DATA STRUCTURES USING C

NOTES

Prepared by

Prof. Prabhu Kichadi, M. Tech
9880437187

UNIT – III

CONTENTS

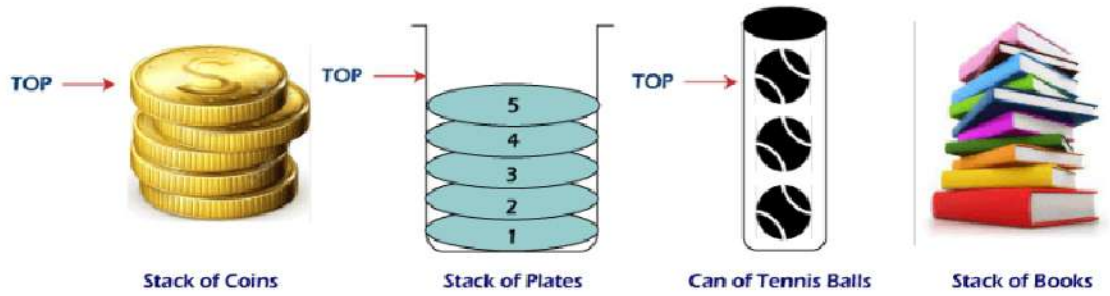
Stacks: Basic concepts – definition and representation of stacks, Operations on stack, push, pop, applications of stack, infix, postfix, prefix notations, Conversion of infix to postfix expressions using stack, Evaluation of postfix expressions using stack, applications of stacks in function calls, **Queues:** basic concepts – definition, and representation of queues, types of queues, simple queues, circular queues, double ended queues, priority queues, operations on simple queues.

Stack:

Definition: *Stack is a linear **LIFO** data structure where element can be added/pushed and deleted/popped from the same end called top of the stack.*

LIFO: Last In First Out – Stack data structure is also called as LIFO, it means the last element inserted is to be removed first.

Some of real world examples:



Representation of Stacks:

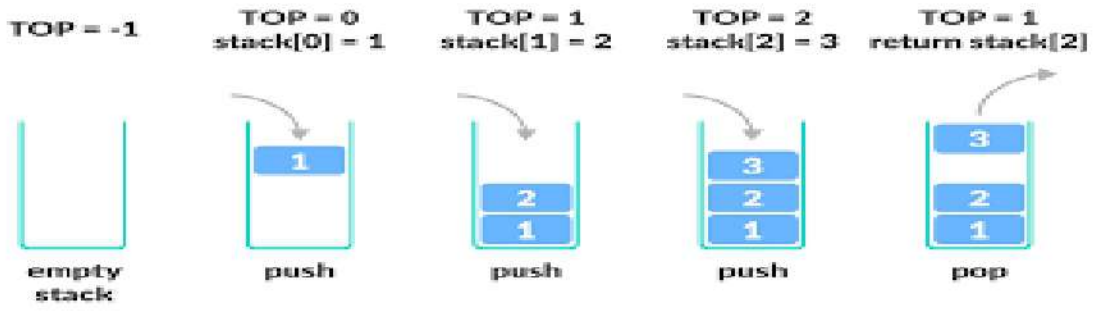
A stack may be represented in the memory in various ways. There are two main ways:

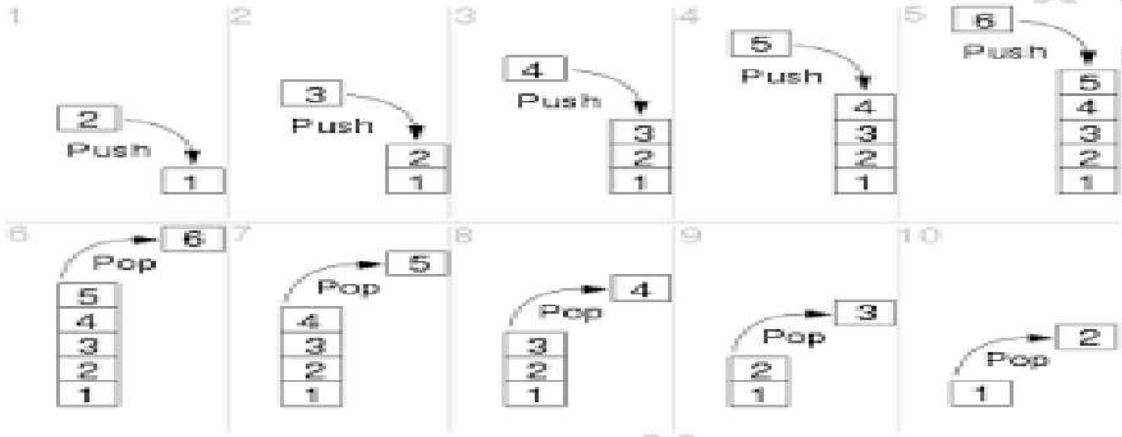
1. **One-Dimensional Array**
2. **A single linked list**

Using Structure and Pointer

A stack can be implemented by means of Array, Structure, Pointer, and Linked List.

Array Representation of Stacks: First we must allocate a memory block of sufficient size to accommodate the full capacity of the stack.





Steps:

- Define **STACKSIZE** for a certain number of elements.
- Create an array having size of STACKSIZE
- Initialize top index as -1 to indicate empty stack.
- While pushing we need to increase top index and while popping, we need to retrieve data from stack and then decrement the top index.
- Design functions for push, pop and display functions to perform operations.

Operations on stack push, pop, and display: For stack implementation and operations, we need to define functions for stack.

1. **Push():** Adding/Inserting an element on the top of the stack is called push operation.
2. **Pop():** Deleting/removing an element from the top of the stack is called pop() operation.
3. **Display():** Printing current elements from the stack is called display() operation.

Status of Stack:

1. **Stack Overflow:** Stack overflow means stack is full of elements we cannot push elements further and that state of the stack is said to be stack overflow.

Condition for Stack Overflow:

```
if(top == STACKSIZE-1)
{
        printf("STACK OVERFLOW");
        return;
}
```

2. **Stack Underflow:** Stack underflow means stack contains no elements. Hence we cannot perform pop() operation in this state.

Condition for Stack Underflow:

```
if(top == -1)
{
        printf("STACK UNDERFLOW");
        return;
}
```

Applications of stack:

- **Evaluation of Arithmetic Expressions**
- **Backtracking**
- **Delimiter Checking**
- **Reverse a Data**
- **Processing Function Calls**

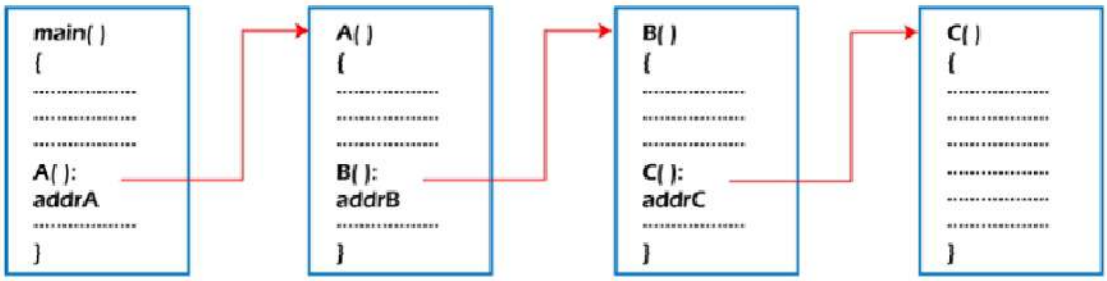
1. **Evaluation of Arithmetic Expressions:** A stack is a very effective data structure for evaluating arithmetic expressions in programming languages. An arithmetic expression consists of operands and operators. To evaluate the expressions, one needs to be aware of the standard precedence rules for arithmetic expression.

2. **Backtracking:** Backtracking is another application of Stack. It is a recursive algorithm that is used for solving the optimization problem.

