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

ОТЧЕТ

по лабораторной работе №4

по дисциплине «Программирование»

Тема: Динамические структуры данных

Вариант 2

Студент гр. 0382 _____ Кондратов Ю.А.

Преподаватель _____ Берленко Т.А.

Санкт-Петербург

2021

Цель работы.

Изучение основных принципов работы с динамическими структурами данных на языке программирования Си. Реализация класса стека на базе списка.

Задание.

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

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.

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.

Пример добавления и удаления элементов из непустого стека (содержащего единицу):

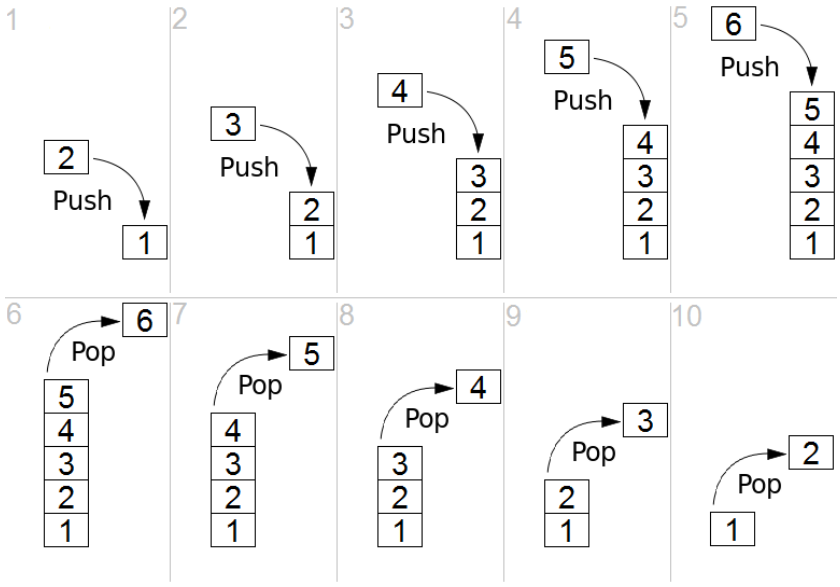

Рисунок 1 — Пример добавления и удаления элементов из стека

Класс в C++ - 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.

1) Реализация класса стека «Custom Stack».

Конструктор класса представляет из себя присваивание полю mHead объекта класса значение nullptr.

Метод void push(int val) — это функция добавления нового элемента со значением val в стек. В данной функции создается новый узел спика, полю mData этого узла присваивается значение val, после чего этот узел ставится на место головы списка.

Метод void pop() - это функция удаления верхнего элемента в стеке. В данной функции производится проверка стека на пустоту. Если проверка пройдена, то головой списка становится второй по счёту элемент, а первый элемент удаляется.

Метод int top() - это функция получения значения первого элемента в списке. В ней также производится проверка стека на пустоту, если проверка пройдена, то функция возвращает значения поля mData головы спика.

Метод size_t size() - это функция подсчёта элементов в стеке. В данной функции производится подсчёт всех элементов списка, то есть количества элементов до того, у которого поле mNext имеет значение nullptr включительно.

Метод bool empty() - это функция, позволяющая проверить список на пустоту. Возвращает единицу, если mHead имеет значение не nullptr и ноль в противном случае.

2) Реализация считывания и обработки

Считывание происходит по символьно при помощи cin с манипулятором noskipws до тех пор, пока очередной символ не будет равен переводу строки. Если очередной символ — это знак операции, то эта операция применяется к двум верхним элементам списка по следующему алгоритму:

```
int right = stack.top();
stack.pop();
int left = stack.top();
stack.pop();
stack.push(left * right);
```

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

Входные данные	Содержимое файла result.txt	Комментарии
1 -10 - 2 *	22	Программа работает правильно
1 2 + 3 4 - 5 * +	-2	Программа работает правильно
1 + 5 3 -	error	Программа работает правильно
-12 -1 2 10 5 -14 17 17 * - - + - * +	304	Программа работает правильно

Выводы.

В ходе работа были изучены основные принципы работы с файловыми динамическими структурами данных на языке Си. Реализован стек на базе линейного списка. Написана программа, выполняющая считывание элементов и операций из входного потока, сохраняющая числа в стеке и совершающая переданные операции над двумя верхними элементами списка.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ФАЙЛОВ ПРОЕКТА

1. Название файла: main.cpp

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

using namespace std;

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


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

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

    void pop() {
        if (this->empty()) {
            cout << "error" << endl;
            exit(0);
        }
        auto tmp = mHead;
        mHead = mHead->mNext;
        delete tmp;
    }

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

    size_t size() {
        size_t size = 0;
        ListNode *cur = mHead;
        while (cur != nullptr) {
            size += 1;
            cur = cur->mNext;
```

```
        }
        return size;
    }

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

protected:
    ListNode *mHead;
};

int main() {
    char c;
    string num;
    cin >> noskipws >> c;
    CustomStack stack;
    while (c != '\n') {
        if (c == '-') {
            cin >> noskipws >> c;
            if (isdigit(c)) {
                num += '-';
                while (isdigit(c)) {
                    num += c;
                    cin >> noskipws >> c;
                }
                stack.push(stoi(num));
                num = "";
            } else {
                int right = stack.top();
                stack.pop();
                int left = stack.top();
                stack.pop();
                stack.push(left - right);
            }
        } else if (isdigit(c)) {
            while (isdigit(c)) {
                num += c;
                cin >> noskipws >> c;
            }
            stack.push(stoi(num));
            num = "";
        } else {
            if (c == '+') {
                int right = stack.top();
                stack.pop();
                int left = stack.top();
                stack.pop();
                stack.push(left + right);
            }
            if (c == '*') {
                int right = stack.top();
                stack.pop();
                int left = stack.top();
                stack.pop();
```

```
                    stack.push(left * right);
                }
                if (c == '/') {
                    int right = stack.top();
                    if  (right  ==  0)  {cout  <<  "error"  <<  endl;
exit(0);}
                    stack.pop();
                    int left = stack.top();
                    stack.pop();
                    stack.push(left / right);
                }
                cin >> noskipws >> c;
            }
        }
        if (stack.size() > 1) cout << "error" << endl;
        else cout << stack.top();
    }
```