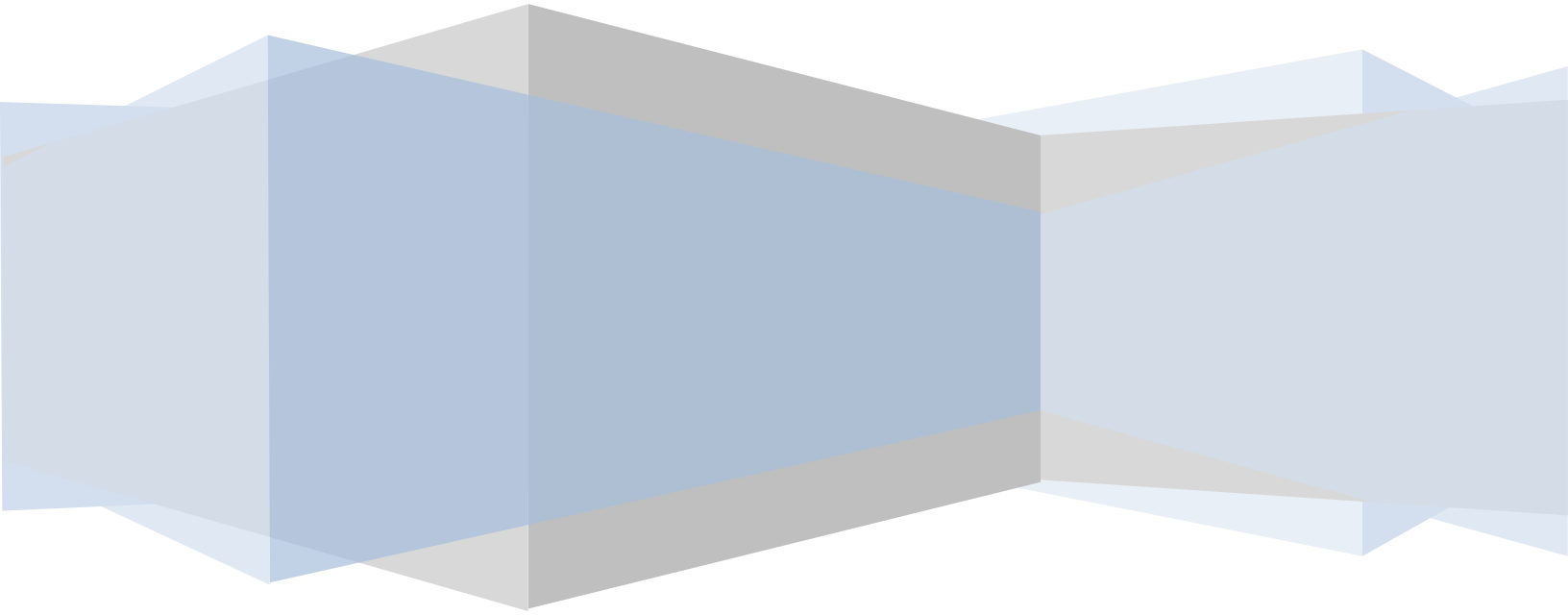
Data Structure

Abu Jakaria Md Numan



Data Structure is a main Thame or main knowladge of C Language. In Data Structure Couse, we know about this topic. Which is bellow :

Before Mid Trem Exam, we know the Topic . Which is :

- About Function
- About Array
    - 1D Array
    - 2D Array
- About Stack
- About Queue
- About Structure
- About Linked List

Now we can Small Describe about this.

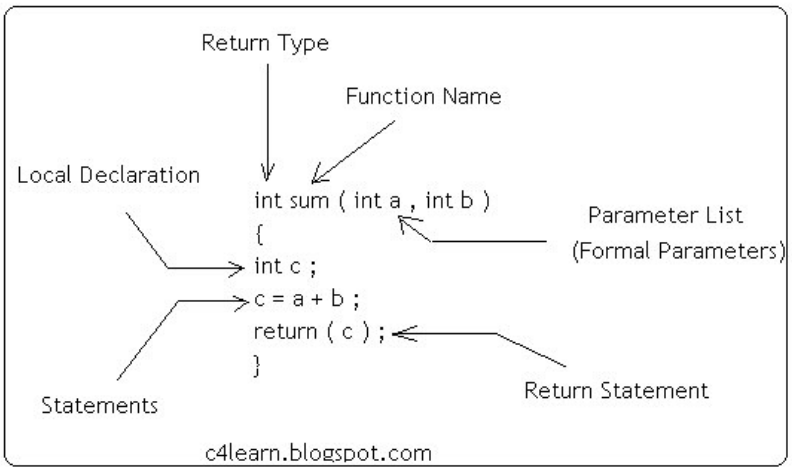
Function

What is function? Everyone say it of his own language.But in C Language, function means that **some group of statement which works when it call.** A **function** declaration tells the compiler about a **function's** name, return type, and parameters. A **function definition** provides the actual body of the **function.**