3. **Delimiter Checking:** The common application of Stack is delimiter checking, i.e., parsing that involves analyzing a source program syntactically. It is also called parenthesis checking. When the compiler translates a source program written in some programming language such as C, C++ to a machine language, it parses the program into multiple individual parts such as variable names, keywords, etc.
4. **Reverse a Data:** A Stack can be used to reverse the characters of a string. This can be achieved by simply pushing one by one each character onto the Stack, which later can be popped from the Stack one by one. Because of the last in first out property of the Stack,

   the first character of the Stack is on the bottom of the Stack and the last character of the String is on the Top of the Stack and after performing the pop operation in the Stack, the Stack returns the String in Reverse order.

5. **Processing Function Calls:** When program contains multiple calls to other functions, the tracing of function calls can be done with the help of stack only. Also used in recursive calls.



Function call

Expressions: *Expression is a valid combination of operators and operands to produce a useful result.*

Expressions are classified as different forms based on usage of data structures.

1. **Infix Notation**
2. **Prefix Notation**
3. **Postfix Notation**

Infix, postfix, prefix notations: There are three notations to represent an arithmetic expression:

- **Infix Notation**: Expression in which operators is placed between the operands is called as Infix Notation expression.

  Ex: (A+B)

- **Prefix Notation:** Expression in which operator is placed before the operands is called as Prefix Notation expression.

  Ex: +AB

- **Postfix Notation:** Expression in which operator is placed after the operands is called as Postfix Notation expression.

  Ex: AB+

Conversion of infix to postfix expressions using stack:

Steps/Algorithm:

1. Start scanning each symbol(operator/operand) from the given expression from left to right and place it into **scanned symbol** column.
2. If the scanned symbol is operand, push it onto the **PostFix Expression** column. If it is operator push it onto the **stack** column.
3. In the next scan, if the symbol is operator, before pushing on **stack** we need to check and confirm operator priority then we must decide for push or pop.
    1. Lower priority operator cannot be pushed on higher priority operator, if it is the case, pop last operator from **stack** and push it onto **postfix expression** column.
    2. Higher priority operator can be pushed on lower priority operator of the **stack**.
    3. No two operators are placed on stack, if the same case pop the top, place it onto postfix and then push on stack.
    4. If closing parathesis occurs pop top operators and push onto stack.

4. Repeat these steps until last symbol and perform until no symbol left on the **stack.**

Operator Priority and Associativity:

Operator	Priority	Associativity
(), {}, []	1	L – R
$, ^	2	R – L
*, /	3	L – R
+, -	4	L – R

Examples:

1. (A+B)

⇨ AB+

Scanned Symbol	Stack	Postfix Expression
(	(	
A	(	A
+	(+	A
B	(+	AB
)	--	AB+

2. (A+B)*(C-D)

⇨ AB+

Scanned Symbol	Stack	Postfix Expression
(	(	
A	(	A
+	(+	A
B	(+	AB
)	--	AB+

3. ((A+(B-C)*D)^E+F)

⇨ ((A+BC- * D)^E+F)

⇨ ((A+ BC-D*)^E+F)
⇨ (ABC-D*+ ^E+F)
⇨ (ABC-D*+E^ +F)
⇨ ABC-D*+E^F+

Scanned Symbol	Stack	Postfix Expression
(	(	
(	((	
A	((	A
+	((+	A
(	((+(	A
B	((+(	AB
-	((+(-	AB
C	((+(-	ABC
)	((+	ABC-
*	((+*	ABC-
D	((+*	ABC-D
)	(	ABC-D*+
^	(^	ABC-D*+
E	(^	ABC-D*+E
+	(+	ABC-D*+E^
F	(+	ABC-D*+E^F
)		ABC-D*+E^F+

