**1. Stack using arrays**

A stack data structure can be implemented using one dimensional array. But stack implemented using array, can store only fixed number of data values. This implementation is very simple, just define a one dimensional array of specific size and insert or delete the values into that array by using **LIFO principle** with the help of a variable **'top'**. Initially top is set to -1. Whenever we want to insert a value into the stack, increment the top value by one and then insert. Whenever we want to delete a value from the stack, then delete the top value and decrement the top value by one.

**Stack Operations using Array**

A stack can be implemented using array as follows...  
  
Before implementing actual operations, first follow the below steps to create an empty stack.

* **Step 1:** Include all the **header files** which are used in the program and define a constant **'SIZE'** with specific value.
* **Step 2:** Declare all the **functions** used in stack implementation.
* **Step 3:** Create a one dimensional array with fixed size (**int stack[SIZE]**)
* **Step 4:** Define a integer variable **'top'** and initialize with **'-1'**. (**int top = -1**)
* **Step 5:** In main method display menu with list of operations and make suitable function calls to perform operation selected by the user on the stack.

**push(value) - Inserting value into the stack**

In a stack, push() is a function used to insert an element into the stack. In a stack, the new element is always inserted at **top** position. Push function takes one integer value as parameter and inserts that value into the stack. We can use the following steps to push an element on to the stack...

* **Step 1:** Check whether **stack** is **FULL**. (**top == SIZE-1**)
* **Step 2:** If it is **FULL**, then display **"Stack is FULL!!! Insertion is not possible!!!"** and terminate the function.
* **Step 3:** If it is **NOT FULL**, then increment **top** value by one (**top++**) and set stack[top] to value (**stack[top] = value**).

**pop() - Delete a value from the Stack**

In a stack, pop() is a function used to delete an element from the stack. In a stack, the element is always deleted from **top** position. Pop function does not take any value as parameter. We can use the following steps to pop an element from the stack...

* **Step 1:** Check whether **stack** is **EMPTY**. (**top == -1**)
* **Step 2:** If it is **EMPTY**, then display **"Stack is EMPTY!!! Deletion is not possible!!!"** and terminate the function.
* **Step 3:** If it is **NOT EMPTY**, then delete **stack[top]** and decrement **top** value by one (**top--**).

**display() - Displays the elements of a Stack**

We can use the following steps to display the elements of a stack...

* **Step 1:** Check whether **stack** is **EMPTY**. (**top == -1**)
* **Step 2:** If it is **EMPTY**, then display **"Stack is EMPTY!!!"** and terminate the function.
* **Step 3:** If it is **NOT EMPTY**, then define a variable '**i**' and initialize with top. Display **stack[i]** value and decrement **i** value by one (**i--**).
* **Step 3:** Repeat above step until **i** value becomes '0'.

**Complete Program in C Programming Language**

The C Program is written for implementation of STACK using Array, the basic operations of stack are PUSH(), POP() and DISPLAY(). PUSH function in the code is used to insert an element to the top of stack, POP function used to remove the element from the top of stack. Finally display function used to print the values at any time. All stack functions are implemented in C Code.

#include<stdio.h>

int stack[100],choice,n,top,x,i;

void push(void);

void pop(void);

void display(void);

int main()

{

//clrscr();

top=-1;

printf("\n Enter the size of STACK[MAX=100]:");

scanf("%d",&n);

printf("\n\t STACK OPERATIONS USING ARRAY");

printf("\n\t--------------------------------");

printf("\n\t 1.PUSH\n\t 2.POP\n\t 3.DISPLAY\n\t 4.EXIT");

do

{ printf("\n Enter the Choice:");

scanf("%d",&choice);

switch(choice)

{ case 1:

{ push();

break;

}

case 2:

{ pop();

break;

}

case 3:

{ display();

break;

}

case 4:

{ printf("\n\t EXIT POINT ");

break;

}

default:

{ printf ("\n\t Please Enter a Valid Choice(1/2/3/4)");