For Example :

```
Int numanfactory (int product,  int salary)
{
        int multiple = product*salary;
        printf("Given Salary : %d",multiple);
        return multiple;
}
```

Here, **numanfactory** is the function name, before **numanfactory** or function name given **int** which is **function return type** and after the **int product** and **int salary** which is **function parameters.** Show The Bellow images clearly know about function.



Array

Array is the most inportant section in C language. In C language, Array is the collection of Data which is the same type data and also accessing using a common name.Array works on likely dimensional. Array are three type. Which is :

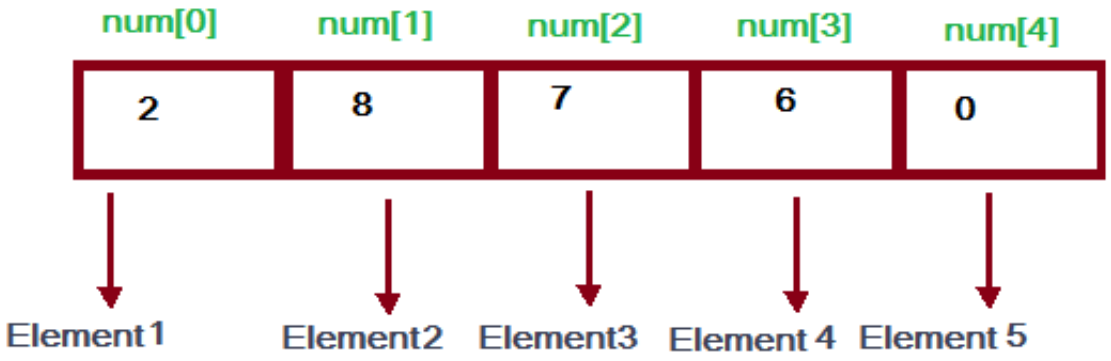
- One dimensional Array.
- Two dimensional Array.
- Dynamic Array or Three dimensional Array.

One dimensional Array Or 1D Array :

One Dimensional Array or 1D Array or Single Array is likely as **a linear array or a list.** Accessing its elements involves a **single** subscript which can either represent a row or column index. Most important think of array is array index start at 0 not 1.

For Example :

```
#include<stdio.h>

int main ()

{

        int num[5];

        num[4] = {2,8,7,6,0};

        return 0;

}
```



Here, **num** is an array and num is the 5 int type value. num[5] value firstly is 2 than 8 than 7 than 6 and lastly 0. As array count firstly 0 so the num index 0 or num[0] =2.Simillary num[1]=8, num[2]=7, num[3]=6, num[4]=0 it is input until of number of array declaration. num[5] means the here the num array has 5 block and this block also a value and num[5] of 5 is declaration of array size.

Two dimensional Array Or 2D Array:

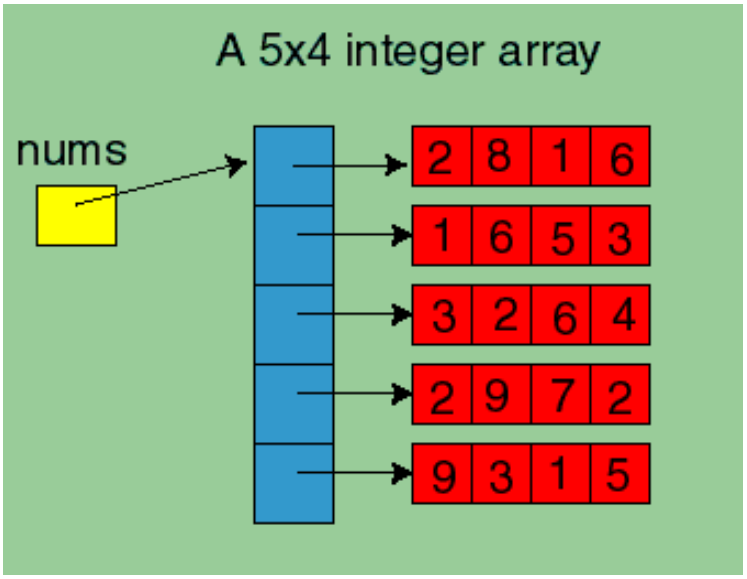
Two Dimensional Array or 2D Array is likely as **a Matrix or a table.** A matrix can be represented as a table of rows and columns. We already know, when we initialize a normal array (or you can say one dimensional array) during declaration, we need not to specify the size of it. However that's not the case with 2D array, you must always specify the second dimension even if you are specifying elements during the declaration.

