# МИНОБРНАУКИ РОССИИ

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**ОТЧЕТ**

**по лабораторной работе №2**

**по дисциплине «Программирование» Тема: «Динамические структуры данных»**

|  |  |
| --- | --- |
| Студент гр. 7381 | Лауцюс М. |
| Преподаватель | Берленко Т.А. |

Санкт-Петербург 2018

# Цель работы.

Ознакомиться со стандартной библиотекой си, изучить ее функционал и возможности.

# Задание.

**Моделирование стека.**

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**.

# Основные теоретические положения.

**Структуры данных**

Помимо уже знакомых нам структур данных, таких как массивы и списки (односвязные, двусвязные, циклические) существует множество различных других. Наиболее распространенные это графы, деревья, очередь и стек.

**Стек**

Стек - это структура данных, в которой хранятся элементы в виде последовательности, организованной по принципу LIFO (Last In — First Out). Такую структуру данных можно сравнить со стопкой тарелок или магазином автомата. Стек не предполагает прямого доступа к элементам и список основных операций ограничивается операциями помещения элемента в стек и извлечения элемента из стека. Их принято называть PUSH и POP соответственно. Также, обычно есть возможность посмотреть на верхний элемент стека не извлекая его (TOP) и несколько других функций, таких как проверка на пустоту стека и некоторые другие.

Очередь

Очередь - эта структура данных, в которой хранятся элементы в виде последовательности, организованной по принципу FIFO (First In — First Out). Эта структура данных более естественна - например, очередь в магазине. Также как и стек, очередь не предполагает прямого доступа к элементам, а основные операции: добавление ENQ (enqueue) и извлечение DEQ(dequeue). Также обычно есть функции получения первого элемента без его извлечения, определения размера очереди, проверки на пустоту и некоторые другие.

**Реализация**

Рассмотрим способы реализации таких структур данных как стек и очередь. Фактически, обе структуры данных можно представлять в памяти либо в виде однонаправленного списка, либо в виде массива.

**Представление в виде списка**

При такой организации хранения элементов, операции добавления элемента в стек и операции удаления элемента из стека эквивалентны операциям над списком: добавление в голову и удаление из головы соответственно. Таким образом, каждый элемент имеет указатель на следующий, лежащий "ниже" него в стеке.

В случае очереди, добавление элемента эквивалентно вставке элемента в конец списка, а извлечение - удаление элемента из головы списка. Если вместе с указателем на голову списка хранить указатель на его последний элемент, операция вставки перестает быть затратной (ввиду отсутствия необходимости проходить список до конца каждый раз). Таким образом, каждый элемент имеет указатель на следующий в порядке очереди элемент.

**Представление в виде массива**

Стек можно легко реализовать на основе массива. Для этого достаточно хранить индекс "верхнего" элемента в стеке. Операция добавления сопровождается инкрементом этого индекса и записью в соответствующую ячейку нового значения. Операция извлечения сопровождается декрементом этого индекса. Дополнительно, может потребоваться реализовать возможность увеличения и уменьшения размера массива.

Реализовать на основе массива очередь немного сложнее. В отличие от стека, потребуется хранить два индекса - индекс первого элемента и индекс последнего. Вставка сопровождается инкрементом индекса последнего элемента и записью нового значения, а извлечение инкрементом индекса первого. В случае равенства индексов - очередь пуста. Проблема возникает в том случае, когда индекс последнего подбирается к границе массива, при этом начало массива уже не используется. Эту проблему можно решить, начав циклически использовать ячейки (в этой ситуации индекс последнего элемента может быть меньше индекса первого).

К недостаткам реализации в виде списка можно отнести более медленную работу и незначительное увеличение затрат памяти. К преимуществам - произвольный объем структуры данных.

