МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №2

по дисциплине «Программирование»

Тема: Динамические структуры данных.

Студент гр. 7383 _____ Бергалиев М.А.

Преподаватель _____ Берленко Т.А.

Санкт-Петербург

2018

ОГЛАВЛЕНИЕ

1. ЦЕЛЬ РАБОТЫ

Цель работы: ознакомиться на практике с динамическими структурами данных.

Формулировка задачи: Требуется написать программу, моделирующую работу стека, реализовав перечисленные ниже методы. Программе на вход подается последовательность команд с новой строки, 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 **списка**.

2. РЕАЛИЗАЦИЯ ЗАДАЧИ

В файле Stack.h описаны стек и элементы списка и объявлены функции для работы со стеком.

В файле Stack.c реализованы функции для работы со стеком:

initStack — создает пустой стек.

push — добавляет наверх стека новый элемент.

pop — удаляет верхний элемент стека и возвращает хранимое в нем значение.

top — возвращает значение, хранимое в верхнем элементе стека.

size — возвращает количество элементов в стеке.

isEmpty — возвращает 1, если стек пуст, и 0 в противном случае.

delStack — удаляет элементы стека и стек, высвобождая выделенную память.

В функции main циклически считываются команды, пока не будет встречена неизвестная команда или команда exit. Введенная команда ищется в массиве команд и в зависимости от номера команды выполняются действия соответствующие команде. После выполнения всех команд вызывается функция delStack, удаляющая созданный стек.

3. ТЕСТИРОВАНИЕ

Программа была собрана в компиляторе GCC в OS Linux Ubuntu.

Тестовые случаи представлены в Приложении А.

По результатам тестирования было показано, что поставленная задача была выполнена.

4. ВЫВОД

Был реализован стек на базе списка. Его преимуществом являются динамическое выделение памяти и быстрота добавления нового элемента. Недостатками являются потеря производительности из-за динамического выделения памяти и выделенная память занимает место в оперативной памяти.

ПРИЛОЖЕНИЕ А. ТЕСТОВЫЕ СЛУЧАИ

Ввод	Вывод
push 1 top push 2 top pop size pop size exit	*ok* *1* *ok* *2* *2* *1* *1* *0* *bye*
size push 3 size top push 4 size top pop size top pop size pop	0 ok 1 3 ok 2 4 4 1 3 3 0 error
push 5 push 4 push 3 push 2 push 1 pop pop pop pop pop exit	ok ok ok ok ok 1 2 3 4 5 bye

ПРИЛОЖЕНИЕ Б. ИСХОДНЫЙ КОД ПРОГРАММЫ

Makefile:

```
Stack: main.o Stack.o
	gcc main.o Stack.o -o Stack
main.o: main.c Stack.h
	gcc -c main.c
Stack.o: Stack.c Stack.h
	gcc -c Stack.c
```

main.c:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "Stack.h"

int main(){
    Stack* stack = initStack();
    char input[10];
    int n;
    char commands[5][5] = {"exit", "pop", "push", "size", "top"};
    int run = 1;
    int command;
    while(run){
        fgets(input, 5, stdin);
        if(input[3] == '\n')
            input[3] = '\0';
        command = (char(*)[5])bsearch(input, commands, 5, sizeof(char[5]),
                        (int (*)(const void*,const void*))strcmp) - commands;
        switch(command){
            case 2:{
                scanf("%d", &n);
```

```
    fgetc(stdin);
    push(stack, n);
    printf("ok\n");
    break;
}
case 1:{
    if (!isEmpty(stack))
        printf("%d\n", pop(stack));
    else{
        printf("error\n");
        run = 0;
    }
    break;
}
case 4:{
    if (!isEmpty(stack))
        printf("%d\n", top(stack));
    else{
        printf("error\n");
        run = 0;
    }
    break;
}
case 3:{
    printf("%d\n", size(stack));
    fgetc(stdin);
    break;
}
case 0:{
    run = 0;
```

```
            printf("bye\n");
            break;
        }
        default:{
            printf("error\n");
            run = 0;
        }
    }
  }
  delStack(stack);
  return 0;
}
```

Stack.h:

```
#ifndef STACK_H
#define STACK_H
typedef struct StackElem{
  int value;
  struct StackElem *next;
}StackElem;

typedef struct Stack{
  struct StackElem *topElem;
}Stack;

Stack* initStack();

void push(Stack* stack, int n);

int pop(Stack* stack);
```

```
int top(Stack* stack);

int size(Stack* stack);

int isEmpty(Stack* stack);

void delStack(Stack* stack);

#endif
```

Stack.c:

```
#include <stdlib.h>
#include <string.h>
#include "Stack.h"

Stack* initStack(){
   Stack* stack = (Stack*) malloc(sizeof(Stack));
   stack->topElem = NULL;
   return stack;
}

void push(Stack* stack, int n){
   StackElem* elem = malloc(sizeof(StackElem));
   elem->value = n;
   elem->next = stack->topElem;
   stack->topElem = elem;
}

int pop(Stack* stack){
        int n = stack->topElem->value;
        StackElem* del = stack->topElem;
```

```
        stack->topElem = del->next;
        free(del);
        return n;
}

int top(Stack* stack){
        return stack->topElem->value;
}

int size(Stack* stack){
    StackElem* current = stack->topElem;
    int i = 0;
    while (current != NULL){
        i++;
        current = current->next;
    }
    return i;
}

int isEmpty(Stack* stack){
    if (stack->topElem == NULL)
        return 1;
    else return 0;
}

void delStack(Stack* stack){
    StackElem* del = stack->topElem;
    while (del != NULL){
        stack->topElem = del->next;
        free(del);
```

```
        del = stack->topElem;
    }
    free(stack);
}
```