For Example :

```
#include<stdio.h>

int main()

{

    int A[5][4];
```

```
A[5][4]={   {2,8,1,6},
            {1,6,5,3},
            {3,2,6,4},
            {2,9,7,2},
            {9,3,1,5}
        };

 for(i=0;  i<5;  i++)

 {

        for(j=0;  j<4;  j++)

        {

                printf("%d", A[i][j]);

        }

 }

 return 0;

}
```

Output is Likely This :



Here, A is an array and this array also Two Dimensitional Array because of a declaration. Here A [5][4] declaration means that in A array has 5 row and 4 coloum. This array size is 5*4 = 20 its means than only an array has 20 intager type value.

Stack

A **stack** is an array or list structure of function calls and parameters used in modern computer **programming** and CPU architecture. Elements in a **stack** are added or removed from the top of the **stack**, in a “last in first, first out” or LIFO order.

Stack are working two type of function. Which are :

- push ()
- pop()

push function is working in Stack when Any Element add on Array list. Its means that when add on value on array that time using on push function.

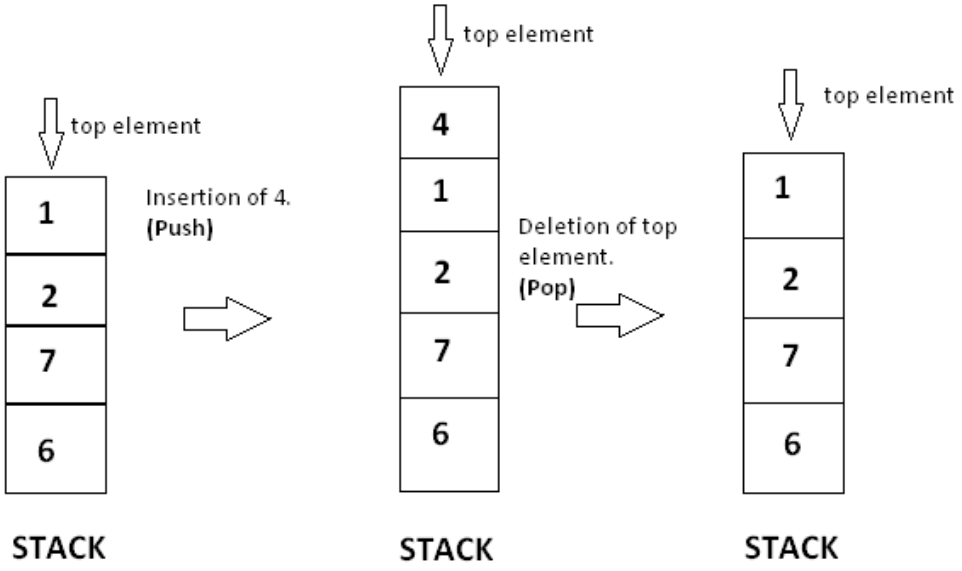
pop function is working when any element on Stack is remove on the list.

