**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МОЭВМ**

отчет

**по лабораторной работе№1**

**по дисциплине «Программирование»**

Тема: **Моделирование стека.**

|  |  |  |
| --- | --- | --- |
| Студент гр. 6304 |  | Васильев А.А. |
| Преподаватель |  | Берленко Т.А. |

Санкт-Петербург

2017

**Оглавление**

[**Цель работы** 3](#_Toc479585953)

[**Формулировка задачи** 3](#_Toc479585954)

[**Ход работы** 4](#_Toc479585955)

[**1.** **Реализация интерфейса** 4](#_Toc479585956)

[**2.** **Детектирование команд** 5](#_Toc479585957)

[**Вывод** 7](#_Toc479585958)

[**Приложение 1** 8](#_Toc479585959)

**Цель работы**

Моделирование стека с помощью массива и создание основных методов работы со стеком.

**Формулировка задачи**

Требуется написать программу, моделирующую работу стека, реализовав перечисленные ниже методы. Программе на вход подается последовательность команд с новой строки (не более 100 команд), в зависимости от которых программа выполняет ту или иную операцию и выводит результат ее выполнения с новой строки.

*Перечень команд:*

* push n - добавляет целое число n в стек. Программа должна вывести "ok";
* pop - удаляет из стека последний элемент и выводит его значение на экран;
* top - программа должна вывести верхний элемент стека на экран не удаляя его из стека;
* size - программа должна вывести количество элементов в стеке;
* exit - программа должна вывести "bye" и завершить работу.

Если в процессе вычисления возникает ошибка (например, вызов метода pop при пустом стеке), программа должна вывести "error" и завершиться.

Стек требуется реализовать самостоятельно на базе массива.

**Ход работы**

1. **Реализация интерфейса**

При реализации стека на базе массива, требуется обратить внимание на работу с памятью. Память под стек выделяется блоками, чтобы не использовать часто функцию **realloc()**.

Также требуется учесть возможное добавление дополнительных команд, для этого реализуется отдельная функция **check\_correction()**, которая распознает введенную команду и аргумент к ней (если он есть).

Требуемые в данной лабораторной работе методы работы со стеком не требуют создания дополнительных функций.

#define COMMANDS 100

int main()

{

int pushing\_value = 0; //Добавляемый элемент

int\* stack = (int\*)malloc(sizeof(int)\*COMMANDS); //Указатель на стек

int multiplier = 1; //Кол-во выделенных сейчас блоков памяти размером COMMANDS.

int len = 0; //Текущий размер стека

char\* choice = (char\*)malloc(sizeof(char)\*20); //Строка для команд.

while (1)

{

/\* Если кол-во элементов равно количеству выделенной в данный момент памяти, то

выделяется доп память размером COMMANDS\*/

if(len == COMMANDS\*multiplier)

stack = (int\*)realloc(stack, sizeof(int)\*COMMANDS\*(++multiplier));

fgets(choice, 20, stdin); //Считываем команду и удаляем ‘\n’ (если есть)

if(strchr(choice, '\n')) \*strchr(choice, '\n') = '\0';

/\* Передаем в проверяющую функцию команду и проверяем возвращаемое значение\*/

switch(check\_correction(choice, &pushing\_value))

{

case 1: //команда pop

if(len)

{

printf("%d\n", stack[len-1]);

len--;

}

else //Если стек пуст, то программа завершает работу.

{

printf("error\n");

return 0;

}

break;

case 2: //команда top

if(len)

printf("%d\n", stack[len-1]); //вывод верхнего элемента

else

{

printf("error\n");

return 0;

}

break;

case 3: //команда size

printf("%d\n", len);

break;

case 4: //команда exit

printf("bye\n");

return 0;

break;

case 5: //команда push

stack[len++] = pushing\_value;

printf("ok\n");

break;

case 0: //при вводе некорректной команды или

default: //пустой строки работа продолжается

continue;

break;

}

}

return 0;

}

1. **Детектирование команд**

Как было сказано выше, для распознавания команд и аргументов используется отдельная функция, которая возвращает определенное значение при каждой команде. Также функция предусматривает ввод некорректной команды или пустой строки. В этом случае работа программы будет продолжаться.

