UNITE 2 - Stack and Queue

Stack, Stack operations, Applications of stack - recursion, polish notations - prefix, infix, postfix, Algorithms of stack applications, Introduction to queue, Algorithms and implementation of simple queue, Circular queue, Double ended queue, Priority queue.

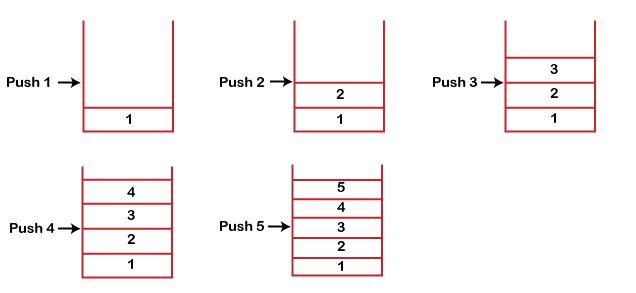
What is a Stack?

A Stack is a linear data structure that follows the LIFO (Last-In-First-Out) principle. Stack has one end, whereas the Queue has two ends (front and rear). It contains only one pointer top pointer pointing to the topmost element of the stack. Whenever an element is added in the stack, it is added on the top of the stack, and the element can be deleted only from the stack. In other words, a stack can be defined as a container in which insertion and deletion can be done from the one end known as the top of the stack.

Working of Stack

Stack works on the LIFO pattern. As we can observe in the below figure there are five memory blocks in the stack; therefore, the size of the stack is 5.

Suppose we want to store the elements in a stack and let's assume that stack is empty. We have taken the stack of size 5 as shown below in which we are pushing the elements one by one until the stack becomes full.



Stack operations

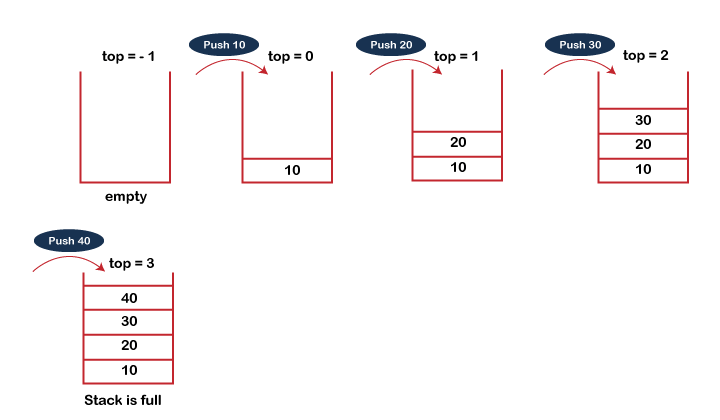
A stack is a linear data structure that follows the Last In First Out (LIFO) principle, meaning the last element inserted is the first one to be removed. The basic operations that can be performed on a stack are:

1. Push: Adds an element to the top of the stack.
2. Pop: Removes the top element from the stack.
3. isEmpty(): It determines whether the stack is empty or not.
4. isFull(): It determines whether the stack is full or not.'
5. peek(): It returns the element at the given position.
6. count(): It returns the total number of elements available in a stack.
7. change(): It changes the element at the given position.
8. display(): It prints all the elements available in the stack

PUSH operation

The steps involved in the PUSH operation is given below:

* Before inserting an element in a stack, we check whether the stack is full.
* If we try to insert the element in a stack, and the stack is full, then the *overflow* condition occurs.
* When we initialize a stack, we set the value of top as -1 to check that the stack is empty.
* When the new element is pushed in a stack, first, the value of the top gets incremented, i.e., top=top+1, and the element will be placed at the new position of the top.
* The elements will be inserted until we reach the *max* size of the stack.



begin

    if top = n then stack full

    top = top + 1

    stack (top) : = item;

end

void push (int val,int n) //n is size of the stack

{

    if (top == n )

    printf("\n Overflow");

    else

    {

    top = top +1;

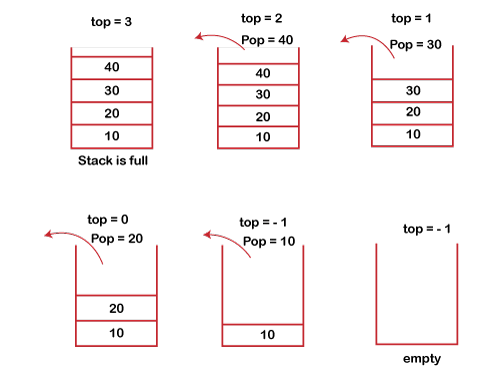
    stack[top] = val;

