

Data Structure

Lab Report

**Submitted by Submitted to**

**Md.Sakhaoat Hossain Richard Philip**

**ID : 163432076 Sr. Lecturer of**

**Batch : 43 Department of CSE**

City University Bangladesh

Date : 10.10.2018

**Midterm**

**1.Array**

**2.Stack**

**3.Queue**

**4.Link list**

**DATA STRUCTURE**

Data Structure is a systematic way to organize data in order to use it efficiently. Almost every enterprise application uses various types of data structures in one or the other way.

Following terms are the foundation terms of a data structure.

* **Interface** − Each data structure has an interface. Interface represents the set of operations that a data structure supports. An interface only provides the list of supported operations, type of parameters they can accept and return type of these operations.
* **Implementation** − Implementation provides the internal representation of a data structure. Implementation also provides the definition of the algorithms used in the operations of the data structure.

**Characteristics of a Data Structure**

* **Correctness** − Data structure implementation should implement its interface correctly.
* **Time Complexity** − Running time or the execution time of operations of data structure must be as small as possible.
* **Space Complexity** − Memory usage of a data structure operation should be as little as possible.

**Data Object**

Data Object represents an object having a data.

**Data Type**

Data type is a way to classify various types of data such as integer, string, etc. which determines the values that can be used with the corresponding type of data, the type of operations that can be performed on the corresponding type of data. There are two data types −

* Built-in Data Type and .Derived Data Type

**Built-in Data Type**

Those data types for which a language has built-in support are known as Built-in Data types. For example, most of the languages provide the following built-in data types.

* Integers
* Boolean (true, false)
* Floating (Decimal numbers)
* Character and Strings

**Derived Data Type**

Those data types which are implementation independent as they can be implemented in one or the other way are known as derived data types. These data types are normally built by the combination of primary or built-in data types and associated operations on them. For example −

* Array
* Stack
* Queue
* Linked List

**Array :**

Objective(s):

To be familiar Data structure of C-programming.

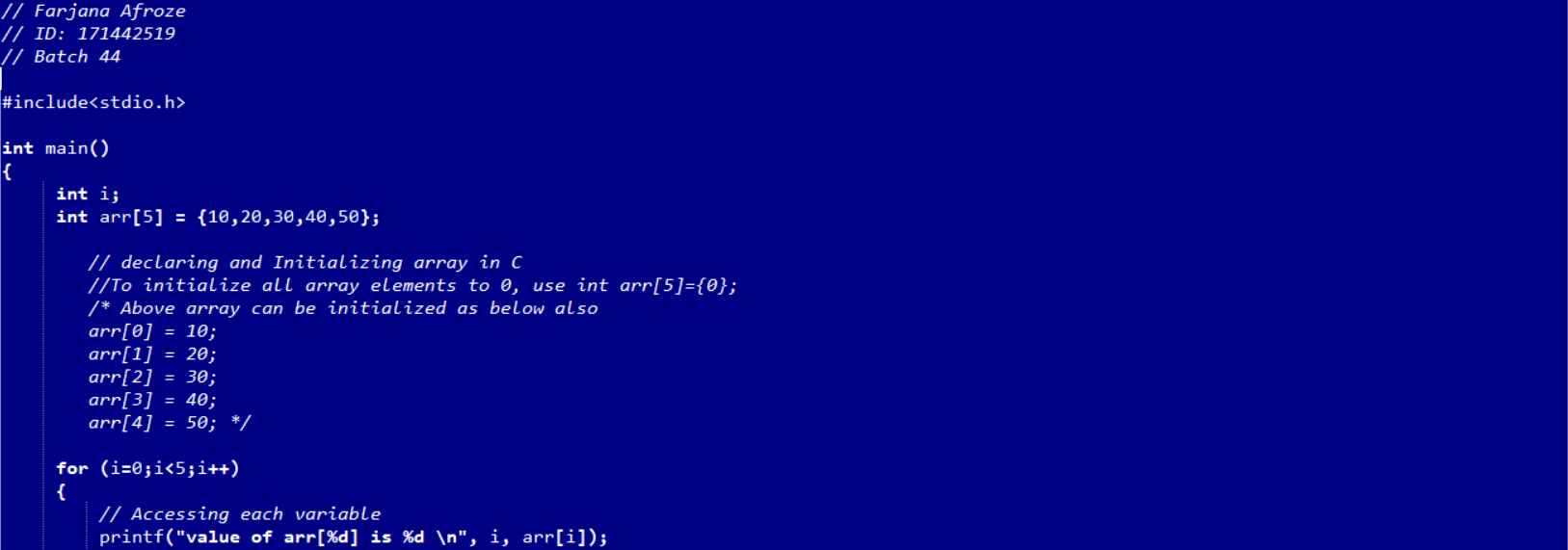
To learn problem solving techniques using C

