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. 3341 _____ Романов А.К.

Преподаватель _____ Глазунов С.А.

Санкт-Петербург
2024

Цель работы

Написать программу, реализующую моделирование работы стека на базе списка. Для этого необходимо создать класс CustomStack с методами push, pop, top, size, empty, которые будут работать с элементами типа int. Программа должна обрабатывать команды из потока ввода stdin и выполнять соответствующие действия согласно протоколу:

- cmd_push n: добавление целого числа n в стек.
- cmd_pop: удаление последнего элемента из стека и вывод его значения.
- cmd_top: вывод верхнего элемента стека.
- cmd_size: вывод количества элементов в стеке.
- cmd_exit: завершение программы.

При возникновении ошибок (например, вызов метода pop или top при пустом стеке), программа должна вывести "error" и завершиться.

Примечания:
- Указатель на голову стека должен быть защищенным (protected).
- Необходимо использовать предоставленную структуру ListNode.
- Не требуется подключение дополнительных заголовочных файлов.
- Не нужно использовать using для пространства имен std.

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

Используемые переменные:

- STRING_SIZE 50
- *CustomStack stack;* — экземпляр класса CustomStack. Собственно сам стек.
- *char input[STRING_SIZE]* — получает у пользователя командуна вход.

Реализованные функции:

- *void match(const char *dirPath, FILE *result, char str)* — принимает на вход адрес обрабатываемой директории, файл, в который будет вестись запись ответа, а также символ *str*. (В последствии функция сравнивает названия файлов с этим символом). Функция обходит все файлы в указанной директории. В случае если встречается директория, функция вызывает саму себя, передавая в качестве параметра *dirPath* адрес найденной директории. В случае если очередной файл не является директорией, функция проверяет, что его имя состоит из одного символа, и если символ совпадает с *str*, осуществляется запись полного пути найденного файла в .txt документ *result*.
- *int main()* - создает экземпляр класса CustomStack, после чего в цикле while осуществляется ввод команд с консоли. Программа прекращает работу, когда введена команда cmd_exit.

Структура класса CustomStack

- Private поля:
  - size_t number_of_elements - содержит информации о количестве элементов в стеке
  - ListNode* previous_element — указатель на последний добавленный в стек элемент
- Protected поля
  - *ListNode* mHead* — указатель на голову стека, т. е. На первый добавленный элемент
- Public поля

- Конструктор и деструктор. В конструкторе указателю mHead передается знчение nullptr. В деструкторе очищается память выделенная под mHead.
- Функции стека в соответствии с заданием.

  1. Void push добавляет новый элемент в стек или создает головной элемент, в случае если стек пустой. Увеличивает счётчик элементов.

  2. Void pop выводит на экран данные последнего добавленного элемента стека (previous_element) и удаляет его. После чего previous_element получает новое значение, соответствующее новму последнему элементу. В случае если в стеке был всего один элемент, то после его удаления mHead будет присвоено значение nullptr. Если элементов не было вообще, программа выведет сообщение об ошибке и завершит работу.

  3. int top — возвращает информацию о последнем добавленном в стек элементе. Если элементов нет, программа выводит сообщение об ошибке и завершает работу.

  4. size_t size — возвращает количество элементов в стеке (в т.ч. и 0)

  5. bool empty — возвращает 1 если список пустой, 0 — если наоборот.

  6. void quit — завершает работу программы с сообщением «bye!»

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 12 cmd_push 14 cmd_top cmd_pop cmd_top cmd_pop cmd_size cmd_exit	ok ok 14 14 12 12 0 bye	Обычный тест
3.	cmd_size cmd_pop	0 error	Обработка некорректного вызова команд
4.	cmd_size cmd_empty cmd_push 68 cmd_empty cmd_pop cmd_empty cmd_top error	0 1 ok 0 68 1 error	Работа команды empty, вызов ошибки
5.	cmd_size cmd_cmd	0 ok	Неизвестные команды игнорируются

	cmd_push 12 cmd_pop cmd_exit	12 bye	

Выводы

Цель программы 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. Все требования к реализации были выполнены, а указатель на голову стека защищен.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.c

```
#include <iostream>
#include <cstring>
#include <string>
#define STRING_SIZE 50


class CustomStack {

private:

    size_t number_of_elements = 0;
    ListNode* previous_element;


public:

    CustomStack(){
        this->mHead = nullptr;
    }

    ~CustomStack(){
        delete this->mHead;
    }

    void push(int val){
        if (number_of_elements == 0){
            ListNode *object = new ListNode;
            object->mData = val;
            object->mNext = nullptr;
            mHead = object;
            previous_element = object;

        }
        else{
            ListNode *object = new ListNode;
            object->mData = val;
            object->mNext = nullptr;
            previous_element->mNext = object;
            previous_element = object;
        }

        number_of_elements++;
    }

    void pop(){

        if(number_of_elements > 0){

            std::cout << previous_element->mData << std::endl;
            delete previous_element;
```

```
            if(number_of_elements == 1){
                mHead = nullptr;
            }
            else{
                ListNode* ptr = mHead;
                while(1){
                    if(ptr->mNext == previous_element){
                        previous_element = ptr;
                        break;
                    }
                    else{
                        ptr = ptr->mNext;
                    }
                }
            }
            number_of_elements--;
        }
        else{
            std::cout << "error" << std::endl;
            exit(0);
        }


    }

    int top(){
        if(number_of_elements > 0){
            return previous_element->mData;
        }
        else{
            std::cout << "error" << std::endl;
            exit(0);
        }
    }

    size_t size(){
        return number_of_elements;
    }

    bool empty(){
        return number_of_elements == 0;
    }

    void quit(){
        std::cout << "bye";
        exit(0);
    }

protected:
    ListNode* mHead;
};

int main() {
    CustomStack stack;

    while(1){
        char input[STRING_SIZE];
```

```
        fgets(input, sizeof(input), stdin);
        input[strcspn(input, "\n")] = '\0';

        if (strstr(input, "cmd_push")){
            int element = std::stoi(strstr(input, " ")+1);
            stack.push(element);
            std::cout << "ok" << std::endl;
        }

        if (strstr(input, "cmd_pop")){
            stack.pop();
        }

        if (strstr(input, "cmd_top")){
            std::cout << stack.top() << std::endl;
        }

        if (strstr(input, "cmd_size")){
            std::cout << stack.size() << std::endl;
        }

        if(strstr(input, "cmd_empty")){
            std::cout << stack.empty() << std::endl;
        }

        if(strstr(input, "cmd_exit")){
            stack.quit();
        }
    }
    return 0;
}
```