/\* Данная функция проверяет корректность введеной команды

и возвращает, в зависимости от команды, определенное целое.

Также для команды push определяется аргумент (value). \*/

int check\_correction(char\* choice, int\* value)

{

if(!choice) return 0; //Проверка на пустую строку.

if(strcmp(choice, "pop") == 0) return 1;

if(strcmp(choice, "top") == 0) return 2;

if(strcmp(choice, "size") == 0) return 3;

if(strcmp(choice, "exit") == 0) return 4;

/\* Если строка содержит команду push, то ищем её аргумент\*/

if (strstr(choice, "push ") && strlen(choice) >= 6)

{

choice = choice + sizeof(char)\*5;

\*value = atoi(choice);

return 5;

}

return 0; //Во всех остальных случаях возвращается 0.

}

**Вывод**

В ходе данной лабораторной работы был смоделирован стек с использованием массива. Для этого использовались стандартные возможности и функции языка Си. Также была реализована возможность добавления дополнительных команд для работы со стеком.

**Приложение 1**

Полный код

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define COMMANDS 100

/\* Данная функция проверяет корректность введеной команды

и возвращает, в зависимости от команды, определенное целое.

Также для команды push определяется аргумент (value). \*/

int check\_correction(char\* choice, int\* value)

{

if(!choice) return 0; //Проверка на пустую строку.

if(strcmp(choice, "pop") == 0) return 1;

if(strcmp(choice, "top") == 0) return 2;

if(strcmp(choice, "size") == 0) return 3;

if(strcmp(choice, "exit") == 0) return 4;

/\* Если строка содержит команду push, то ищем её аргумент\*/

if (strstr(choice, "push ") && strlen(choice) >= 6)

{

choice = choice + sizeof(char)\*5;

\*value = atoi(choice);

return 5;

}

return 0; //Во всех остальных случаях возвращается 0.

}

int main()

{

int pushing\_value = 0; //Добавляемый элемент

int\* stack = (int\*)malloc(sizeof(int)\*COMMANDS); //Указатель на стек

int multiplier = 1; //Кол-во выделенных сейчас блоков памяти размером COMMANDS.

int len = 0; //Текущий размер стека

char\* choice = (char\*)malloc(sizeof(char)\*20); //Строка для команд.

while (1)

{

/\* Если кол-во элементов равно количеству выделенной в данный момент памяти, то

выделяется доп память размером COMMANDS\*/

if(len == COMMANDS\*multiplier)

stack = (int\*)realloc(stack, sizeof(int)\*COMMANDS\*(++multiplier));

fgets(choice, 20, stdin); //Считываем команду и удаляем ‘\n’ (если есть)

if(strchr(choice, '\n')) \*strchr(choice, '\n') = '\0';

/\* Передаем в проверяющую функцию команду и проверяем возвращаемое значение\*/

switch(check\_correction(choice, &pushing\_value))

{

case 1: //команда pop

if(len)

{

printf("%d\n", stack[len-1]);

len--;

}

else //Если стек пуст, то программа завершает работу.

{

printf("error\n");

return 0;

}

break;

case 2: //команда top

if(len)

printf("%d\n", stack[len-1]); //вывод верхнего элемента

else

{

printf("error\n");

return 0;

}

break;

case 3: //команда size

printf("%d\n", len);

break;

case 4: //команда exit

printf("bye\n");

return 0;

break;

case 5: //команда push

stack[len++] = pushing\_value;

printf("ok\n");

break;

case 0: //при вводе некорректной команды или

default: //пустой строки работа продолжается

continue;

break;

}

}

return 0;

}