For Example :

```
#include<stdio.h>
int head = 0, Stack [30];
void push ( int data )
{
        Stack[head]=data;
        head ++;
}
Void pop ()
{
        int value = Stack[head];
        head --;
}
int main()
{
        push(6);
        push(7);
        push(2);
        push(1);
        push(4);
        pop();
        return 0;
```

}

Output Likely This :

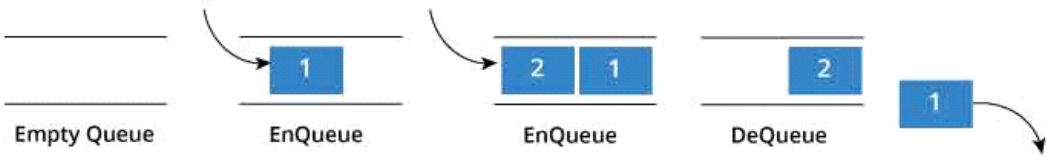


Here we see that firstly 4 elements are push on Stack than push another one which value is 4.so the last value 4 which we add on Stack. So it is the top value on the Stack. Now we pop on the stack and we see that the last value which we add on Stack this value is removing on the Stack. So the Stack list now 6 , 7 , 2 , 1.

Queue

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.



In programming terms, putting an item in the queue is called an "enqueue" and removing an item from the queue is called "dequeue".

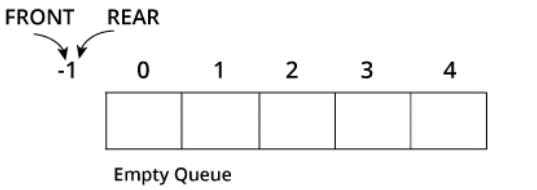
For Example:

```
#include<stdio.h>
#define SIZE 5

void enQueue(int);
void deQueue();
void display();

int items[SIZE], front = -1, rear = -1;


void enQueue ( int value )
{
        if(rear == SIZE-1)
        {
        printf("\nQueue is Full!!");
        }
        else
        {
                if(front == -1)
        front = 0;
                rear++;
                items[rear] = value;
                printf("\nInserted -> %d", value);
        }
}

void deQueue()
{
        if(front == -1)
        {
        printf("\nQueue is Empty!!");
        }
        else
```

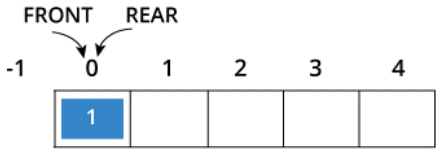
```
        {
        printf("\nDeleted : %d", items[front]);
        front++;
        if(front > rear)
        front = rear = -1;
        }
}

void display(){
        if(rear == -1)
printf("\nQueue is Empty!!!");
        else{
        int i;
        printf("\nQueue elements are:\n");
        for(i=front; i<=rear; i++)
        printf("%d\t",items[i]);
        }
}



int main()
{
        //deQueue is not possible on empty queue
        deQueue();

        //enQueue 5 elements
        enQueue(1);
        enQueue(2);
        enQueue(3);
        enQueue(4);
        enQueue(5);

        //6th element can't be added to queue because queue is
full
        enQueue(6);
```

```
        display();

        //deQueue removes element entered first i.e. 1
        deQueue();

        //Now we have just 4 elements
        display();

        return 0;

}
```

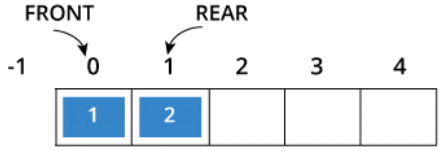
Output is likely that :



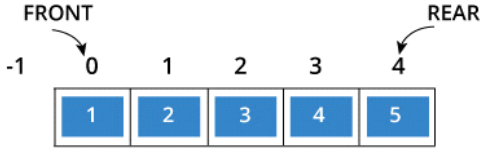
Empty Queue



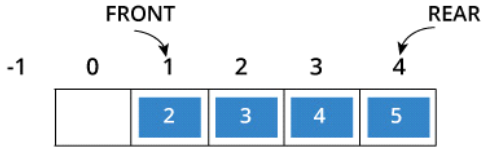
EnQueue first element



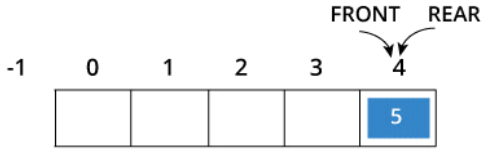
EnQueue



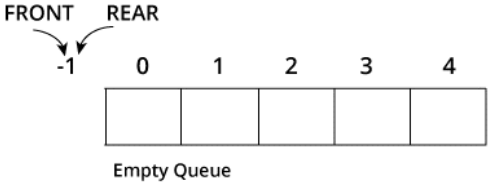
EnQueue



DeQueue



DeQueue last element



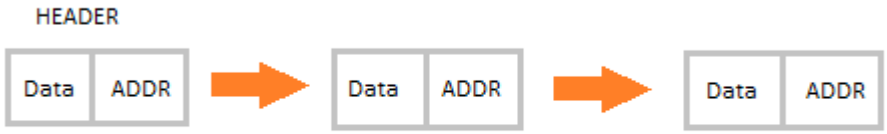
Empty Queue

Linked list

Linked List is a very commonly used linear data structure which consists of group of **nodes** in a sequence.