An array is a collection of one or more values of the same type. Each value is called an element of the array. The elements of the array share the same variable name but each element has its own unique index number (also known as a subscript). An array can be of any type, For example: int, float, char etc. If an array is of type intthen it's elements must be of type int only.

Arrays can be single or multidimensional. The number of subscript or index determines dimensions of the array. An array of one dimension is known as a one-dimensional array or 1-D array, while an array of two dimensions is known as a two-dimensional array or 2-D array.

**1-D array :**

The following declares an array called ‘numbers’ to hold 5 integers and sets the first and last elements. C arrays are always indexed from 0. So the first integer in ‘numbers’ array is numbers [0] and the last is numbers [4].



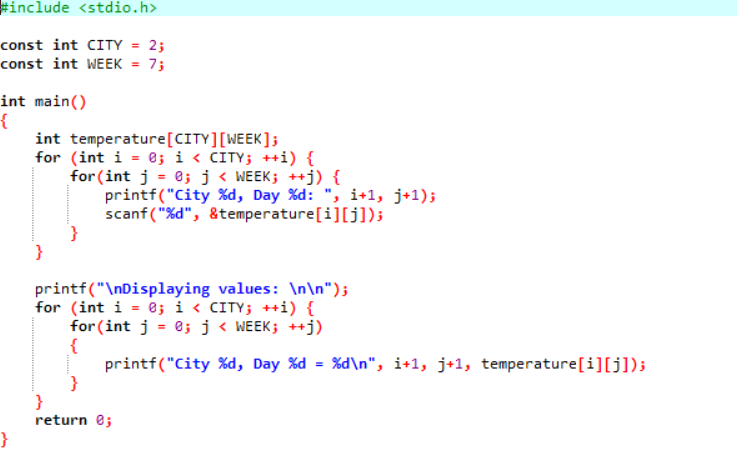
**Sum of 1-D array:**

Finding sum of 1-D array elements is easy when you know how to iterate through array elements.



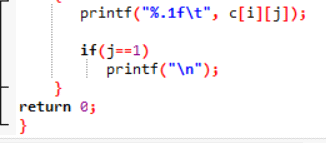
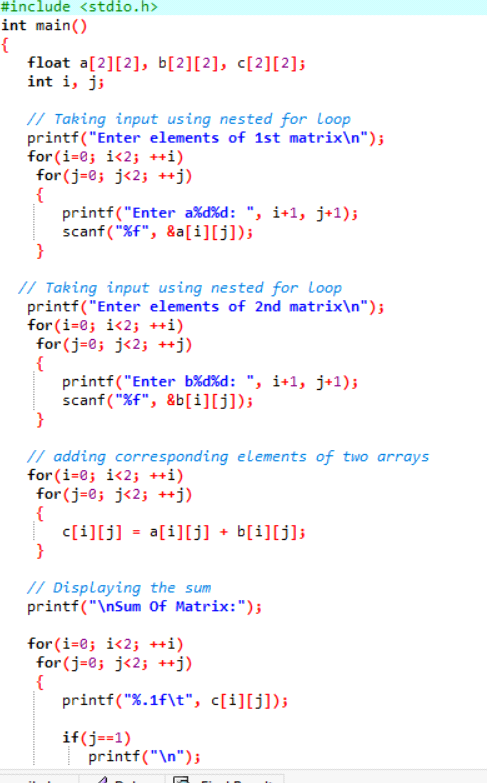
**2-D Array :**

An array of arrays is known as 2D array. The two dimensional (2D) array in C Programming is also known as matrix. A matrix can be represented as a table of rows and columns. Before we discuss more about two Dimensional array let’s have a look at the following C program.



