UNIT - II

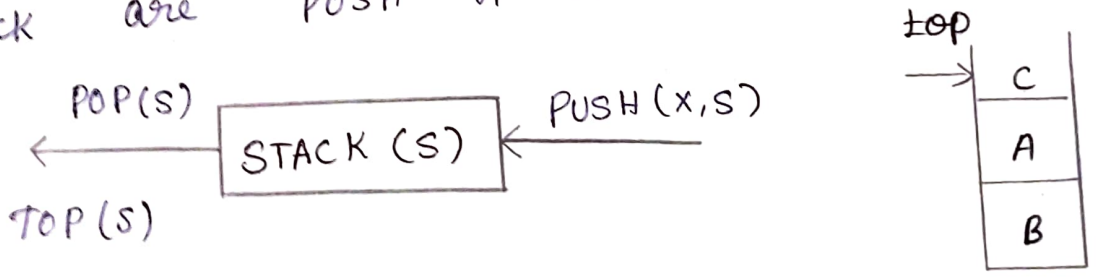
Stacks and queues

STACKS ADT

Stack ADT :-

Stack is an abstract datatype. which is a collection of items inwhich the data are inserted and removed from one end called the top of the stack.

Stack is a list and A stack is a linear data structure. which follows Last In First Out (LIFO) in which both insertion and deletion occur at only one end of the list called the top. The operations performed on stack are PUSH & POP.



OPERATIONS ON STACK :-

The fandamental operation Performed on a stack are Push and Pop.

Push :-

The Process of inserting one element to the top of the stack. For every Push operation the top is incremented by 1.

Pop :-

The Process of deleting an element from the top of stack it called pop operation. After every Pop operation the top pointed is decremented by 1.

overflow :-

Attempt to insert an element when the stack is ~~empty~~ full is said to be overflow.

underflow :-

Attempt to delete an element when the stack is empty is said to be underflow.

