**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 |  | Мальцев К.Л. |
| 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:

сmd\_push n - добавляет целое число n в стек. Программа должна вывести "ok"

сmd\_pop - удаляет из стека последний элемент и выводит его значение на экран

сmd\_top - программа должна вывести верхний элемент стека на экран не удаляя его из стека

сmd\_size - программа должна вывести количество элементов в стеке

сmd\_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.

## Основные теоретические положения

Стек (stack) - это абстрактная структура данных, которая представляет собой коллекцию элементов, организованных по принципу Last In First Out (LIFO). Это означает, что элементы добавляются и удаляются из стека только с одного конца, называемого вершиной стека.

Вот основные теоретические положения о стеке:

1. Операции со стеком:

- push(): добавляет элемент на вершину стека.

- pop(): удаляет и возвращает элемент с вершины стека.

- top(): возвращает элемент, находящийся на вершине стека, без его удаления.

- empty(): проверяет, пуст ли стек.

- size(): возвращает количество элементов в стеке.

2. Вершина стека:

- Вершина стека - это элемент, добавленный последним. Она представляет последний добавленный и первый удаляемый элемент стека.

3. Реализация стека:

- Стек можно реализовать с помощью статического массива, динамического массива или связного списка.

- В C++ стандартная библиотека содержит класс std::stack, который представляет стек.

4. Применение стека:

- Стек широко используется в программировании. Некоторые примеры использования стека:

- Рекурсивные вызовы функций.

- Обработка операций в обратной польской записи (постфиксной нотации).

- Управление операциями возврата (backtracking).

- Обработка операций undo/redo.

- Решение задач на графах (DFS - Depth First Search).

5. Важность стека:

- Использование стека позволяет эффективно управлять данными, сохраняя порядок их добавления и удаления.

- Стек обеспечивает простой доступ к последнему добавленному элементу и удобство его обработки.

Стек - это важная структура данных, которая играет ключевую роль во многих алгоритмах и программах. Понимание его основных принципов и операций поможет в разработке эффективных и легко поддерживаемых программ.

## Выполнение работы

Ход работы по коду:

1. Создается класс CustomStack, содержащий методы для работы со стеком и управления элементами:

- push(int data): добавляет элемент на вершину стека.

- pop(): удаляет элемент с вершины стека.

- top(): возвращает значение элемента на вершине стека.

- size(): возвращает количество элементов в стеке.

- empty(): проверяет, пуст ли стек.

2. Реализованы методы:

- throwIndexError(): выводит сообщение об ошибке и завершает программу.

- showData(ListNode\* node): выводит данные узла (не используется в коде).

3. Пользовательские функции:

- quit(): выводит сообщение о завершении программы и завершает программу.

- throwOkMessage(): выводит сообщение об успешном выполнении операции.

4. Функция commandAllocator(CustomStack& stack): обрабатывает ввод пользователя и вызывает соответствующие методы стека в зависимости от команды, вводимой пользователем. Доступны команды: cmd\_push, cmd\_pop, cmd\_top, cmd\_size, cmd\_exit.

5. В main() функции создается экземпляр объекта CustomStack, после чего вызывается commandAllocator(), где происходит обработка команд пользователя согласно логике, описанной в коде.

Таким образом, программа позволяет пользователю добавлять элементы в стек, удалять элементы, просматривать верхний элемент, узнавать размер стека и завершать выполнение программы.

## Тестирование

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Входные данные | Выходные данные | Комментарии |
|  | 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 |

## 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: solution.c

class CustomStack

{

public:

CustomStack() {

this->mHead = nullptr;

}

~CustomStack() {

delete this->mHead;

}

void push(int data)

{

ListNode\* newNode = new ListNode();

newNode->mData = data;

newNode->mNext = this->mHead;

this->mHead = newNode;

}

void pop()

{

if (this->empty()) {

this->throwIndexError();

}

this->mHead = this->mHead->mNext;

}

int top()

{

if (this->empty()) {

throwIndexError();

}

return this->mHead->mData;

}

size\_t size()

{

ListNode\* current = mHead;

size\_t size = 0;

while (current != nullptr) {

current = current->mNext;

size++;

}

return size;

}

bool empty()

{

return (this->mHead == nullptr);

}

private:

void throwIndexError()

{

std::cout << "error" << std::endl;

exit(0);

}

void showData(ListNode\* node)

{

std::cout << node->mData << std::endl;

}

protected:

ListNode\* mHead;

};

void quit()

{

std::cout << "bye" << std::endl;

exit(0);

}

void throwOkMessage()

{

std::cout << "ok" << std::endl;

}

void commandAllocator(CustomStack& stack)

{

std::string command;

while (true) {

std::cin >> command;

if (command == "cmd\_push") {

int data;

std::cin >> data;

stack.push(data);

throwOkMessage();

}

if (command == "cmd\_pop") {

std::cout << stack.top() << std::endl;

stack.pop();

}

if (command == "cmd\_top") {

std::cout << stack.top() << std::endl;

}

if (command == "cmd\_size") {

std::cout << stack.size() << std::endl;

}

if (command == "cmd\_exit") {

quit();

}

}

}

int main()

{

CustomStack stack;

commandAllocator(stack);

return 0;

}