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 _____ Гончаров С.А.

Преподаватель _____ Глазунов С.А.

Санкт-Петербург
2024

Цель работы

Требуется написать программу, моделирующую работу стека на базе списка.

Задание

Вариант 4.

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 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

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 (для операции в стеке не хватает аргументов),
- по завершении работы программы в стеке более одного элемента,

программа должна вывести "**error**" и завершиться.

Примечания:

1. Указатель на массив 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

Был создан класс CustomStack, предоставляющий функциональность стека. Он включает методы push для добавления элементов, pop для извлечения элементов, size для определения размера стека, empty для проверки на пустоту, top для просмотра верхнего элемента. Внутренняя структура класса использует приватное поле для хранения размера стека и защищенное поле mData для хранения самих данных. В функции main происходит чтение и обработка команд пользователя, при этом проверяется, не пуст ли стек, перед выполнением операций pop и top, чтобы избежать ошибок.

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 написана программа на языке C++, которая формирует динамическую структуру данных – стек, используя список в качестве базы. В ней реализованы методы для работы с этой структурой, а также для считывания пользовательских команд и их выполнения.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
class CustomStack {
public:
    CustomStack() : mHead(nullptr) {}   // Конструктор

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

    void pop() {
        if (empty()) {
            return;
        }

        ListNode* temp = mHead;
        mHead = mHead->mNext;
        delete temp;
    }

    int top() const {
        if (empty()) {
            std::cout << "error" << std::endl;
            exit(1);
        }
        return mHead->mData;
    }

    size_t size() const {
        size_t count = 0;
        ListNode* current = mHead;

        while (current) {
            count++;
            current = current->mNext;
        }
        return count;
    }

    bool empty() const {
        return mHead == nullptr;
    }

    ~CustomStack() {                    // Деструктор
        while (!empty()) {
            pop();
```

```
        }
    }

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

    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" << std::endl;
                stack.pop();
                break;
            }
        } else if (command == "cmd_top") {
            if (!stack.empty()) {
                std::cout << stack.top() << std::endl;
            } else {
                std::cout << "error" << std::endl;
                break;
            }
        } else if (command == "cmd_size") {
            std::cout << stack.size() << std::endl;
        } else if (command == "cmd_exit") {
            std::cout << "bye" << std::endl;
            break;
        } else {
            std::cout << "error" << std::endl;
            break;
        }
    }

    return 0;
}
```