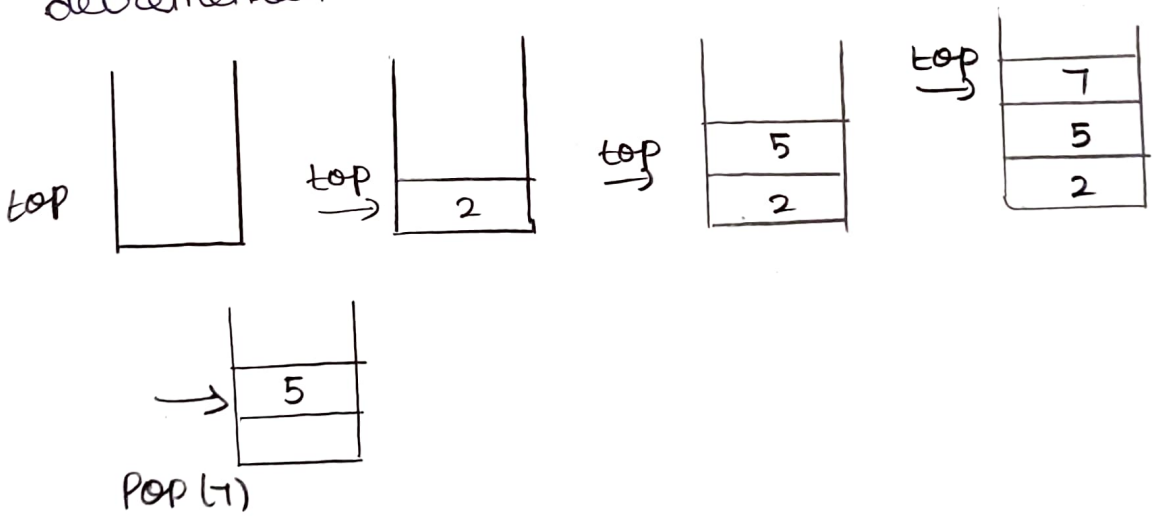
Implementation of Stack :-

1. Array Implementation.
2. linked list Implementation.

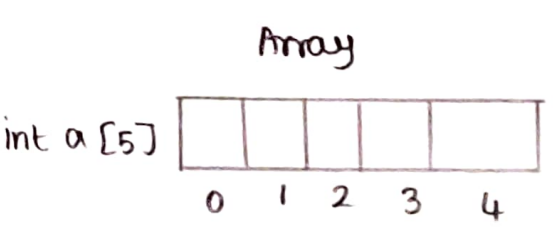
Array Implementation :-

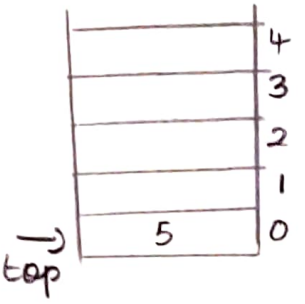
In this implementation each stack is associated with a Pop Pointer which is -1 for an empty stack. To Push an element X on to the stack, TOP pointer is incremented and Set stack[top] = X.

To Pop an element the stack[top] value is returned and the top pointer is decremented.



Implementation of stack using Array :-



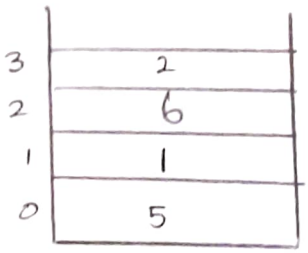


int stack [5]

```
Void main ()
{
   Push ();
   Pop ();
   display ();
}
# define N 5;
int Stack [N];
int top = -1;
void Push ()
{
   Printf (" Enter data ");
   Scanf ("%d", &x);
   if (top == N-1)
   {
      Printf (" overflow ");
   }
   else {
      top ++;
      Stack [top] = x;
   }
}
```

```
Void Pop()
 {
   int item;
   if (top == -1)
   {
     Printf (" underflow ");
   }
   else {
      item = stack [top];
      top--;
      Printf ("%d", item);
    }
  }
```

```
Void display()
 {
   int i;
   for (i = top; i >= 0; i--)
   {
     Printf ("%d", stack [i]);
    }
  }
```



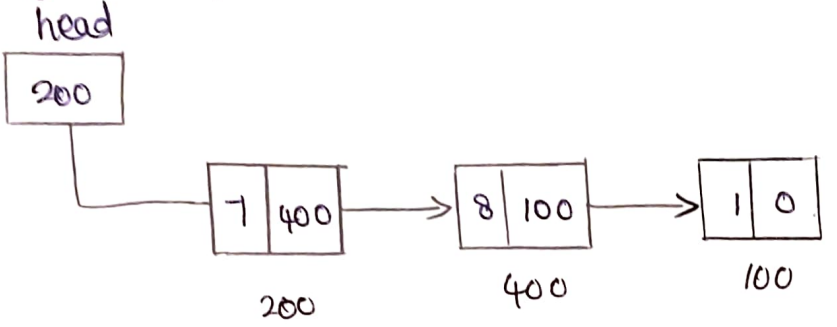
output

2 6 1 5

Implementation of stack using linked list :-

Stack always followed LIFO.

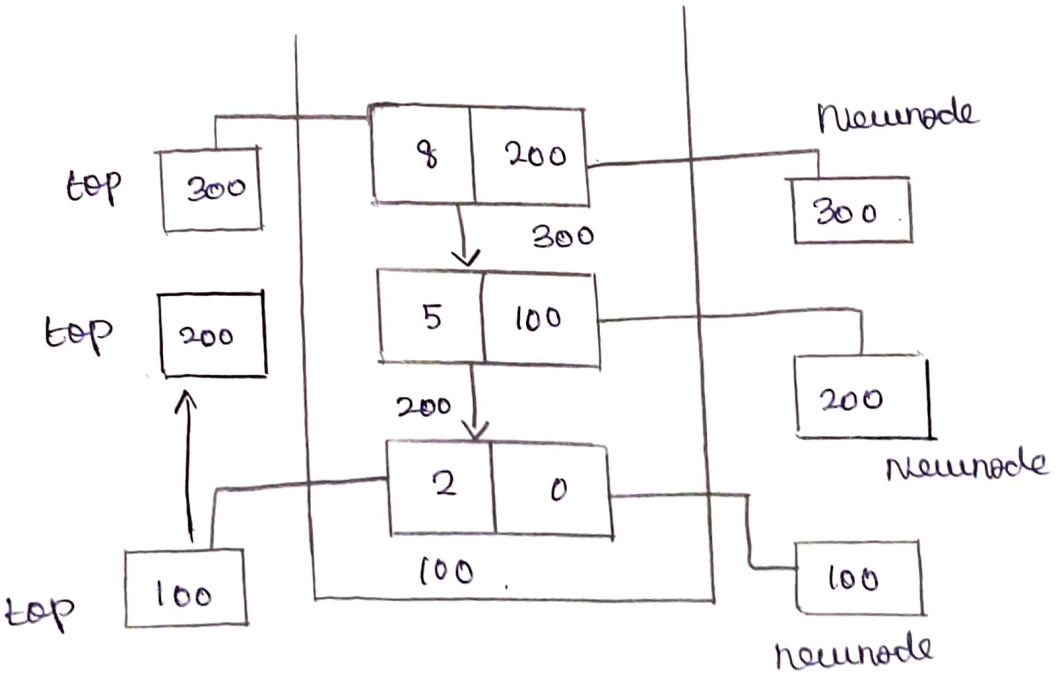
Property. ie) Last In First out.