4. (A+(B*C)-D)/E

⇨ (A+ BC* -D) / E
⇨ (ABC*+ - D) / E
⇨ ABC*+D- / E
⇨ ABC*+D-E/

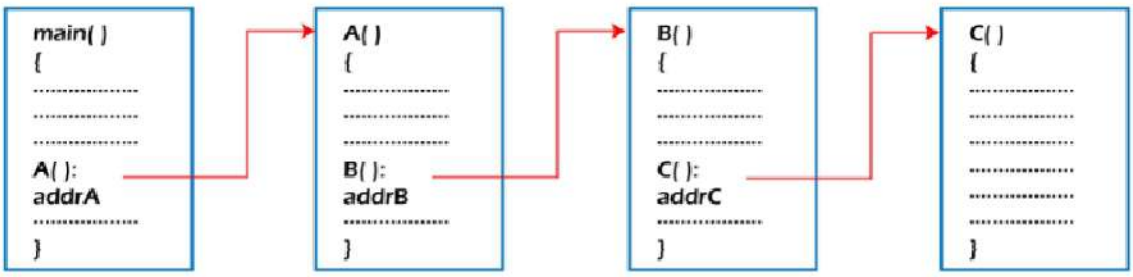
Scanned Symbol	Stack	Postfix Expression
(	(	
A	(	A
+	(+	A
(	(+(	A
B	(+(	AB
*	(+(*	AB

C	(+(*	ABC
)	(+	ABC*
-	(-	ABC*+
D	(-	ABC*+D
)		ABC*+D-
/	/	ABC*+D-
E	/	ABC*+D-E
		ABC*+D-E/

Evaluation of postfix expressions using stack:

Applications of stacks in function calls:

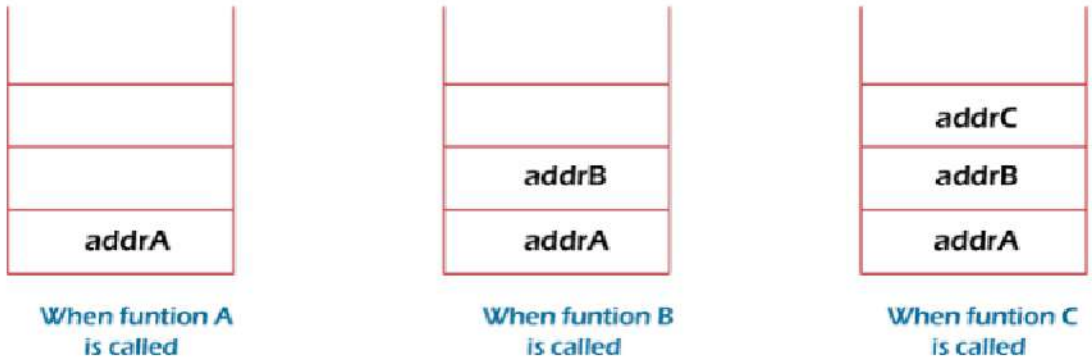
Stack plays an important role in programs that call several functions in succession. Suppose we have a program containing three functions: A, B, and C. function A invokes function B, which invokes the function C.



Function call

When we invoke function A, which contains a call to function B, then its processing will not be completed until function B has completed its execution and returned. Similarly for function B and C. So we observe that function A will only be completed after function B is completed and function B will only be completed after function C is completed. Therefore, function A is first to be started and last to be completed. To conclude, the above function activity matches the last in first out behavior and can easily be handled using Stack.

Consider addrA, addrB, addrC be the addresses of the statements to which control is returned after completing the function A, B, and C, respectively.



Different states of stack

The above figure shows that return addresses appear in the Stack in the reverse order in which the functions were called. After each function is completed, the pop operation is performed, and execution continues at the address removed from the Stack.

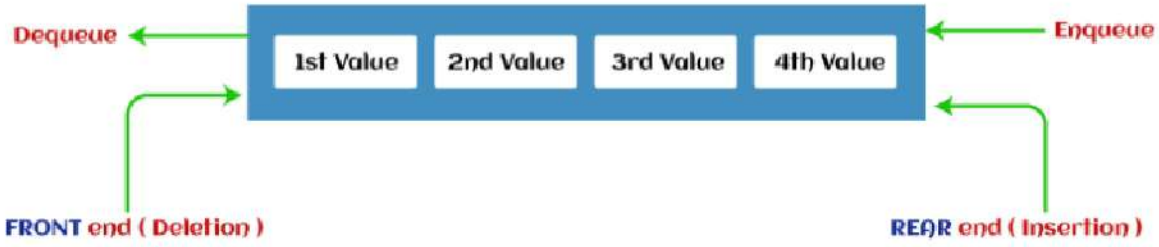
Queues: basic concepts – definition, and representation of queues:

Queue is a linear and FIFO data structure where elements are inserted at one end called rear and deleted from another end called front end.