    }

}

POP operation

The steps involved in the POP operation is given below:

* Before deleting the element from the stack, we check whether the stack is empty.
* If we try to delete the element from the empty stack, then the *underflow* condition occurs.
* If the stack is not empty, we first access the element which is pointed by the *top*
* Once the pop operation is performed, the top is decremented by 1, i.e., top=top-1.

begin

    if top = 0 then stack empty;

    item := stack(top);

    top = top - 1;

end;

int pop ()

{

    if(top == -1)

    {

        printf("Underflow");

        return 0;

    }

    else

    {

        return stack[top - - ];

    }

}

Recursion

Recursion is a fundamental concept in computer science that plays a crucial role in data structures. In the context of data structures, recursion refers to the process of defining a function that calls itself repeatedly until it reaches a base case, which terminates the recursion.

Q1. What is a recursive function?

 A recursive function is a function that calls itself one or more times within its body. A recursive function solves a particular problem by calling a copy of itself and solving smaller subproblems of the original problems.

Q2. What is a base case in recursion?

A base case is a condition in which the recursion can terminate and return the result immediately.

#include <iostream>

using namespace std;

int fibonacci(int n) {

if (n == 0) {

return 0;

} else if (n == 1) {

return 1;

} else {

return fibonacci(n - 1) + fibonacci(n - 2);

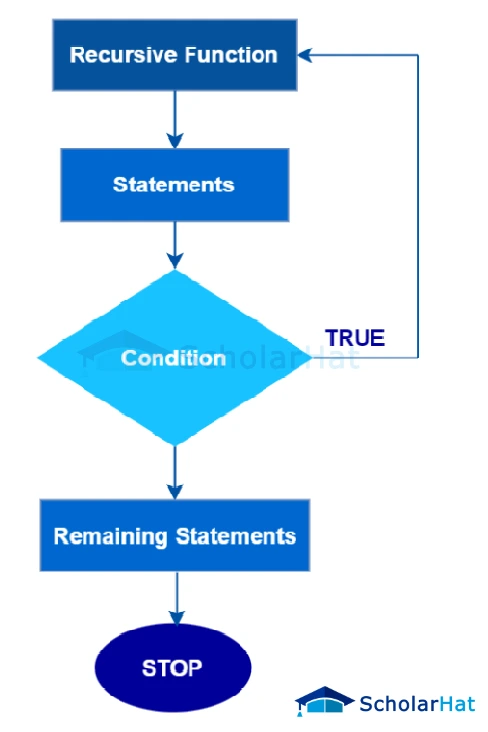
}

}

int main() {

int n, f;

n=12;

 f = fibonacci(n);

cout << f << endl;

return 0;

}

Queue

1. A queue can be defined as an ordered list which enables insert operations to be performed at one end called REAR and delete operations to be performed at another end called FRONT.

2. Queue is referred to be as First In First Out list.

3. For example, people waiting in line for a rail ticket form a queue.



Operations performed on queue

The fundamental operations that can be performed on queue are listed as follows -

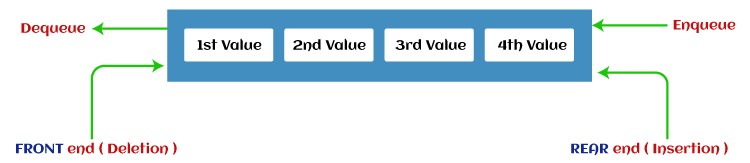
* **Enqueue:** The Enqueue operation is used to insert the element at the rear end of the queue. It returns void.
* **Dequeue:** It performs the deletion from the front-end of the queue. It also returns the element which has been removed from the front-end. It returns an integer value.
* **Peek:** This is the third operation that returns the element, which is pointed by the front pointer in the queue but does not delete it.
* **Queue overflow (isfull):** It shows the overflow condition when the queue is completely full.
* **Queue underflow (isempty):** It shows the underflow condition when the Queue is empty, i.e., no elements are in the Queue.