Push and Pop operation :-

Insert a node and delete a node from the beginning of the list.

becaz stack Property.



Push (2)

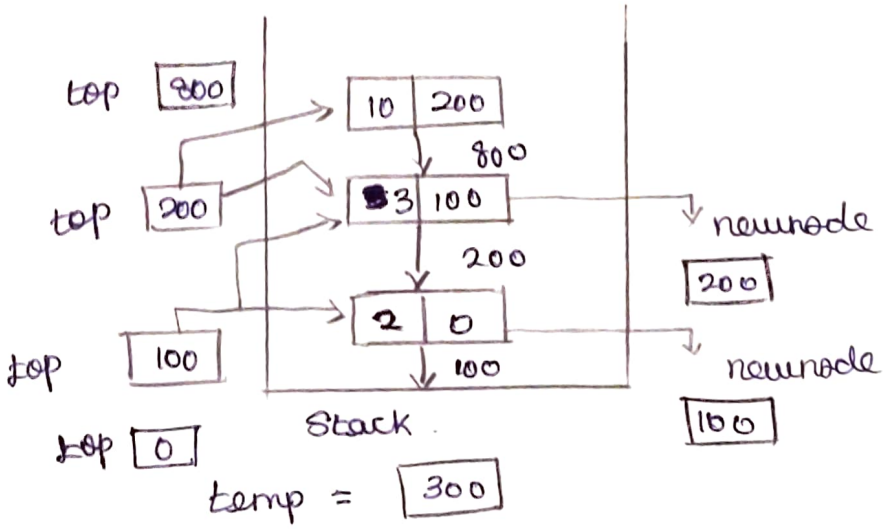
(5)

(8)

```
void main()
{
  Push(2);
  Push(3);
  Push(10);
 display(); 10,3,2
  Peek(); 10
  Pop(); 10
  Peek(); 3
 display(); 3,2
 }
```

declaration

```
Struct node
{
 int data;
 Struct node *next;
 }
void Push(int x)
{
  Struct node * ~~newnode~~ top;
  Struct node * newnode;
  newnode = (Struct node *) malloc (Size of(
                                  Struct node));
   newnode -> data = x;
   newnode -> next = top;
     top = newnode;
   }
```




```
void display()
{
  Struct node * temp;
  temp = top;
  if (top ==0)
  {
    Printf ("underflow");
  }
  else {
    while (temp! =0)
    Printf ("%d", temp -> data);
     temp = temp -> next;
  }
}
```

output

10

3

2

```
Void Peak()
{
  temp = top;
  if (temp == 0)
  {
   Printf (" Stack is empty ");
  }
  else {
     printf (" top element is " top -> data);
    }
 }

Void pop()
{
  Stack node * temp;
  temp = top;
  if (top == 0) {
     Printf (" underflow ") ;
   }
  else {
     Printf ("%d ", top -> data);
     top = top -> next;
     free (temp);
   }
}
```

APPLICATIONS :-

Applications of Stack :-

1. Balancing the symbols
2. Evaluating Arithmetic operations
3. Function calls.
4. Towers of hanoi
5. 8 Queens Problem.

Balancing the symbol :-

BALANCING SYMBOL :-

Compilers check Programs for syntax errors but frequently of Lack of one symbol will cause the compiler to check a hundred lines of diagnostics without identifying the real error.

