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. 0382 _____ Довченко М.К.

Преподаватель _____ Берленко Т.А.

Санкт-Петербург

2021

Цель работы.

Изучение динамических структур данных.

Задание.

Вариант 4. Моделирование стека.

Требуется написать программу, моделирующую работу стека на базе списка. Для этого необходимо:

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

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() - возвращает верхний элемент

size_t size() - возвращает количество элементов в стеке

bool empty() - проверяет отсутствие элементов в стеке

2) Обеспечить в программе считывание из потока stdin последовательности команд (каждая команда с новой строки), в зависимости от которых программа выполняет ту или иную операцию и выводит результат ее выполнения с новой строки.

Перечень команд, которые подаются на вход программе в stdin:

cmd_push n - добавляет целое число n в стек. Программа должна вывести "ok"

cmd_pop - удаляет из стека последний элемент и выводит его значение на экран

cmd_top - программа должна вывести верхний элемент стека на экран не удаляя его из стека

cmd_size - программа должна вывести количество элементов в стеке

cmd_exit - программа должна вывести "bye" и завершить работу

Основные теоретические положения.

Язык C++ реализует объектно-ориентированную парадигму программирования, которая включает в себя реализацию механизма инкапсуляции данных . Инкапсуляция в C++ подразумевает, что: в одной языковой конструкции размещаются как данные, так и функции для обработки этих данных.

Доступ к данным извне этой конструкции ограничен, иными словами, напрямую редактировать данные как в структурах C нельзя. Пользователю предоставляется интерфейс из методов (API) с помощью которого он может влиять на состояние данных.

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

Выполнение работы.

В качестве стека используется *class CustomStack*, в нём реализованы следующие функции:

- *CustomStack()* — конструктор класса, который инициализирует приватные переменные и выделяет память на массив *mData*.
- *~CustomStack()* — деструктор класса, освобождающий память массива mData.
- *void push(int num)* — добавляет элемент в стэк.
- *void pop()* — удаляет последний элемент из стэка, если стэк пуст, то выводит «error» и завершает программу с кодом 0.
- *int top()* — возвращает последний элемент, добавленный в стэк. Если стэк пуст, то выводит сообщение «error» и завершает программу.
- *size_t size()* — выводит количество элементов стэка.

- *Bool empty()* — проверяет пуст ли стэк, если стэк пуст возвращает значение True, если нет, то значение False.

Разработанный программный код см. в приложении А.

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	cmd_push 1 cmd_top cmd_push 2 cmd_top cmd_pop cmd_size cmd_pop cmd_size cmd_exit	ok 1 ok 2 2 1 1 0 bye	Программа работает корректно

Выводы.

Были изучены принципы динамические структур данных – стэк и очередь.

Разработана программа, в которой реализован стэк на базе списка.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lb4.cpp

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


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


class CustomStack {


public:
    CustomStack() {
        mHead = static_cast<ListNode *>(malloc(sizeof(ListNode)));
        lastNode = NULL;
        mHead = NULL;
    }


    ~CustomStack() {
        ListNode *current = mHead;
        while (current != NULL) {
            mHead = mHead->mNext;
            free(current);
            current = mHead->mNext;



        }
    }


    void push(int num) {
        if (mHead == NULL) {
            mHead = static_cast<ListNode
*>(malloc(sizeof(ListNode)));
            mHead->mData = num;
            mHead->mNext = NULL;
            lastNode = mHead;
            return;
        }
        lastNode->mNext = static_cast<ListNode
*>(malloc(sizeof(ListNode)));
        lastNode = lastNode->mNext;
        lastNode->mNext = NULL;
        lastNode->mData = num;
    }
```

```
    void pop() {
        if (mHead == NULL) {
            std::cout << "error" << std::endl;
            exit(0);
        }
        if (mHead->mNext == NULL) {
            std::cout << mHead->mData << std::endl;
            free(mHead);
            mHead = NULL;


            return;
        }
        ListNode *current = mHead;
        while (current->mNext->mNext != NULL) {
            current = current->mNext;
        }
        std::cout << current->mNext->mData << std::endl;
        free(current->mNext);
        lastNode = current;
        current->mNext = NULL;
    }


    int top() {
        if (mHead == NULL) {
            std::cout << "error" << std::endl;
            exit(0);
        }


        return lastNode->mData;
    }


    size_t size() {
        ListNode *current = mHead;
        size_t size = 0;
        while (current != NULL) {
            size++;
            current = current->mNext;
        }
        return size;
    }



    bool empty() {
        return mHead == NULL;
    }



private:
    ListNode *lastNode;

protected:
```

```
        ListNode *mHead;
    };


    int main() {
        CustomStack Stack;
        const int input_length = 10;
        char commands[5][input_length] = {"cmd_push", "cmd_pop",
"cmd_top", "cmd_size", "cmd_exit"};
        int option;
        char what_to_push[20];
        char input[input_length];
        while (true) {
            std::cin >> input;
            if (!strcmp(input, commands[0])) {
                std::cin >> what_to_push;
                option = 1;
            }
            for (int j = 1; j < 5; j++) {
                if (!strcmp(input, commands[j])) {
                    option = j + 1;
                }
            }
            switch (option) {
                case 1:
                    Stack.push(atoi(what_to_push));
                    std::cout << "ok\n";
                    break;
                case 2:
                    Stack.pop();
                    break;
                case 3:
                    std::cout << Stack.top() << std::endl;
                    break;
                case 4:
                    std::cout << Stack.size() << std::endl;
                    break;
                case 5:
                    std::cout << "bye\n";
                    exit(0);
                default:
                    std::cout << "non-existent command\n";
            }
            option = 0;
        }
        return 0;
    }
```