**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. 3343 |  | Пименов П.В. |
| Преподаватель |  | Государкин Я.С. |

Санкт-Петербург

2024

## Цель работы

Изучить общие понятия о таких структурах данных, как стек и очередь. Написать программу на языке C++, в которой реализован стек на основе массива.

## Задание

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.
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

Требуемая в задании структура данных успешно реализована.

Описание структуры класса CustomStack:

* public
  + *CustomStack()* – конструктор, выделяет память на внутренний массив из 10 элементов, инициализирует переменные mIndex и mCapacity
  + *~CustomStack()* – деструктор, очищает память, выделенную под внутренний массив
  + *void push(int val)* – метод, добавляющий элемент в стек
  + *void pop()* – метод, удаляющий последний элемент из стека
  + *int top()* – метод, возвращающий значение последнего элемента в списке
  + *size\_t size()* – метод, возвращающий размер стека
  + *bool empty()* – метод, возвращающий логическое значение – пустой массив или нет
  + *void extend(int n)* – метод, расширяющий внутренний массив на n ячеек
* private
  + *size\_t mIndex* – поле, содержащее индекс последнего элемента в стеке
  + *size\_t mCapacity* – поле, содержащее вместимость внутреннего массива
* protected
  + *int\* mData* – внутренний массив стека

В функции main происходит создание стека, считывание новой команды, поступающей на ввод с новой строки, определение, какая именно команда была введена, выполнение команды, обработка возможных ошибок.

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 | Программа работает корректно. |
| 2. | cmd\_push 1  cmd\_push 2  cmd\_size  cmd\_exit | ok  ok  2  bye | Программа работает корректно. |
| 3. | cmd\_destroy | error | Программа работает корректно. |

## Выводы

Были изучены общие понятия о таких структурах данных, как стек и очередь, написана программа на языке C++, в которой реализован стек на основе массива.

# Приложение А Исходный код программы

Название файла: main.c

#define STACK\_DEFAULT\_SIZE 10

class CustomStack {

public:

CustomStack() {

this->mIndex = -1;

this->mCapacity = STACK\_DEFAULT\_SIZE;

this->mData = (int\*)malloc(this->mCapacity \* sizeof(int));

}

~CustomStack() { free(mData); }

void

push(int val) {

if (size() == this->mCapacity) {

this->mCapacity += STACK\_DEFAULT\_SIZE;

this->mData = (int\*)realloc(this->mData, this->mCapacity \* sizeof(int));

}

this->mData[++(this->mIndex)] = val;

}

void

pop() {

if (empty()) {

throw 1;

}

this->mIndex -= 1;

}

int

top() {

if (empty()) {

throw 2;

}

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

}

size\_t

size() {

return this->mIndex + 1;

}

bool

empty() {

return this->mIndex == -1;

}

void

extend(int n) {

if (n <= 0) {

throw 3;

}

this->mCapacity += n;

this->mData = (int\*)realloc(this->mData, mCapacity \* sizeof(int));

}

private:

size\_t mIndex;

size\_t mCapacity;

protected:

int\* mData;

};

int

main() {

CustomStack stack = CustomStack();

char buffer[51];

while (fgets(buffer, 51, stdin)) {

try {

if (strstr(buffer, "cmd\_push")) {

int a;

sscanf(buffer, "cmd\_push %d", &a);

stack.push(a);

cout << "ok" << endl;

} else if (strstr(buffer, "cmd\_pop")) {

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

stack.pop();

} else if (strstr(buffer, "cmd\_top")) {

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

} else if (strstr(buffer, "cmd\_size")) {

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

} else if (strstr(buffer, "cmd\_exit")) {

cout << "bye" << endl;

break;

} else {

throw 4;

}

} catch (int e) {

cout << "error" << endl;

break;

}

}

return 0;

}