Thus, in C, right braces, brackets and Parentheses must correspond to their left counter parts

- [()] is legal
- [(]) is wrong

Algorithm of balancing the symbols

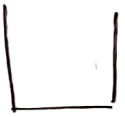
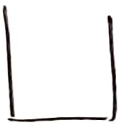
Step 1 : If the character is an opening symbol Push it on to the Stack

Step 2 : If the character is a closing symbol, and if the stack is empty, then report an error as missing opening symbol.

Step 3 : If the character is a closing symbol and if it has a corresponding opening symbol in the stack then Pop it from the stack, otherwise report an error as mismatched symbols.

Step 4 : At the end of file, if the stack is not empty, report an error as missing closing symbol. otherwise report as balanced Symbol.

Consider (a + b) #

Read Character	Stack
(	
a	
+	
b	
)	
#	

Balanced Symbol. Stack is empty.

EVALUTING ARITHMETIC EXPRESENSE :-

The evaluting arithmetic Expresepense can be written in 3 different notations.

AE = A / B + C

3 different ways of represenring the arithmetic expresepense.

1). Infix notation.
2). Postfix notation.
3). Prefix notation.

Infix :-

Arithmetic operator appears between the two operands to which it is being applied

ex A / B + C

Postfix :-

The Arithmetic operator appears directly after the two operands to which it applies also called reverse Polish notation

ex

((A/B) + C) => AB / C +

Prefix :-

The Arithmetic operator is Placed before the two operands to which it applies also called as Polish notation.

ex

$(A/B)+C \Rightarrow +/ABC$

INFIX TO POST FIX CONVERSIONS :-

To evaluate an arithmetic expressions, find convert the given Infix expression to Postfix expressions and then evaluate the Postfix expression using stack.

Infix to Postfix :-

Step 1 : If the character is an operand Place it on to the output

Step 2 : If the character is an operator Push it on to the stack . If the stack operator has a higher or equal Priority than input operator then Pop the operator then the stack and Place it on to the output.

Step 3 : If the character is a left Parenthesis, Push it on to the stack

Step 4 : If the character is right Parenthesis Pop all the operators from the stack till it encounters left Paranthesis discard both the Parenthesis in the output.

Evaluating Postfix Expression :-

Step 1 : If the character is an operand, Push its associated value on to the stack.

Step 2 : If the character is an operator, Pop two values from the stack apply the operator to them and Push the result on to the stack.

A * B #

Read character	Stack	output
A	⌊ ⌋	A
*	| * |	A
B	| * |	AB
#	⌊ ⌋	AB *

Evaluation of AB *

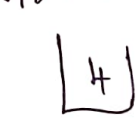
values A = 4 , B = 5

Read character	Stack
A	4
B	5 / 4
*	20

result = 20.

45 + 72 - *

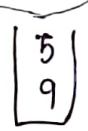
45 + 72 - *



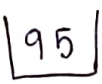
45+72-*

972 - *

95 *



95 *



Result '*'