Applications of Queue

Due to the fact that queue performs actions on first in first out basis which is quite fair for the ordering of actions. There are various applications of queues discussed as below.

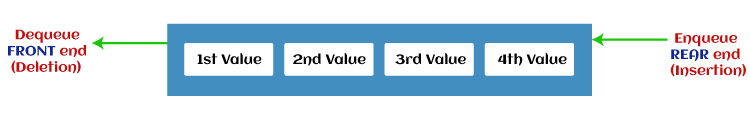
What is a Queue?

Queue is the data structure that is similar to the queue in the real world. A queue is a data structure in which whatever comes first will go out first, and it follows the FIFO (First-In-First-Out) policy. Queue can also be defined as the list or collection in which the insertion is done from one end known as the **rear end** or the **tail** of the queue, whereas the deletion is done from another end known as the **front end** or the **head** of the queue.



Simple Queue or Linear Queue

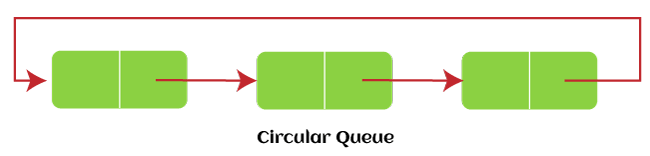
In Linear Queue, an insertion takes place from one end while the deletion occurs from another end. The end at which the insertion takes place is known as the rear end, and the end at which the deletion takes place is known as front end. It strictly follows the FIFO rule.



The major drawback of using a linear Queue is that insertion is done only from the rear end. If the first three elements are deleted from the Queue, we cannot insert more elements even though the space is available in a Linear Queue. In this case, the linear Queue shows the overflow condition as the rear is pointing to the last element of the Queue.

Circular Queue

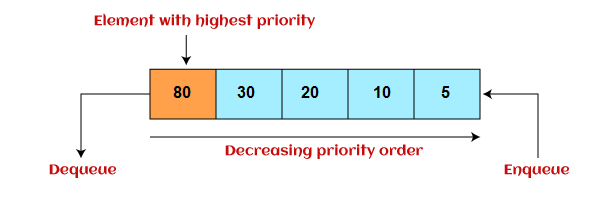
In Circular Queue, all the nodes are represented as circular. It is similar to the linear Queue except that the last element of the queue is connected to the first element. It is also known as Ring Buffer, as all the ends are connected to another end. The representation of circular queue is shown in the below image -



The drawback that occurs in a linear queue is overcome by using the circular queue. If the empty space is available in a circular queue, the new element can be added in an empty space by simply incrementing the value of rear. The main advantage of using the circular queue is better memory utilization.

Priority Queue

It is a special type of queue in which the elements are arranged based on the priority. It is a special type of queue data structure in which every element has a priority associated with it. Suppose some elements occur with the same priority, they will be arranged according to the FIFO principle. The representation of priority queue is shown in the below image -



Insertion in priority queue takes place based on the arrival, while deletion in the priority queue occurs based on the priority. Priority queue is mainly used to implement the CPU scheduling algorithms.

There are two types of priority queue that are discussed as follows -

* **Ascending priority queue -** In ascending priority queue, elements can be inserted in arbitrary order, but only smallest can be deleted first. Suppose an array with elements 7, 5, and 3 in the same order, so, insertion can be done with the same sequence, but the order of deleting the elements is 3, 5, 7.
* **Descending priority queue -** In descending priority queue, elements can be inserted in arbitrary order, but only the largest element can be deleted first. Suppose an array with elements 7, 3, and 5 in the same order, so, insertion can be done with the same sequence, but the order of deleting the elements is 7, 5, 3.

Deque (or, Double Ended Queue)

In Deque or Double Ended Queue, insertion and deletion can be done from both ends of the queue either from the front or rear. It means that we can insert and delete elements from both front and rear ends of the queue. Deque can be used as a palindrome checker means that if we read the string from both ends, then the string would be the same.

Deque can be used both as stack and queue as it allows the insertion and deletion operations on both ends. Deque can be considered as stack because stack follows the LIFO (Last In First Out) principle in which insertion and deletion both can be performed only from one end. And in deque, it is possible to perform both insertion and deletion from one end, and Deque does not follow the FIFO principle.

The representation of the deque is shown in the below image -