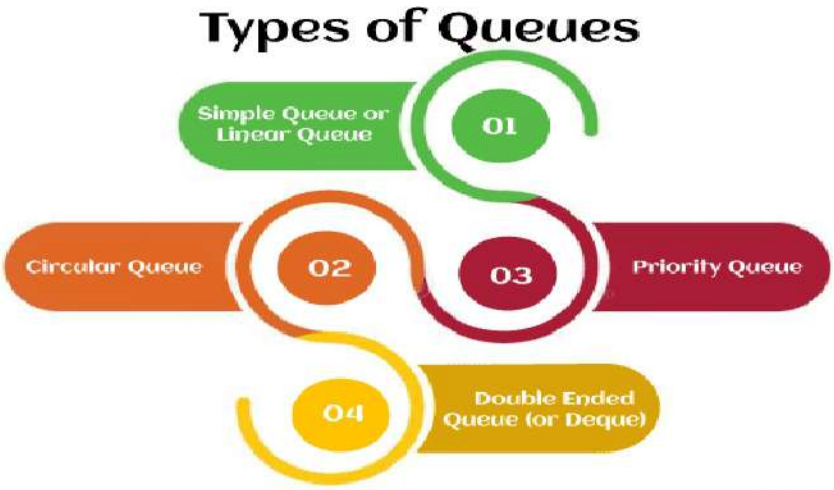
FIFO : FIRST IN FIRST OUT

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.

Representation of Queues:



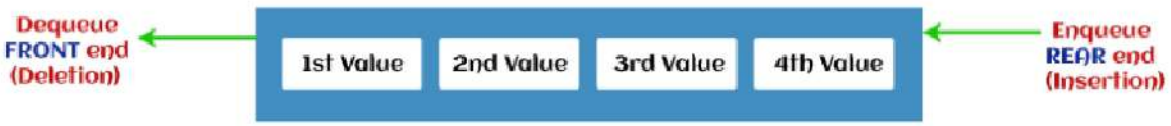
Types of queues:



- Simple Queue or Linear Queue
- Circular Queue
- Priority Queue
- Double Ended Queue (or Deque)

Simple/Ordinary queues: 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 queues:

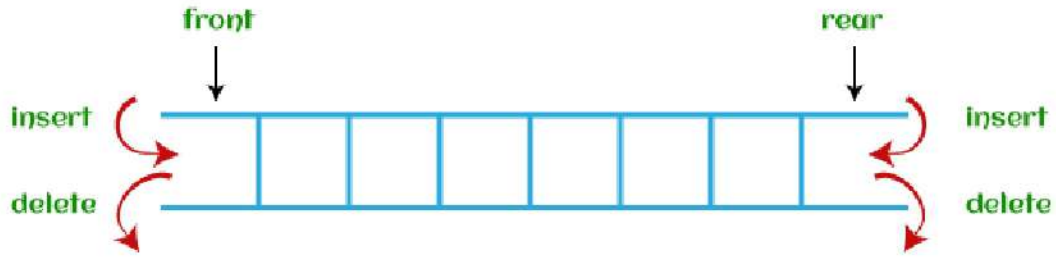
In Circular Queue, all the nodes are represented as circular. It is similar to the linear Queue except that the last element of the queue is connected to the first element. It is also known as Ring Buffer, as all the ends are connected to another end. The representation of circular queue is shown in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

Circular Queue

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.

Double ended queues(Deque): 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.