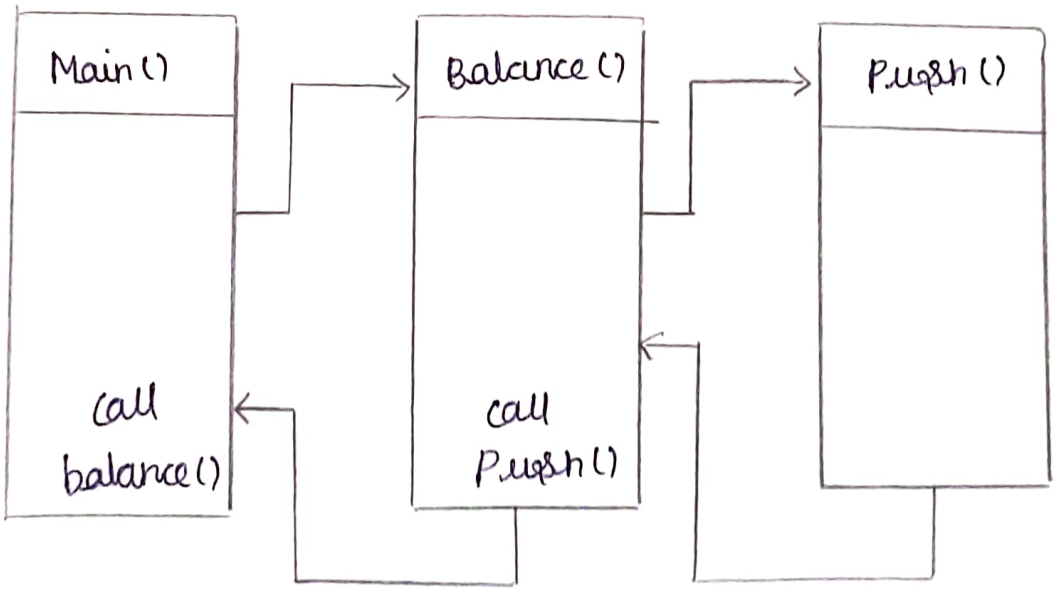
H.W Convert infix to Postfix

a + b * c + (d * e + f) * g

FUNCTION CALLS :.

When a call is made to a new function all the variables local to the calling routine need to be saved otherwise new function will over write the calling routine variables.

Similarly, the current location address in the routine must be saved. Show that the new function knows where to go after it is completed.



All of this work can be done using a stack, and that is exactly in every Programming language that implements recursion. The information saved is called either an activation record or stack frame

Ex : Program factorial

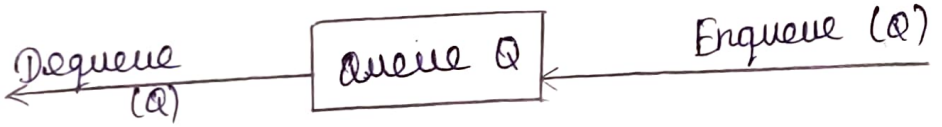
```
int fact (int a)
{
 int s ;
 if (n==1)
  return (1);
  else
  s = n * fact (n-1);
  return (s);
 }
```

QUEUE ADT

queue ADT :-

queues are list, In queue ADT insertion is done at one end where as deletion is performed at the other end

Model of Queue



OPERATIONS :-

The basic operations on queue are Enqueue and Dequeue.

Enqueue :-

The Process of inserting an element at the end of the list (called the rear)

Dequeue :-

The Process of deleting the element at the start of the list (known as front).

Implementation of Queue

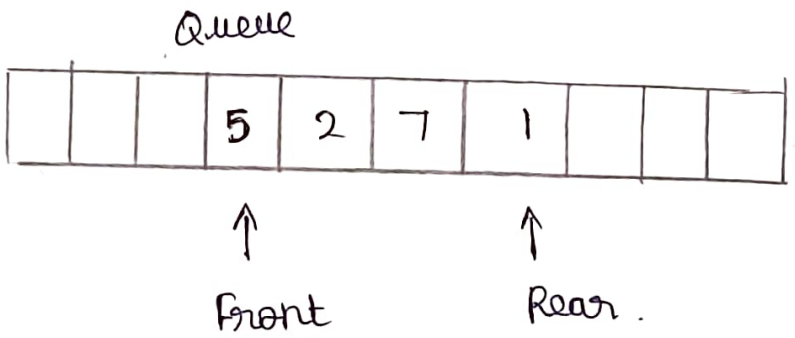
* Using array.
* Using Linked list.

Array Implementation of queue

For each data structure, keep array name as queue [], and the positions Front and Rear, which represent the ends of the queue.

* Number of elements that are actually in the queue is called size.

The following Fig. shows a queue in some intermediate state.

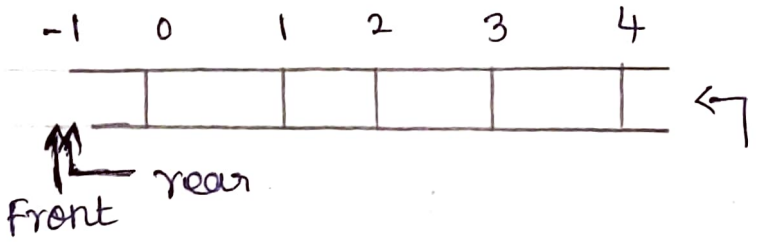


To enqueue an element x, we increment size and Rear.