**Sum of 2-D array:**

C program to find the sum of two matrices of order 2\*2 using multidimensional arrays.

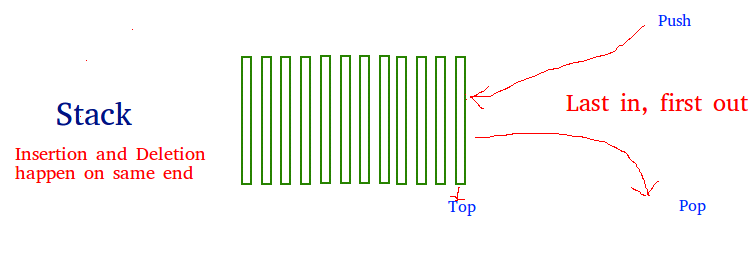


**Stack :**

Stack is a linear data structure which follows a particular order in which the operations are performed. The order may be LIFO (Last In First Out) or FILO(First In Last Out).

Mainly the following three basic operations are performed in the stack:

* **Push:**Adds an item in the stack. If the stack is full, then it is said to be an Overflow condition.
* **Pop:** Removes an item from the stack. The items are popped in the reversed order in which they are pushed. If the stack is empty, then it is said to be an Underflow condition.
* **Peek or Top:** Returns top element of stack.
* **isEmpty:**Returns true if stack is empty, else fals



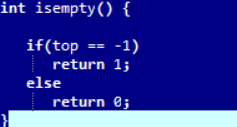
**How to Understand a stack practically?**

There are many real life examples of stack. Consider the simple example of plates stacked over one another in canteen. The plate which is at the top is the first one to be removed, i.e. the plate which has been placed at the bottommost position remains in the stack for the longest period of time. So, it can be simply seen to follow LIFO/FILO order.

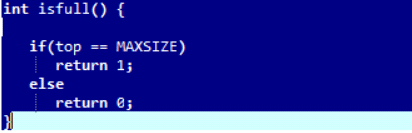
Implementation of peek() function in C programming language −



Implementation of isempty() function in C programming language is slightly different. We initialize top at -1, as the index in array starts from 0. So we check if the top is below zero or -1 to determine if the stack is empty. Here's the code –

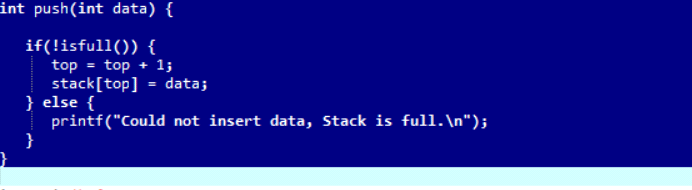


Implementation of isfull() function in C programming language



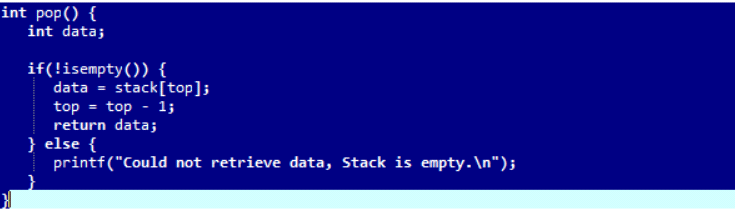
**Push Operation**

Implementation of this algorithm in C, is very easy. See the following code –



**Pop Operation**

Accessing the content while removing it from the stack, is known as a Pop Operation. In an array implementation of pop() operation, the data element is not actually removed, instead top is decremented to a lower position in the stack to point to the next value. But in linked-list implementation, pop() actually removes data element and deallocates memory space.

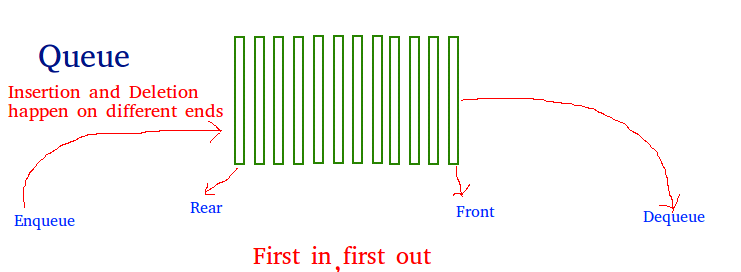


**For a complete stack program in C programming language, please following my files**

**QUEUE**

Queue is an abstract data structure, somewhat similar to Stacks. Unlike stacks, a queue is open at both its ends. One end is always used to insert data (enqueue) and the other is used to remove data (dequeue). Queue follows First-In-First-Out methodology, i.e., the data item stored first will be accessed first.