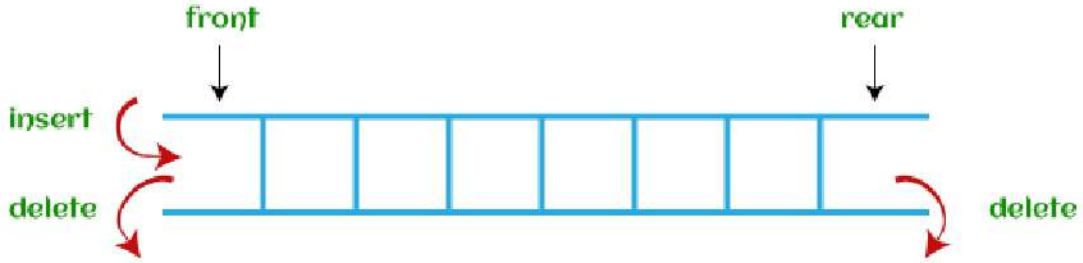
Deque can be used both as stack and queue as it allows the insertion and deletion operations on both ends. Deque can be considered as stack because stack follows the LIFO (Last In First Out) principle in which insertion and deletion both can be performed only from one end. And in deque, it is possible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



double ended queue

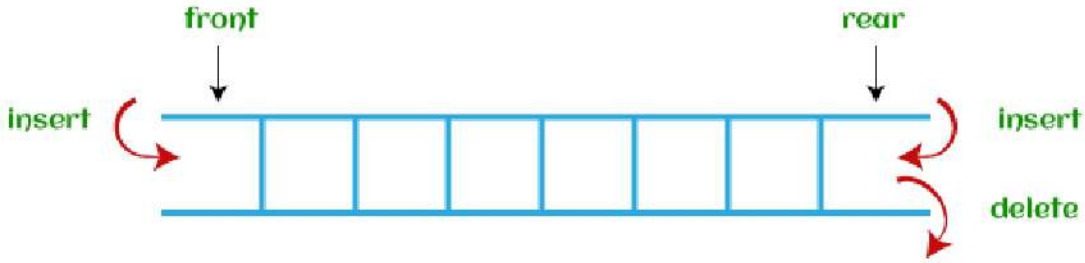
There are two types of deque that are discussed as follows –

1. **Input restricted deque -** As the name implies, in input restricted queue, insertion operation can be performed at only one end, while deletion can be performed from both ends.



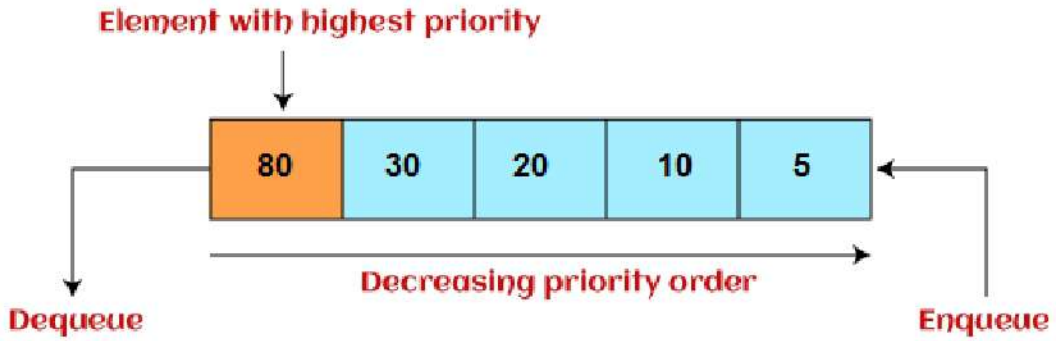
input restricted double ended queue

2. **Output restricted deque** - As the name implies, in output restricted queue, deletion operation can be performed at only one end, while insertion can be performed from both ends.



Output restricted double ended queue

Priority queues: 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.

Operations on simple queues.

- **Enqueue(Insert) :** The Enqueue operation is used to insert the element at the rear end of the queue. It returns void.
- **Dequeue(Delete):** 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.

Ways to implement the queue

There are two ways of implementing the Queue:

- **Implementation using array:** The sequential allocation in a Queue can be implemented using an array.

- **Implementation using Linked list:** The linked list allocation in a Queue can be implemented using a linked list.

Drawbaks of ordinary queue:

- The operations such as insertion and deletion of elements from the middle are time consuming.
- Limited Space.
- In a classical queue, a new element can only be inserted when the existing elements are deleted from the queue.
- Searching an element takes O(N) time.
- Maximum size of a queue must be defined prior.