Then set queue [Rear] = x.

To dequeue an element, -> Return value to queue [Front] decrement size

and then increment Front.



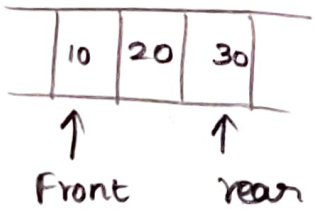
Array means Stack memory locations.

It follows FIFO Rule.

" First In First Out ".

```
void main()
{
  enqueue (2)
  enqueue (3)
  enqueue (10)
  display ();
  Peek ();
  dequeue ();
  Peek ();
  display ();
  }
# define N 5
int queue [N];
int front = -1;
int rear = -1;
```

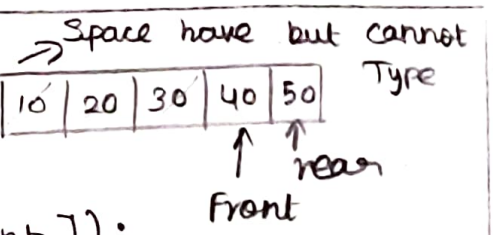
```
void enqueue (int x)
{
  if (rear == N-1)
  {
    Printf (" overflow ");
  }
  else if (front == -1 && rear == -1)
  {
    front = rear = 0;
    queue [rear] = x;
  }
  else {
    rear ++;
    queue [rear] = x;
  }
}
```



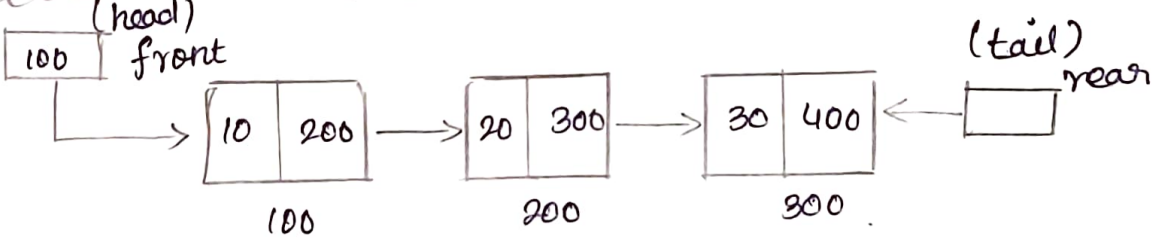

```
Void dequeue ()
{
  if (front == -1 && rear = -1)
  {
    Printf (" underflow");
  }
  elseif (Front == rear)
  {
    Front = rear = -1;
```

```
    }
  else
  {
    Printf ("%d dequeue element " Queue [Front]);
     front ++;
    }
  }
Void display ()
 {
  if (front == -1 && rear == -1)
  {
    Printf (" Queue is empty ");
    }
  else
  {
    for (i = front ; i < rear ++;  i++)
     {
       Printf ("%d", queue [i]);
      }
    }
  }
Void Peek ()
{
  if (rear == -1 && Front == -1)
  {
    Printf (" underflow ");
```

```
    }
  else
  {
    printf ("%d", queue [Front]);
  }
}
```



Implementation of Queue using linked list

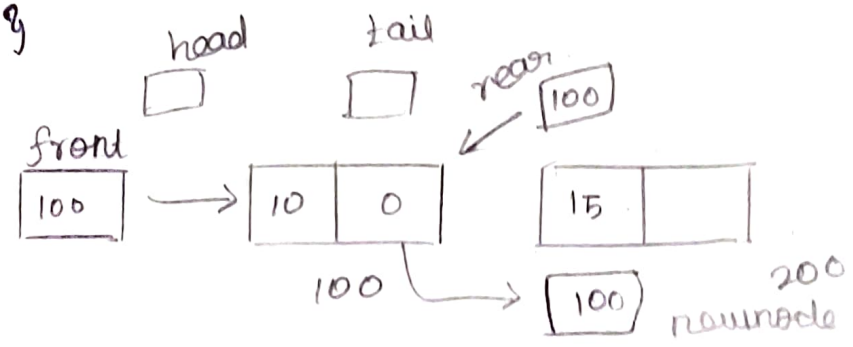



```
void main()
{
  enqueue (10);
  enqueue (12);
  enqueue (15);
  display ();
  dequeue();
  Peek ();
}
```