A Queue is a linear structure which follows a particular order in which the operations are performed. The order is First in First Out (FIFO). A good example of a queue is any queue of consumers for a resource where the consumer that came first is served first. The difference between stacks and queues is in removing. In a stack we remove the item the most recently added; in a queue, we remove the item the least recently added.



**Basic Operation**

Queue operations may involve initializing or defining the queue, utilizing it, and then completely erasing it from the memory. Here we shall try to understand the basic operations associated with queues –

enqueue() − add (store) an item to the queue.

dequeue() − remove (access) an item from the queue.

Few more functions are required to make the above-mentioned queue operation efficient. These are –

peek() − Gets the element at the front of the queue without removing it.

isfull() − Checks if the queue is full.

isempty() − Checks if the queue is empty

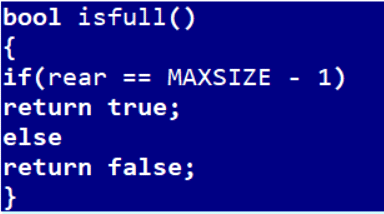
**Queue Representation**

As we now understand that in queue, we access both ends for different reasons. The following diagram given below tries to explain queue representation as data structure –

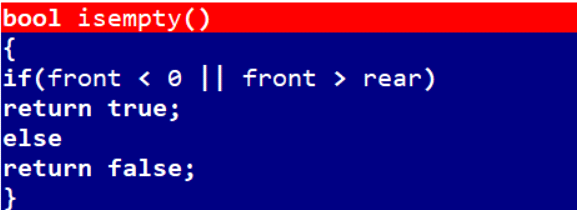
Implementation of peek() function in C programming language –



Implementation of isfull() function in C programming language –



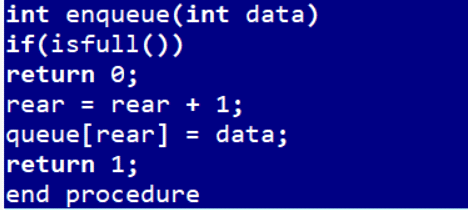
If the value of front is less than MIN or 0, it tells that the queue is not yet initialized, hence empty. Here's the C programming code –



**Enqueue Operation**

Queues maintain two data pointers, front and rear. Therefore, its operations are comparatively difficult to implement than that of stacks.

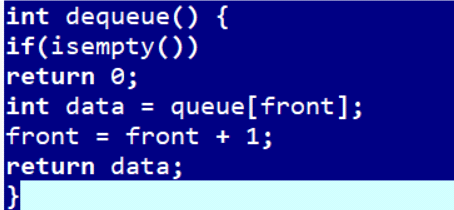
Implementation of enqueue() in C programming language –



**Dequeue Operation**

Accessing data from the queue is a process of two tasks − access the data where front is pointing and remove the data after access.

Implementation of dequeue() in C programming language –



**For a complete queue program in C programming language, please following my files**

**A LINKED LIST**

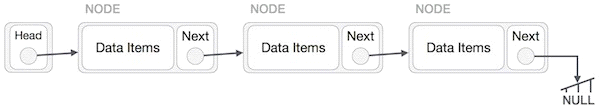
A linked list is a sequence of data structures, which are connected together via links.

Linked List is a sequence of links which contains items. Each link contains a connection to another link. Linked list is the second most-used data structure after array. Following are the important terms to understand the concept of Linked List.

* **Link** − Each link of a linked list can store a data called an element.
* **Next** − Each link of a linked list contains a link to the next link called Next.
* **LinkedList** − A Linked List contains the connection link to the first link called First.

**Linked List Representation**

Linked list can be visualized as a chain of nodes, where every node points to the next node.



As per the above illustration, following are the important points to be considered.

* Linked List contains a link element called first.
* Each link carries a data field(s) and a link field called next.
* Each link is linked with its next link using its next link.
* Last link carries a link as null to mark the end of the list.

