, и метод возвращает true, в противном случае возвращает false.

В функции main() создается объект класса CustomStack под названием stack, который будет использоваться для хранения чисел и выполнения операций. Считывается строка с помощью cin.getline(). Строка представляет

собой математическое выражение в постфиксной записи. С помощью функции strtok() строка разбивается на токены (числа или операторы), которые затем обрабатываются в цикле. Внутри цикла проверяется тип каждого токена: если это число, оно добавляется в стек; если это оператор (+, -, *, /), то из стека извлекаются два числа, на которых выполняется соответствующая операция, и результат помещается обратно в стек. Если токен не является ни числом, ни оператором, выводится сообщение об ошибке. После обработки всех токенов проверяется, осталось ли в стеке ровно одно значение. Если нет, выводится сообщение об ошибке. Если все прошло успешно, на вершине стека остается результат вычисления выражения, который выводится на экран.

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	Тест с emoevm
2.	1 + 5 3 -	error	Тест с emoevm
3.	-12 -1 2 10 5 -14 17 17 * - - + - * +	304	Тест с emoevm
4.	0 0 /	error	Тест деление на ноль
5.	1000000000 1000000000 +	2000000000	Проверка на корректно считывание больших чисел.

Выводы

В ходе работы была изучена структура данных "стек" и его основные принципы работы. Была разработана программа на языке C++, которая реализует стек и выполняет вычисление арифметических выражений, записанных в постфиксной записи.

В программе использован класс CustomStack, который представляет собой реализацию стека целых чисел. Этот класс содержит методы для добавления элемента в стек (push), удаления элемента из стека (pop), получения верхнего элемента без его удаления (top), определения размера стека (size) и проверки на пустоту (empty). Также была реализована обработка ввода арифметического выражения в постфиксной записи, его разбиение на токены (числа и операторы) с использованием функции strtok, и выполнение соответствующих арифметических операций с помощью стека.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

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

#define BUF 100

using namespace std;

class CustomStack {
    public:

        CustomStack(){
            mHead = nullptr;
        }

        ~CustomStack(){
            while (mHead){
                pop();
            }
        }
        void push(int val){
            ListNode* current = new ListNode;
            if (current != nullptr){
                current->mData = val;
                current->mNext = mHead;
                mHead = current;
            }
        }

        void pop(){
            ListNode* current = mHead;
            if (this->empty()){
                mHead = mHead->mNext;
                delete current;
            }
        }

        int top(){
            if (mHead){
                return mHead->mData;
            }
            return -1;
        }

        size_t size(){
            ListNode* current = mHead;
            size_t size = 0;
            while (current){
```

```
                    size++;
                    current = current->mNext;
                }
                return size;
            }

            bool empty(){
                if (mHead)
                    return 1;
                return 0;
            }

        protected:
            ListNode* mHead;
    };

    int main() {
        CustomStack stack;
        char string[BUF];
        cin.getline(string, BUF);
        char* ptr = strtok(string, " ");
        while (ptr != nullptr) {
             if (isdigit(*ptr) || (*ptr == '-' && isdigit(*(ptr + 1))))
{
                stack.push(atoi(ptr));
            }
            else if (strlen(ptr) == 1 && (*ptr == '+' || *ptr == '-' ||
*ptr == '*' || *ptr == '/')) {
                if (stack.size() < 2) {
                    cout << "error" << endl;
                    return 0;
                }
                int num2 = stack.top();
                stack.pop();
                int num1 = stack.top();
                stack.pop();
                switch (*ptr) {
                    case '+':
                        stack.push(num1 + num2);
                        break;
                    case '-':
                        stack.push(num1 - num2);
                        break;
                    case '*':
                        stack.push(num1 * num2);
                        break;
                    case '/':
                        if (num2 == 0) {
                            cout << "error" << endl;
                            return 0;
                        }
                        stack.push(num1 / num2);
                        break;
                }
            }
            else {
                cout << "error" << endl;
                return 0;
```

```
            }
            ptr = strtok(nullptr, " ");
        }

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