|  |
| --- |
| **LAB REPORT**  DATA STRUCTURE AND IT’S APPLICATIONS |
|  |
|  |
| **by Lima Islam** |
| **10th October 2018** |

Contents

[1 Introduction to Data Structure 3](#_Toc526898010)

[2 Basic Data Structures 3](#_Toc526898011)

[2.1 Array 3](#_Toc526898012)

[2.1.1 1D array sum 3](#_Toc526898013)

[2.1.2 2D array sum 3](#_Toc526898014)

[2.2 Stack 4](#_Toc526898015)

[2.2.1 Basic Features 4](#_Toc526898016)

[2.2.2 Applications of Stack 4](#_Toc526898017)

[2.2.3 Implementation of Stack in C 4](#_Toc526898018)

[2.3 Queue 5](#_Toc526898019)

[2.3.1 Basic Features 6](#_Toc526898020)

[2.3.2 Implementation of Queue in C 6](#_Toc526898022)

[2.4 Linked List 7](#_Toc526898023)

[2.4.1 Introduction 7](#_Toc526898024)

[2.4.2 Implementation of Linked List in C 7](#_Toc526898026)

[2.4.3 Stack Implementation using Linked List (Algorithm) 9](#_Toc526898027)

[2.4.4 Queue Implementation using Linked List (Algorithm) 10](#_Toc526898028)

**1. Basic Data Structures**

**2.1 Array**

Arrays a kind of data structure that can store a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type. All arrays consist of contiguous memory locations. The lowest address corresponds to the first element and the highest address to the last element. There are two type of arrays, they are: 1. One-dimensional arrays 2. Multidimensional arrays

2.1.1 One dimensional Array

To declare an array in C, a programmer specifies the type of the elements and the number of elements required by an array as follows − type arrayName [arraySize]; This is called a single-dimensional array. The arraySize must be an integer constant greater than zero and type can be any valid C data type. For example, float mark[5] Data Structure 4 Here, we declared an array, mark, of floating-point type and size 5. Meaning, it can hold 5 floating-point values.

**Initializing**

There are various ways to initial a array. They are:

It's possible to initialize an array during declaration. For¬ example, int mark[5] = {19, 10, 8, 17, 9};

Another method to initialize array during declaration:

int mark[] = {19, 10, 8, 17, 9};

**Insert & Print**

Step 1: Declare array

Step 2: Insert the number of element, n of the array

Step 3: Use for loop to scan and print 0 to n element.

2.1.2 Two dimensional Array

C programming language allows multidimensional arrays. The simplest form of multidimensional array is the two-dimensional array. A two-dimensional array is, in essence, a list of one-dimensional arrays. To declare a two-dimensional integer array of size [x][y], you would write something as follows –

type arrayName [ x ][ y ];

**Initializing**

It's possible to initialize an array during declaration. For¬ example, int mark[2][2] = {10, 8, 17, 9};

**Insert & Print**

Step 1: Declare array

Step 2: Insert the number of row and column of the array Data Structure 6

Step 3: Use two for loop to scan and print elements.

**2.2 Stack**

2.2.1 Basic Features

A stack is a container of objects that are inserted and removed according to the Last In First Out (LIFO) principle.In this pushdown stacks only two operations are allowed – **push** (the item into the stack) and **pop** (the item ot of the stack.

2.2.2 Applications of Stack

Stack can be used in Balancing of symbols, infix to profix/prefix conversation, Redo-Undo features at many palces, forward & backward features in web browsers, Backtracking etc.

2.2.3 Implementation of Stack using array in C

**Algorithm**

/\* Stack operation Push and Pop algorithm using array\*/

void push(int data)

{

head++;

stack[head] = data;

}

int pop()

{

int data = stack[head];

head--;

return data;

}

void display()

{

printf("Data in your stack\n");

int i;

for(i=0;i<=head;i++)

{

printf ("%d ",stack[i]);

}

}

**2.3 Queue**

2.3.1 Basic Features

A queue is a useful data structure in programming. It is similar to the ticket queue outside a cinema hall, where the first person entering the queue is the first person who gets the ticket. Queue follows the First In First Out(FIFO) rule - the item that goes in first is the item that comes out first too.

Enqueue: Add element to end of queue, having a particular pointer

Dequeue: Remove element from front of queue, having a particular pointer

2.3.2 Implementation of Queue using array in C

#include<stdio.h>

#define SIZE 10

int queue[SIZE];

int head = -1; //empty queue

int tail = 0;

void enQueue(int value)

{

if(head == SIZE-1) //checking the queue is full or not

{

printf("\nQueue is Full!");

}

else

{

head++;

queue[head] = value;

}

}

void deQueue()

{

if(head == -1) //checking the queue is empty or not

{

printf("\nQueue is Empty!!! ");

}

else

{

printf("\nDeleted : %d", queue[tail]);

tail++;

}

}

void display()

{

int i;

printf("\nQueue elements are :\n");

for(i=tail; i<=head; i++)

printf("%d\t",queue[i]);

}

void main()

