**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. 2384 |  | Кузьминых Е.М |
| Преподаватель |  | Гаврилов А.В. |

Санкт-Петербург

2023

**Цель работы.**

Написать реализацию стека с помощью класса, а также api для взаимодействия со стеком на языке C++.

**Задачи.** Вариант 3

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.

Примечания:

Указатель на массив 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.

Методы не должны выводить ничего в консоль.

**Выполнение работы.**

Определяется класс *CustomStack*, который представляет собой стек. Конструктор класса *CustomStack* может принимать целочисленный параметр *maxSize*, который задает максимальный размер стека. Если параметр не указан, то стек создается с максимальным размером по умолчанию, равным 100.

Метод *push* добавляет новый элемент на вершину стека. Метод *pop* удаляет верхний элемент стека. Если стек пустой, то выводится сообщение об ошибке и программа завершается. Метод *top* возвращает значение верхнего элемента стека. Если стек пустой, то выводится сообщение об ошибке и программа завершается. Метод *size* возвращает текущий размер стека. Метод *empty* возвращает *true*, если стек пустой.

Метод *extend* позволяет увеличивать размер стека на заданное количество элементов.

Затем определяется функция *main(),* которая использует класс *CustomStack* для создания стека и управления им.

В первой строке определяется константа *len\_input*, которая задает максимальную длину входной строки.

Затем создается объект класса *CustomStack* с максимальным размером 100. Создается массив строк *comands*, который содержит 5 команд для управления стеком: "cmd\_push", "cmd\_pop", "cmd\_top", "cmd\_size", "cmd\_exit".

Далее создается бесконечный цикл *while(true)*, который считывает входную строку *input* с помощью функции *cin*, а затем проверяет ее на соответствие одной из команд. Если входная строка соответствует команде "cmd\_push", то считывается строка el\_push, которая представляет собой значение, которое нужно добавить в стек. После этого выбирается номер команды (choice), соответствующей входной строке, и выполняется соответствующая операция с помощью оператора *switch*. Если входная строка не соответствует ни одной из команд, то выводится сообщение об ошибке.

Оператор *switch* выполняет операции в зависимости от выбранной команды: добавляет элемент в стек, выводит верхний элемент и удаляет его из стека, выводит верхний элемент, выводит размер стека, завершает программу.

В конце цикла переменная *choice* устанавливается в 0, чтобы при следующей итерации цикла можно было считать новую входную строку.

**Тестирование.**

|  |  |  |  |
| --- | --- | --- | --- |
| № | 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 |

**Вывод.**

В ходе выполнения лабораторной работы были изучены основы языка C++, написана реализация стека на массиве с помощью класса и *api* для взаимодействия со стеком. Кроме, был написан цикл *while* и *switch case*, позволяющий пользователю вводить команды в консоль для взаимодействия со стеком.

**Приложение А.**

**Исходный код программы.**

#include <iostream>

#include <string>

#include <cstring>

class CustomStack {

public:

CustomStack(){}

CustomStack(int maxSize) {

sizeOfStack = 0;

mData = new int[maxSize];

}

~CustomStack() {

delete[] mData;

}

void push(int newElement) {

mData[sizeOfStack] = newElement;

sizeOfStack++;

}

void pop() {

if(sizeOfStack==0){

cout<<"error"<<endl;

exit(0);

}

else

{

sizeOfStack--;

}

}

int top() {

if(sizeOfStack==0){

cout<<"error"<<endl;

exit(0);

}

else

{

return mData[sizeOfStack - 1];

}

}

int size() {

return sizeOfStack;

}

bool empty() {

return sizeOfStack == 0;

}

void extend(int n) {

int \*data\_ext = new int[sizeOfStack + n];

copy(mData, mData + sizeOfStack, data\_ext);

delete[] mData;

mData = data\_ext;

}

protected:

int \*mData;

int sizeOfStack;

};

int main()

{

const int len\_input = 10;

CustomStack stack(100);

char comands[5][len\_input] = {"cmd\_push","cmd\_pop","cmd\_top",

"cmd\_size","cmd\_exit"};

int choice;

char el\_push[20];

char input[len\_input];

while(true){

cin >> input;

if(!strcmp(input,comands[0])){

cin >> el\_push;

choice = 1;

}

for(int j = 1;j<5;j++){

if(!strcmp(input,comands[j])){

choice = j+1;

}

}

switch (choice) {

case 1:

stack.push(atoi(el\_push));

cout<<"ok"<<endl;

break;

case 2:

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

stack.pop();

break;

case 3:

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

break;

case 4:

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

break;

case 5:

cout << "bye\n";

exit(0);

break;

default:

cout << "try again\n";

}

choice = 0;

}

return 0;

}