}

}

}

while(choice!=4);

return 0;

}

void push()

{ if(top>=n-1)

{ printf("\n\tSTACK is over flow");

}

else

{

printf(" Enter a value to be pushed:");

scanf("%d",&x);

top++;

stack[top]=x;

}

}

void pop()

{ if(top<=-1)

{

printf("\n\t Stack is under flow");

}

else

{

printf("\n\t The popped elements is %d",stack[top]);

top--;

}

}

void display()

{

if(top>=0)

{

printf("\n The elements in STACK \n");

for(i=top; i>=0; i--)

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

printf("\n Press Next Choice");

}

else

{

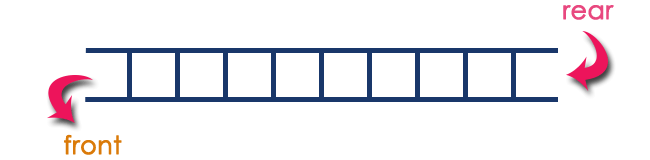
printf("\n The STACK is empty");

}

}

**2. Queue using arrays**

Queue is a linear data structure in which the insertion and deletion operations are performed at two different ends. In a queue data structure, adding and removing of elements are performed at two different positions. The insertion is performed at one end and deletion is performed at other end. In a queue data structure, the insertion operation is performed at a position which is known as 'rear' and the deletion operation is performed at a position which is known as 'front'. In queue data structure, the insertion and deletion operations are performed based on FIFO (First In First Out) principle.



In a queue data structure, the insertion operation is performed using a function called "enQueue()" and deletion operation is performed using a function called "deQueue()".

**Queue data structure can be defined as follows...**

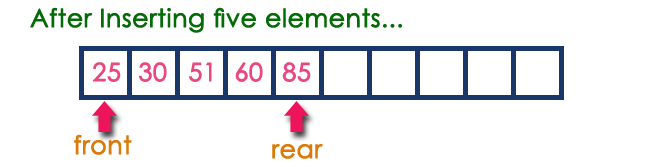
Queue data structure is a linear data structure in which the operations are performed based on FIFO principle.

**A queue can also be defined as**

"Queue data structure is a collection of similar data items in which insertion and deletion operations are performed based on FIFO principle".

**Example**

Queue after inserting 25, 30, 51, 60 and 85.



**Operations on a Queue**

The following operations are performed on a queue data structure...

**enQueue()** - (To insert an element into the queue)

**deQueue()** - (To delete an element from the queue)

**display()** - (To display the elements of the queue)

**Queue data structure can be implemented in two ways. They are as follows...**

Using Array

Using Linked List

When a queue is implemented using array, that queue can organize only limited number of elements. When a queue is implemented using linked list, that queue can organize unlimited number of elements.

**Queue Operations using Array**

Before we implement actual operations, first follow the below steps to create an empty queue.

* **Step 1:** Include all the **header files** which are used in the program and define a constant **'SIZE'** with specific value.
* **Step 2:** Declare all the **user defined functions** which are used in queue implementation.
* **Step 3:** Create a one dimensional array with above defined SIZE (**int queue[SIZE]**)
* **Step 4:** Define two integer variables **'front'** and '**rear**' and initialize both with **'-1'**. (**int front = -1, rear = -1**)
* **Step 5:** Then implement main method by displaying menu of operations list and make suitable function calls to perform operation selected by the user on queue.

**enQueue(value) - Inserting value into the queue**

In a queue data structure, enQueue() is a function used to insert a new element into the queue. In a queue, the new element is always inserted at **rear** position. The enQueue() function takes one integer value as parameter and inserts that value into the queue. We can use the following steps to insert an element into the queue...

* **Step 1:** Check whether **queue** is **FULL**. (**rear == SIZE-1**)
* **Step 2:** If it is **FULL**, then display **"Queue is FULL!!! Insertion is not possible!!!"** and terminate the function.
* **Step 3:** If it is **NOT FULL**, then increment **rear** value by one (**rear++**) and set **queue[rear]** = **value**.