Applications of Queue:

- **Multi programming:** Multi programming means when multiple programs are running in the main memory. It is essential to organize these multiple programs and these multiple programs are organized as queues.
- **Network:** In a network, a queue is used in devices such as a router or a switch. another application of a queue is a mail queue which is a directory that stores data and controls files for mail messages.
- **Job Scheduling:** The computer has a task to execute a particular number of jobs that are scheduled to be executed one after another. These jobs are assigned to the processor one by one which is organized using a queue.
- **Shared resources:** Queues are used as waiting lists for a single shared resource.

Queue Operations:

1. **Enqueue(Insert):** The Enqueue operation is used to insert the element at the rear end of the queue. It returns void.

```
void insert_rear(int data)
 {
    if(q.r == QUESIZE-1)
       {
          printf("\n\n*******QUEUE OVERFLOW*********");
          return;
```

```
        }
        q.r=q.r+1;
        q.items[q.r]=data;
    }
```

2. Dequeue(Delete): 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.

```
void delete_front()
{
   int data;
   if(q.f>q.r)
   {
      printf("\n\n*******QUEUE UNDERFLOW***********");
      return;
   }
 data=q.items[q.f++];
 printf("\n\nDeleted data : %d",data);
 if(q.f>q.r)
 {
    q.f=0;
    q.r=-1;
 }
}
```

Queue Operations Program (Lab Program):

```
#include<stdio.h>
#include<stdlib.h>
#define QUESIZE 4
typedef struct
{
   int f,r;
   int items[10];
}queue;
```

```
queue q;
void display();
void insert_rear(int data);
void delete_front();
void main()
{
   int choice,option,data;
   q.f=0;
   q.r=-1;

   do
   {
      printf("\n\nEnter your choice : \n");
      printf("\n\n1.Insert\n2.Delete\n3.Display\n4.Exit : ");
      scanf("%d",&choice);

      switch (choice)
      {
         case 1:printf("\n\nEnter data to insert :");
                scanf("%d",&data);
                insert_rear(data);
                break;

         case 2:delete_front();
                break;

         case 3:display();
                break;

         case 4:printf("\n\nExiting from program,press any key");
                getch();
```

```
            exit(1);

        default: printf("\n\nInvalid choice : ");
                 break;
      }

      printf("\n\nWant to continue?press 1/0 :");
      fflush(stdin);
      scanf("%d",&option);

   }while(option);
}

 void insert_rear(int data)
 {
    if(q.r == QUESIZE-1)
       {
          printf("\n\n*******QUEUE OVERFLOW*********");
          return;
       }
       q.r=q.r+1;
       q.items[q.r]=data;
 }
void delete_front()
{
   int data;
   if(q.f>q.r)
   {
      printf("\n\n*******QUEUE UNDERFLOW***********");
      return;
   }
```

```
 data=q.items[q.f++];
 printf("\n\nDeleted data : %d",data);
 if(q.f>q.r)
 {
    q.f=0;
    q.r=-1;
 }
}
void display()
{
      int i;
    if(q.f>q.r)
    {
       printf("\n\n********QUEUE UNDERFLOW******");
       return;
    }
   printf("\n\n*******QUEUE ELEMENTS********\n\n");
   for(i=q.f;i<=q.r; i++)
   {
      printf("%d\t",q.items[i]);
   }
}
```


QUESTION BANK

1. What is LIFO? List applications of it.
2. Define stack. Explain various operations performed on stack. (10Marks).
3. Define stack. List out operations on stack.
4. Convert infix expression to postfix.(5 Marks)
   i. A/B*(C+D/E)-F
   ii. ((A*(B+C))/(D)-F
   iii. (x+y)*(m/n+d)

iv. (A+(B*C)-D)/E

5. Write a note on
   Conversion of expressions using stack. (5Marks).
6. Explain double ended queue. (5Marks)
7. What is queue? Explain types of queues(10Marks)
8. Write a C program to perform all operations on simple/ordinary queue. (10 Marks).
9. What are drawbacks of ordinary queue.
10. What is priority queue? Explain types of it.