{

enQueue(10); //calling enqueue function

enQueue(20); //calling enqueue function

enQueue(30); //calling enqueue function

enQueue(40); //calling enqueue function

display(); //calling display function

deQueue(); //calling dequeue function

deQueue(); //calling dequeue function

display(); //calling display function

}

**2.4 Linked List**

2.4.1 Introduction

A linked list is a sequential access data structure where each element can be accesed on in particular order. A typical illustration of sequential access is a roll paper or tape, if we want to get a data all prior material must be unrolled

2.4.2 Implementation of Linked List in C

#include <stdio.h>

struct Node{

int data;

struct Node \*next;

};

struct Node \*head = NULL;

void insertAtBeginning(int value){

struct Node \*newNode;

newNode = (struct Node\*)malloc(sizeof(struct Node));

newNode->data = value;

newNode->next = NULL;

if(head == NULL)

{ head = newNode; }

else{

newNode->next = head;

head = newNode;}

}

void display(){

struct Node \*temp = head;

printf("\n\nList elements are - \n");

while(temp->next != NULL){

printf("%d ",temp->data);

temp = temp->next;

}

printf("%d ",temp->data);

}

void insertAtEnd(int value){

struct Node \*newNode;

newNode = (struct Node\*)malloc(sizeof(struct Node));

newNode->data = value;

newNode->next = NULL;

if(head == NULL){

head = newNode;

}

else{

struct Node \*temp = head;

while(temp->next != NULL){

temp = temp->next;

}

temp->next = newNode;

}

}

void removeBeginning(){

struct Node \*temp = head;

if(head->next == NULL){

head = NULL;

free(temp);}

else{

head = temp->next;

free(temp);}

}

}

void removeEnd(){

{

struct Node \*temp1 = head,\*temp2;

if(head->next == NULL){

head = NULL;

}

else {

while(temp1->next != NULL){

temp2 = temp1;

temp1 = temp1->next;

}

temp2->next = NULL;

}

free(temp1);

}

}

int main()

{

insertAtEnd(10);

insertAtEnd(20);

insertAtBeginning(40);

insertAtBeginning(50);

insertAtEnd(100);

display();

removeBeginning();

display();

removeEnd();

display();

return 0;

}

2.4.3 Stack Implementation using Linked List (Algorithm)

Stack implementation using Linked List

Here is the algorithm and C code for implementation of STACK(without loss of generality, integer type data has been used here. One can also use character or float or string type data.) using LINKED LIST.

**Algorithm**

/∗implementation of s t a c k using linked list algorithm∗/

struct Node{

int data ;

struct Node ∗next ;

};

struct Node ∗top = NULL ; / / initializes top at null

push ( ){

p r i n tf(”\enenter number of elements to push ”) ;

read the number n [ say ] ;

p r i n t f ” e n t e r the elements ” ;

f o r ( i = 1 ; i<= n ; i ++){

s t r u c t Node ∗temp = ( Node∗) malloc ( s i z e o f ( s t r u c t Node ) ) ;

temp−>data = element ;

temp−>next = top ;

top = temp ;} / / insertion takes place where the top pointer points

}

pop ( ){

i f ( top == NULL )

printf (“ stack is empty ” );

e l s e{

struct Node ∗p = top ;

top = top−>next ;

f r e e ( p ) ;}

}

p r i n t ( )

{

struct Node∗temp = top ;

i f ( temp == NULL )

printf (“the stack is empty ”) ;

e l s e{

printf(“ s t a c k i s ”) ;

while ( temp != NULL )

{

printf ( temp−>data ) ;

temp = temp−>next ;}

}

}

/∗implementation of stack using linked list∗/

2.4.4 Queue Implementation using Linked List (Algorithm)

#include <stdio.h>

struct Node

{

int data;

struct Node \*next;

};

struct Node \*front = NULL;

struct Node \*head = NULL;

void enQueue(int value)

{

struct Node \*newNode;

newNode = (struct Node\*)malloc(sizeof(struct Node));

newNode->data = value;

newNode->next = NULL;

if(front == NULL)

{

front = head = newNode;

}

else

{

head -> next = newNode;

head = newNode;

}

}

void deQueue()

{

struct Node \*temp = front; //if front == NULL then the queue is empty, thus there's no elements to dequeue

front = front -> next;

printf("\nDeleted element: %d\n", temp->data);

free(temp);

}

void display()

{

if(front == NULL)

printf("\nQueue is Empty!!!\n");

else

{

struct Node \*temp = front;

while(temp->next != NULL)

{

printf("%d\t",temp->data);

temp = temp -> next;

}

printf("%d\n",temp->data);

}

}

int main()

{

enQueue(10);// 10

enQueue(20); // 10 20

enQueue(40); // 10 20 40

enQueue(50); // 10 20 40 50

display();

deQueue(); // 20 40 50

display();

return 0;

}

© 2018 GitHub, Inc.

Terms

Privacy

Security

Status

Help

Contact GitHub

Pricing

API

Training

Blog

About

Press h to open a hovercard with more details.