**deQueue() - Deleting a value from the Queue**

In a queue data structure, deQueue() is a function used to delete an element from the queue. In a queue, the element is always deleted from **front** position. The deQueue() function does not take any value as parameter. We can use the following steps to delete an element from the queue...

* **Step 1:** Check whether **queue** is **EMPTY**. (**front == rear**)
* **Step 2:** If it is **EMPTY**, then display **"Queue is EMPTY!!! Deletion is not possible!!!"** and terminate the function.
* **Step 3:** If it is **NOT EMPTY**, then display **queue[front]** as deleted element. Then increment the **front** value by one (**front ++**). Then check whether both **front** and **rear** are equal (**front** == **rear**), if it **TRUE**, then set both **front** and **rear** to '**-1**' (**front** = **rear** = **-1**).

**display() - Displays the elements of a Queue**

We can use the following steps to display the elements of a queue...

* **Step 1:** Check whether **queue** is **EMPTY**. (**front == rear**)
* **Step 2:** If it is **EMPTY**, then display **"Queue is EMPTY!!!"** and terminate the function.
* **Step 3:** If it is **NOT EMPTY**, then define an integer variable '**i**' and set '**i** = **front+1**'.
* **Step 3:** Display '**queue[i]**' value and increment '**i**' value by one (**i++**). Repeat the same until '**i**' value is equal to **rear** (**i** <= **rear**)

#include<stdio.h>

#include<conio.h>

#define SIZE 10

void enQueue();

void deQueue();

void display();

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

void main()

{

int choice;

clrscr();

while(1){

printf("\n\n\*\*\*\*\* MENU \*\*\*\*\*\n");

printf("1. Insertion\n2. Deletion\n3. Display\n4. Exit");

printf("\nEnter your choice: ");

scanf("%d",&choice);

switch(choice){

case 1: enQueue();

break;

case 2: deQueue();

break;

case 3: display();

break;

case 4: exit(0);

default: printf("\nWrong selection!!! Try again!!!");

}

}

}

void enQueue(){

int value;

if(rear == SIZE-1)

printf("\nQueue is Full!!! Insertion is not possible!!!");

else{

if(front == -1)

front = 0;

rear++;

printf("Enter the value to be insert: ");

scanf("%d",&value);

queue[rear] = value;

printf("\nInsertion success!!!");

}

}

void deQueue(){

if(front == rear)

printf("\nQueue is Empty!!! Deletion is not possible!!!");

else{

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

}

}

 OUTPUT:

\*\*\*\*\* MENU \*\*\*\*\*

1.Insertion

2.Deletion

3.Display

4.Exit

Enter the Choice:1

Enter the value to be insert:10

Insertion success!!!

\*\*\*\*\* MENU \*\*\*\*\*

1.Insertion

2.Deletion

3.Display

4.Exit

Enter the Choice:1

Enter the value to be insert: 54

Insertion success!!!

\*\*\*\*\* MENU \*\*\*\*\*

1.Insertion

2.Deletion

3.Display

4.Exit

Enter the Choice:1

Enter the value to be insert: 98

Insertion success!!!

\*\*\*\*\* MENU \*\*\*\*\*

1.Insertion

2.Deletion

3.Display

4.Exit

Enter the Choice:1

Enter the value to be insert: 234

Insertion success!!!

\*\*\*\*\* MENU \*\*\*\*\*

1.Insertion

2.Deletion

3.Display

4.Exit

Enter the Choice:3

Queue Elements are:

10 54 98 234

Enter the Choice:2

Deleted Element is : 10

\*\*\*\*\* MENU \*\*\*\*\*

1.Insertion

2.Deletion

3.Display

4.Exit

Enter the Choice : 3

Queue Elements are:

54 98 234

\*\*\*\*\* MENU \*\*\*\*\*

1.Insertion

2.Deletion

3.Display

4.Exit

Enter the Choice:4