**Types of Linked List**

Following are the various types of linked list.

* **Simple Linked List** − Item navigation is forward only.
* **Doubly Linked List** − Items can be navigated forward and backward.
* **Circular Linked List** − Last item contains link of the first element as next and the first element has a link to the last element as previous.

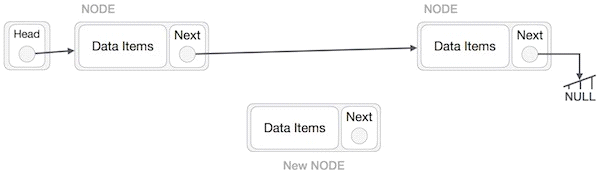
**Basic Operations**

Following are the basic operations supported by a list.

* **Insertion** − Adds an element at the beginning of the list.
* − Deletes an element at the beginning of the list.
* **Deletion** **Display** − Displays the complete list.
* **Search** −Searches an element using the given key.
* **Delete** − Deletes an element using the given key.

**Insertion Operation**

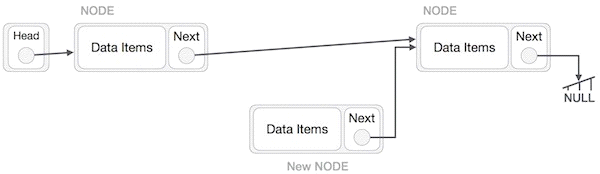
Adding a new node in linked list is a more than one step activity. We shall learn this with diagrams here. First, create a node using the same structure and find the location where it has to be inserted.



Imagine that we are inserting a node **B** (NewNode), between **A** (LeftNode) and **C** (RightNode). Then point B.next to C −

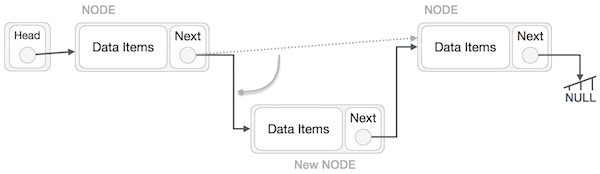
NewNode.next−>RightNode;

It should look like this –



Now, the next node at the left should point to the new node.

LeftNode.next−>



NewNode;

This will put the new node in the middle of the two. The new list should look like this –



Similar steps should be taken if the node is being inserted at the beginning of the list. While inserting it at the end, the second last node of the list should point to the new node and the new node will point to NULL.

**Deletion Operation**

Deletion is also a more than one step process. We shall learn with pictorial representation. First, locate the target node to be removed, by using searching algorithms.

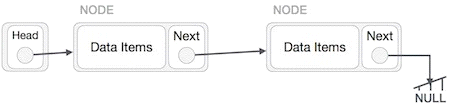


The left (previous) node of the target node now should point to the next node of the target node −

LeftNode.next−>TargetNode.next;

**Reverse Operation**

This operation is a thorough one. We need to make the last node to be pointed by the head node and reverse the whole linked list.



**Circular Linked List**

Circular Linked List is a variation of Linked list in which the first element points to the last element and the last element points to the first element. Both Singly Linked List and Doubly Linked List can be made into a circular linked list.

Singly Linked List as Circular

In singly linked list, the next pointer of the last node points to the first node.

**Basic Operations**

Following are the important operations supported by a circular list.

* **insert** − Inserts an element at the start of the list.
* **delete** − Deletes an element from the start of the list.
* **display** −Displays the list.

Ref:

<http://www.portcity.edu.bd/ELibrary/CSE/DatastructurewithC.pdf>

<https://www.includehelp.com/c-programs/c-program-to-calculate-sum-and-product-of-array-elements.aspx>

<https://cboard.cprogramming.com/c-programming/150451-stuck;-beginning-c-programming.html>

<http://www.ece.iastate.edu/~alexs/classes/2007_Fall_207/lab_manual/chapter7_lab.pdf>

<https://www.isid.ac.in/~deepayan/ICP2017/projects/Hriday_Paul/report.pdf>

<http://qu.edu.iq/el/pluginfile.php/69306/mod_resource/content/1/part3.pdf>

<https://github.com/Lintik/hackerrank/tree/master/Languages/C/1D%20Arrays%20in%20C>

<https://www.dotnetperls.com/stack>