**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: Введение в язык С++

|  |  |  |
| --- | --- | --- |
| Студентка гр. 3343 |  | Лобова Е. И. |
| Преподаватель |  | Государкин Я.С. |

Санкт-Петербург

2024

## Цель работы

Целью работы является изучение основных механизмов языка С++ 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 С++, необходимые

для реализации стека и очереди;

– реализовать индивидуальный вариант стека в виде С++ класса, его

операции в виде функций этого класса, ввод и вывод данных программы.

## Задание

Моделирование стека.

Требуется написать программу, моделирующую работу стека на базе массива. Для этого необходимо:

1) Реализовать класс CustomStack, который будет содержать перечисленные ниже методы. Стек должен иметь возможность хранить и работать с типом данных int.

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() - возвращает верхний элемент
* size\_t size() - возвращает количество элементов в стеке
* bool empty() - проверяет отсутствие элементов в стеке
* extend(int n) - расширяет исходный массив на n ячеек

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.

## Выполнение работы

В ходе выполнения лабораторной работы был написан класс CustomStack с такими методами, как

* конструктор CustomStack(size\_t initialCapacity), который присваивает полю mCapacity, переданное значение, выделяет память под массив целых чисел и присваивает полю mIndex значение -1.
* конструктор CustomStack(), который вызывает другой конструктор с размером стека равным 10.
* деструктор ~CustomStack(), который очищает выделенную под массив память.
* метод void push(int val) проверяет, что размера стека достаточно для нового элемента, увеличивает значение поля, отвечающего за индекс, и помещает по номеру индекса элемент в массив.
* метод void pop() проверяет, что массив не пуст, иначе выводится сообщение об ошибке и завершается программа, а так же уменьшает значение поля mIndex на один, как бы удаляя последний элемент.
* метод int top() проверяет, что массив не пуст, иначе выводится сообщение об ошибке и завершается программа, а так же возвращает значение последнего элемента массива.
* метод size\_t size() const возвращает размер стека, то есть индекс последнего элемента, увеличенный на один.
* метод void extend(int n), который расширяет размер стека на n.
* приватный метод size\_t getNewCapacity() const возвращает новую емкость для массива данных.
* приватный метод void ensureSpace() проверяет, достигла ли текущая емкость массива данных предела. Если это так, то вызывает метод getNewCapacity() для получения новой емкости и затем вызывает метод resize() для изменения емкости массива данных.
* приватный метод void resize(size\_t newCapacity) изменяет размер массива, на переданный и создает новый массив нового размера, куда копирует данные из старого массива, старый массив удаляется, и указатель на новый массив и значение новой емкости обновляются.

В main’e создается экземпляр класса CustomStack и с помощью цикла while считываются команды пользователя, в зависимости от которых вызываются определенные методы для стека, пока не поступит команда "cmd\_exit". После цикла в консоль выводится сообщение "bye".

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 |
|  | 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 | Выходные данные корректны. |
|  | cmd\_push 1  cmd\_top  cmd\_pop  cmd\_size  cmd\_pop | ok  1  1  0  error | Случай с некорректным удалением обрабатывается корректно. |

## Выводы

Были подробно изучены такие структуры данных, как стек и очередь и требующиеся для их реализации механизмы языка С++. В соответствии с вариантом лабораторной работы был написан класс CustomStack, моделирующий поведение стека на основе массива.

# Приложение А Исходный код программы

Название файла: main.c

class CustomStack {

public:

CustomStack(size\_t initialCapacity) {

this->mCapacity = initialCapacity;

this->mData = new int[initialCapacity];

this->mIndex = -1;

}

CustomStack() : CustomStack(10) {

}

~CustomStack() {

delete[] this->mData;

}

void push(int val) {

this->ensureSpace();

this->mIndex++;

this->mData[this->mIndex] = val;

}

void pop() {

if (this->empty()) {

cout<<"error";

exit(0);

}

this->mIndex--;

}

int top(){

if (this->empty()) {

cout<<"error";

exit(0);

}

return this->mData[this->mIndex];

}

size\_t size() const {

return this->mIndex + 1;

}

bool empty() const {

return this->mIndex == -1;

}

void extend(int n) {

if (n <= 0) {

throw logic\_error("extend() called with a nonpositive argument");

}

this->resize(this->mCapacity + n);

}

protected:

size\_t mCapacity;

size\_t mIndex;

int \*mData;

private:

size\_t getNewCapacity() const {

return this->mCapacity \* 3 / 2 + 1;

}

void ensureSpace() {

if (this->mIndex + 1 == mCapacity) {

size\_t newCapacity = this->getNewCapacity();

this->resize(newCapacity);

}

}

void resize(size\_t newCapacity) {

if (newCapacity == mCapacity) {

return;

}

if (newCapacity < mCapacity) {

throw logic\_error("resize() called with a lower capacity");

}

int \*newData = new int[newCapacity];

copy(this->mData, this->mData + this->mCapacity, newData);

delete[] this->mData;

this->mData = newData;

this->mCapacity = newCapacity;

}

};

int main(){

CustomStack s = CustomStack();

string command;

int value;

while (command!="cmd\_exit"){

getline(cin, command);

if (command.find("cmd\_push") != string::npos){

value = stoi(command.substr(command.find(" ") + 1));

s.push(value);

cout<<"ok"<<endl;

}

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

value = s.top();

cout<<value<<endl;

}

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

value = s.top();

s.pop();

cout<<value<<endl;

}

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

value = s.size();

cout<<value<<endl;

}

}

cout<<"bye"<<endl;

return 0;