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. 3342 _____ Корниенко А.Е.

Преподаватель _____ Глазунов С.А.

Санкт-Петербург
2024

Цель работы

Целью данной лабораторной работы является изучение работы с динамическими структурами данных и их создание. Также одна из целей – изучение основ работы с языком C++. Требуется написать программу, моделирующую работу стека на базе списка**.**

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.***

Структура класса узла списка:

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

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 голову

   ListNode* mHead;
};
```

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

Если в процессе вычисления возникает ошибка (например вызов метода **pop** или **top** при пустом стеке), программа должна вывести "**error**" и завершиться.

Примечания:

1. Указатель на голову 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.
5. Структуру ListNode реализовывать самому не надо, она уже реализована.

Выполнение работы

Реализован класс CustomStack, который имеет следующие методы: push, pop, size, empty, top. Фунция push добавляет новый элемент, pop – удаляет последний элемент, top – возвращает верхний элемент, size – возвращает количество элементов в стеке, empty – проверяет отсутствие элементов в стеке. Реализован main() в котором считываются и выполняются пользовательские команды. Есть проверка на пустоту массива, при вызове pop и top.

Тестирование

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные
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

Выводы

Была разработана программа на языке C++, которая создаёт динамическую структуру данных – стек на базе списка. Реализованы методы для работы с созданной структурой и считывание пользовательских команд и их выполнение.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
class CustomStack {

public:
    CustomStack()
    {
        mHead = nullptr;
    }
    void
    push(int val)
    {
        ListNode* root = new ListNode;
        if(!root){
            std::cout << "Memory allocation error" << std::endl;
            exit(1);
        }
        root->mData = val;
        root->mNext = mHead;
        mHead = root;
    }
    void
    pop()
    {
        if(empty())
        {
            return;
        }
        ListNode* temp = mHead;
        mHead = mHead->mNext;
        delete temp;
    }
    int
    top()
    {
        return mHead->mData;
    }
    size_t
    size()
    {
        size_t count = 0;
        ListNode* current = mHead;
        while (current != nullptr)
        {
            count++;
            current = current->mNext;
        }
        return count;
    }
    bool
    empty()
    {
```

```
        if(mHead)
            return false;

        return true;

    }
    ~CustomStack()
    {
        while(!empty())
        {
            pop();
        }
    }
protected:

    ListNode* mHead;
};



int main() {


    CustomStack stack;

    std::string command;
    while (std::cin >> command) {
        if (command == "cmd_push") {
            int val;
            std::cin >> val;
            stack.push(val);
            std::cout << "ok" << std::endl;
        } else if (command == "cmd_pop") {
            if(!stack.empty())
            {
                std::cout << stack.top() << std::endl;
                stack.pop();
            }
            else
            {
                std::cout << "error";
                stack.pop();
                break;
            }
        } else if (command == "cmd_top") {
            if(!stack.empty()){
                std::cout << stack.top() << std::endl;
            }
            else
            {
                std::cout << "error";
                break;
            }
        } else if (command == "cmd_size") {
            std::cout << stack.size() << std::endl;
        } else if (command == "cmd_exit") {
            std::cout << "bye";
            break;
```

```
            }
        }

        return 0;
    }
```