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 _____ Романов Е.А.

Преподаватель _____ Глазунов С.А.

Санкт-Петербург
2024

Цель работы

Изучение динамических структур данных, а также основ языка программирования C++. Написание программы, реализовывающей структуру данных стек на базе связного списка.

Задание

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 на 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

Выполнение работы

Программа состоит из описания класса CustomStack и функции main.

Методы класса:

- Конструкторы. Класс содержит описание двух конструкторов: первый используется, когда экземпляр класса создаётся без головного элемента, то есть стек пуст, второй – когда в качестве аргумента подаётся указатель на структуру ListNode, то есть стек создаётся с 1 элементом.
- push. Принимает в качестве аргумента переменную типа integer, создаёт указатель на структуру ListNode, заполняя её поле mData значением, полученным методом в качестве аргумента, а поле mNext указателем на головной элемент стека. После этого метод устанавливает созданную структуру в качестве головного элемента стека.
- pop. Получает указатель на верхний элемент стека, меняет его значение на значение следующего элемента, а освободившийся элемент удаляет при помощи оператора delete.
- top. Обращается к верхнему элементу стека и возвращает его значение.
- size. Получает указатель на верхний элемент стека и при помощи цикла while перемещается по стеку до нижнего элемента по полям mNext элементов, подсчитывая их количество. Полученное значение возвращается.
- empty. Метод проверяет, является ли указатель на верхний элемент структуры nullptr и возвращает true, если это так, иначе false.
- Деструктор. Вызывает метод pop до тех пор, пока стек не будет очищен.

В функции main создаётся экземпляр класса CustomStack, при помощи функции cin производится считывание команд из стандартного потока ввода и их последующее выполнение, согласно условиям задания.

Переменные используемые в программе:

-mHead указатель типа ListNode – приватное поле класса CustomStack, используемое для хранения верхнего элемента стека

-ListNode* newElem – указатель на область памяти, выделенную для новой структуры ListNode, которая будет помещена в стек. Локальная переменная метода push

-ListNode* headNode указатель на верхний элемент стека. Локальная переменная метода pop

-int count хранит количество элементов стека. Локальная переменная метода size

-ListNode* currentElement указатель на текущий элемент стека, при движении по списку элементов. Локальная переменная метода size

-CustomStack myStack экземпляр класса CustomStack

-int nodeData хранит значение, считанное из стандартного потока ввода, которое будет помещено в элемент стека

-string currWord хранит строковое представление считанной из стандартного потока ввода команды по работе со стеком

Функции стандартной библиотеки и операторы, используемые в программе:

-new выделяет динамическую память в куче

-cerr выводит сообщение в стандартный поток вывода ошибок

-exit немедленно завершает выполняет программу

-delete освобождает выделенную динамически память

-using namespace используется для определения пространства имён

-cout используется для вывода данных в стандартный поток вывода

-cin используется для получения данных из стандартного потока ввода

Разработанный программный код см. в приложении А.

Тестирование

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарий
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	Ответ верный

Выводы

В результате работы была реализована программа на языке программирования C++, моделирующая работу динамической структуры данных стек, на основе связного списка.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
class CustomStack {
public:

    CustomStack(){
        mHead = nullptr;
    }
    CustomStack(ListNode* headElem){
        mHead = headElem;
    }

    void push(int val){
        ListNode* newElem = new ListNode;

        if (newElem == nullptr) {
            cerr << "Ошибка выделения памяти!\n";
            exit(1);
        }

        newElem->mData = val;
        newElem->mNext = mHead;

        mHead = newElem;

    }

    void pop(){

        ListNode* headNode = mHead;

        if (headNode == nullptr)
            return;

        mHead = mHead->mNext;

        delete headNode;
    }

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

    size_t size(){
        int count = 0;

        ListNode* currentElement = mHead;

        if (currentElement == nullptr) return count;

        while (currentElement != nullptr){
```

```
            currentElement = currentElement->mNext;
            count++;
        }

        return count;
    }

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

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

protected:
    ListNode* mHead;
};


int main(){
    using namespace std;

    CustomStack myStack;

    string currWord;

        while (cin >> currWord){

        if (currWord == "cmd_push") {
            int nodeData;
            cin >> nodeData;

            myStack.push(nodeData);

            cout << "ok\n";

        } else if (currWord == "cmd_pop"){
            if (!myStack.empty()){
                cout << myStack.top() << '\n';
                myStack.pop();
            } else {
                cout << "error" << '\n';
                break;
            }

        } else if (currWord == "cmd_size"){

            cout << myStack.size() << "\n";

        } else if(currWord == "cmd_top"){
            if (!myStack.empty()){
                cout << myStack.top() <<'\n';
            } else {
```

```
                cout << "error";
                break;
            }
        } else if(currWord == "cmd_exit"){
            cout << "bye\n";
            break;
        }

    }

    return 0;
}
```