Each node holds its own **data** and the **address of the next node** hence forming a chain like structure.

Linked Lists are used to create trees and graphs



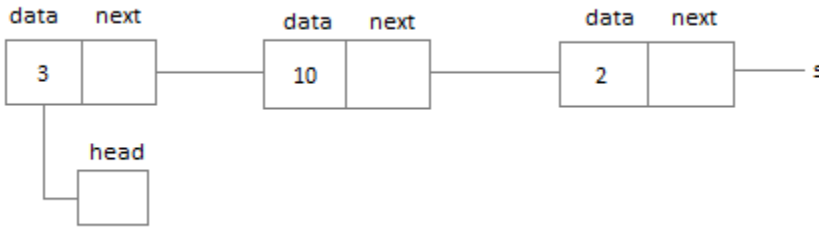
Types of Linked Lists

- Singly Linked List
- Doubly Linked List
- Circular Linked List

Singly Linked List:

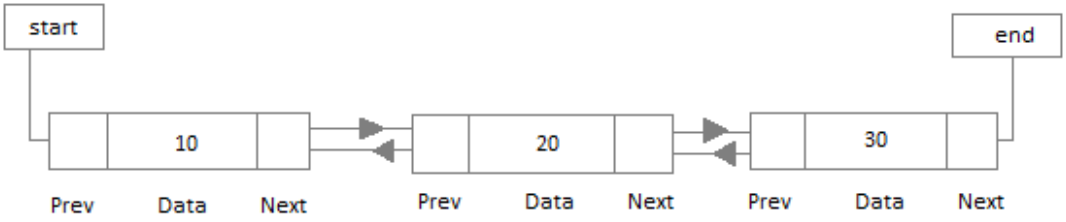
Singly linked lists contain nodes which have a **data** part as well as an **address part next**, which points to the next node in the sequence of nodes.

The operations we can perform on singly linked lists are **insertion**, **deletion and traversal**



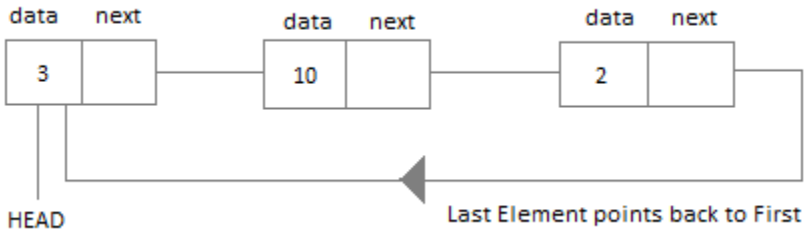
Doubly Linked List:

n a doubly linked list, each node contains a data part and two addresses, one for the **previous node** and one for the **next node**



Circular Linked List:

In circular linked list the last node of the list holds the address of the first node hence forming a circular chain



Structure

Structure is a collection of variables of different types under a single name

example:

I want to store some information about a person: his/her name, citizenship number and salary. I can easily create different variables name, citNo, salary to store these information separately.

in the future, you would want to store information about multiple persons. Now, i need to create different variables for each information per person: name1, citNo1, salary1, name2, citNo2, salary2

And i can easily visualize how big and messy the code would look. Also, since no relation between the variables (information) would exist, it's going to be a daunting task.

A better approach will be to have a collection of all related information under a single name Person, and use it for every person. Now, the code looks much cleaner, readable and efficient as well.

This collection of all related information under a single name Person is a structure.

Syntax of structure:

```
struct structure_name

{

   data_type member1;
```

```
    data_type member2;
    data_type memeber;
};
```

And we can create the structure for a person as mentioned above as:

```
struct person
{
    char name[50];
    int citNo;
    float salary;
};
```

After creating structure we should declare variable like:

```
int main()
{
    struct person person1, person2, person3[20];
    return 0;
}
```