**API**

API (application programming interface) - это некоторый набор функций, предоставляемых библиотекой или некоторым модулем для использования. API определяет функциональность, которую предоставляет модуль и интерфейс взаимодействия, позволяя абстрагироваться от его реализации. Таким образом, модуль представляет собой черный ящик, взаимодействие с которым возможно строго определенным образом. Например, при реализации стека можно определить следующий набор функций, которые будут представлять собой API для работы с нашей структурой данных:

void push(struct Stack \*stack, type value);

type pop(struct Stack \*stack);

type top(struct Stack \*stack);

int isEmpty(struct Stack \*stack);

struct Stack \*initStack();

Где type - тип данных, которые нам требуется хранить в стеке. В зависимости от реализации (на основе списка или массива) структура Struct может иметь, например, следующий вид:

// array stack

struct Stack{

 type arr[N];

 int topIndex;

};

// list stack

struct Stack{

 struct StackElem \*topElem;

};

struct StackElem{

 type value;

 struct StackElem \*next;

}

Следует отметить, что вне зависимости от реализации сохраняется единообразный интерфейс работы со стеком. Такое разделение интерфейса и реализации не только делает код более читаемым, но и упрощает его сопровождение. (Скажем, для изменения структуры хранения элементов, достаточно только изменить функции для работы с ними).

# Ход работы

Была написана программа, моделирующая работу стека и с реализацией методов, выполняющих команды, в виде функций. Программа на вход принимает 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 выводит **"ok"**
* **pop** - удаляет из стека последний элемент и выводит его значение на экран
* **top** - программа выводит верхний элемент стека на экран, не удаляя его из стека
* **size** - программа выводит количество элементов в стеке
* **exit** - программа выводит "**bye**" и завершает работу

Если в процессе вычисления возникает ошибка (например, вызов метода **pop** при пустом стеке), программа выводит "**error**" и завершается.

Стек реализован самостоятельно на базе **массива**.

# Вывод:

Было проведено ознакомление с такими структурами данных, как стек, очередь, а также их реализациями на основе списков и массивов. На основе полученных знаний была создана программа, реализующая стек на основе массива.

Исходный код проекта:

Файл Makefile

all:

gcc main.c

Файл main.c

# #include<stdio.h>

# #include<string.h>

# #include<stdlib.h>

# typedef struct{int \*arr;int size;}Stack;

# void error(Stack\* stack){

# printf("error");

# if(stack->arr!=NULL)free(stack->arr);

# exit(0);

# }

# void pop(Stack\* stack){

# if(stack->size==0){

# error(stack);

# }

# stack->size--;printf("%d\n", stack->arr[stack->size]); stack->arr=(int\*)realloc(stack->arr,stack->size\* sizeof (int));

# }

# void push(Stack\* stack){

# stack->size++;

# stack->arr=(int\*)realloc(stack->arr,stack->size\*sizeof(int));

# if(scanf("%d",&(stack->arr[stack->size-1]))==EOF)error(stack);

# printf("ok\n");

# }

# void top(Stack\* stack){

# if(stack->size==0){

# error(stack);

# }

# printf("%d\n", stack->arr[stack->size-1]);

# }

# void size(Stack\* stack){

# printf("%d\n",stack->size);

# }

# int main(){

# Stack stack;

# stack.size=0;

# stack.arr=(int\*)malloc(sizeof(int));

# char str[6];int i=0;char c;

# while(1){i=0;

# if((c=getchar())!='\n')

# ungetc(c,stdin);

# while((c=getchar())!=' '&&c!='\n'&&i<=6){

# str[i]=c;i++;

# }

# str[i]='\0';

# if(!strcmp(str,"pop"))

# pop(&stack);

# else if(!strcmp(str,"push"))

# push(&stack);

# else if(!strcmp(str,"exit")){

# printf("bye");

# if(stack.arr!=NULL)free(stack.arr);return 0;

# }

# else if(!strcmp(str,"top"))

# top(&stack);

# else if(!strcmp(str,"size"))

# size(&stack);

# else {error(&stack);}

# }

# }