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.

Студентка гр. 3341 Чинаева М.Р.

Преподаватель Глазунов С.А.

Санкт-Петербург
2024

Цель работы

Целью работы является изучение основных механизмов языка C++ путем разработки структур данных стека и очереди на основе динамической памяти.

Для достижения поставленной цели требуется решить следующие задачи:

– ознакомиться со структурами данных стека и очереди, особенностями их реализации;

– изучить и использовать базовые механизмы языка C++, необходимые для реализации стека и очереди;

– реализовать индивидуальный вариант стека в виде C++ класса, его операции в виде функций этого класса, ввод и вывод данных программы.

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 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

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

Примечания:

Указатель на голову должен быть protected.

Подключать какие-то заголовочные файлы не требуется, всё необходимое подключено.

Предполагается, что пространство имен std уже доступно.

Использование ключевого слова using также не требуется.

Структуру ListNode реализовывать самому не надо, она уже реализована.

Выполнение работы

Реализованные функции:

1. int main() - создает экземпляр класса CustomStack, после чего передает его в функцию для считывания и выполнения команд с консоли.

2. void input(CustomStack& stack) – с помощью цикла while (check_input) поочередно считываются и выполняются команды, соответствующие введенным. Для команды "cmd_pop" сначала выводится значение с помощью метода top(), потом последнее значение удаляется.

Структура класса CustomStack

Private поля:

1. lenStack – хранит длину стека

2. void check() – в случае если длина стека равна 0 выводит сообщение “error” и завершает работу программы

Protected поля

ListNode* mHead — указатель на голову стека, т.е. На первый добавленный элемент

Public поля:

1. Конструктор и деструктор.

В конструкторе указателю mHead передается значение nullptr. В деструкторе очищается память, выделенная под mHead.

2. void push(int val)

Создается указатель на новый элемент типа ListNode, полю mData которого присваивается переданное значение. Если длина стека равна 0, переданный элемент становится головой. Если нет, то с помощью цикла while (current->mNext != nullptr) проходится до конца списка и после последнего элемента добавляется новый. Длина стека увеличивается.

3. void pop()

Сначала вызывается проверка на то, что список не пустой. Потом если длина списка равна 1, список становится пустым. Если же список больше, то с помощью цикла while (current->mNext->mNext != nullptr) находится

предпоследний элемент и удаляется элемент после него. Длина стека уменьшается.

4. int top()

Сначала вызывается проверка на то, что список не пустой. Потом с помощью цикла while (current->mNext != nullptr) находится последний элемент стека. Функция возвращает его поле mData

5. size_t size()

Возвращает длину стека.

6. bool empty()

Если длина стека равна 0, возвращает true, если нет, то 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	Тест с e.moevm
2.	cmd_push 1 cmd_push 2 cmd_size cmd_pop cmd_size cmd_push 3 cmd_pop cmd_size cmd_pop cmd_exit	ok ok 2 2 1 ok 3 1 1 bye	Проверка работы основных методов класса
3.	cmd_size cmd_pop	0 error	Обработка некорректного вызова метода pop()
4.	cmd_size cmd_top	0 error	Обработка некорректного вызова метода top()
5.	cmd_size cmd_cmd cmd_push 1 cmd_pop cmd_exit	0 ok 1 bye	Неизвестные команды игнорируются

Выводы

Цель работы была успешно достигнута. Был создан класс CustomStack, реализующий моделирование работы стека на базе списка. Программа обрабатывает команды из потока ввода stdin и выполняет соответствующие действия согласно протоколу, включая добавление элементов в стек, удаление последнего элемента, вывод верхнего элемента, вывод количества элементов и завершение программы по команде "cmd_exit". При возникновении ошибок, таких как вызов метода pop или top при пустом стеке, программа корректно выводит "error" и завершается.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.c

```
class CustomStack {
public:
    CustomStack() {
        lenStack = 0;
        this->mHead = nullptr;
    }

    ~CustomStack() {
        delete this->mHead;
    }
    void push(int val) {
        ListNode* new_el= new ListNode;
        new_el->mData = val;
        new_el->mNext = nullptr;
        if (lenStack == 0) {
            mHead = new_el;
        }
        else {
            ListNode* current = mHead;
            while (current->mNext != nullptr) {
                current = current->mNext;
            }
            current->mNext = new_el;

        }
        lenStack++;
    }

    void pop() {
        check();
        if (lenStack == 1) {
            this->mHead = nullptr;
        }
        else {
            ListNode* current = mHead;
            while (current->mNext->mNext != nullptr) {
                current = current->mNext;
            }
            current->mNext = nullptr;
        }
        lenStack--;
    }

    int top() {
        check();
        ListNode* current = mHead;
        while (current->mNext != nullptr) {
            current = current->mNext;
        }
        return current->mData;
    }
```

```
    size_t size() {
        return lenStack;
    }

    bool empty() {
        if (lenStack == 0) {
            return true;
        }
        return false;
    }

private:
    int lenStack;

    void check() {
        if (lenStack == 0) {
            std::cout << "error\n";
            exit(0);
        }
    }

protected:
    ListNode* mHead;
};

void input(CustomStack& stack) {
    std::string input_cmd;
    int check_input = 1;
    while (check_input) {
        std::cin >> input_cmd;
        if (input_cmd == "cmd_push") {
            int val;
            std::cin >> val;
            stack.push(val);
            std::cout << "ok\n";
        }
        if (input_cmd == "cmd_pop") {
            std::cout << stack.top() << "\n";
            stack.pop();
        }
        if (input_cmd == "cmd_top") {
            std::cout << stack.top() << "\n";
        }
        if (input_cmd == "cmd_size") {
            std::cout << stack.size() << "\n";
        }
        if (input_cmd == "cmd_exit") {
            std::cout << "bye\n";
            check_input = 0;
        }
    }
}

int main() {
    CustomStack stack;
    input(stack);
    return 0;
}
```