Routine to insert a node

```
Struct node
  {
    int data;
    Struct node * next;
  }
```

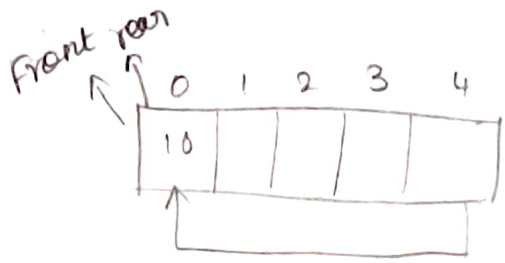
```
void enqueue (int x)
{
  Struct node * front = 0;
  struct node * rear = 0;
  Struct node * newnode;
  newnode = (struct node *) malloc (size of
                                    (struct node));
  newnode -> data = x;
  newnode -> next = 0;
  if (front == 0 && rear == 0)
  {
    front = rear = newnode;
  }
else
  {
   rear -> next = newnode;
   rear = newnode;
  }
}
```



```
void display()
{
  struct node * temp;
  if (Front == 0 && rear == 0)
  {
    Printf (" Queue is empty ");
    }
  else
  {
    temp = front;
    while (temp != 0)
    {
      Printf ("%d", temp -> data);
      temp = temp -> next;
    }
  }
}
void dequeue()
{
  struct node * temp;
  temp = front;
  if (front == 0 && rear == 0)
  {
    Printf (" underflow ");
  }
```

```
else
{
   Printf ("%d", front -> data);
   front = front -> next;
   free (temp);
  }
}
Void  Peek()
{
  if ( front == 0 && rear == 0)
   {
     Printf ("underflow");
    }
   else
   {
     Prinatf ("%d", front -> data);
    }
   }
```

```
# define N, 5
int queue [N];
int front = -1;
int rear = -1;
void enqueue (int x)
{
  if (front == -1 && rear == -1)
  {
    front = rear = 0;
    queue [rear] = x;
  }
else if ((rear+1) % N == front)
  {
    Printf ("queue is full");
  }
else
  {
    rear = (rear+1) % N;
    queue [rear] = x;
  }
}
```

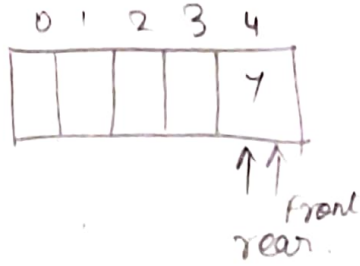


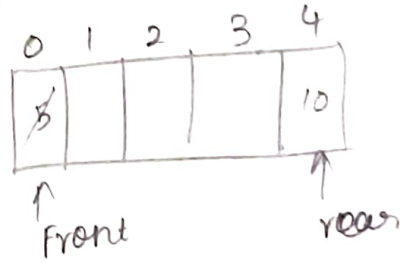
rear = 0

0 + 1 = 1 % 5

rear = 1

```
void dequeue()
{
  if (front == -1 && rear == -1)
  {
   Printf ("underflow");
  }
else if (front == rear)
{
  front = rear = -1;
 }
else
  {
    Printf ("%d", queue [front]);
    front = (front +1) %N;
  }
}
```

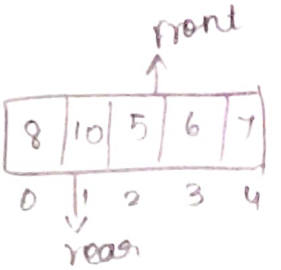




0 + 1 % 5 = 1

```
void display()
{
  int i = front;
  if (front == -1 && rear == -1)
  {
   Printf ("queue is empty");
  }
else
```

```
{
Printf ("The queue element is queue:");
while (i != rear)
{
  printf ("%d", queue[i]);
  i = (i+1) % N;
}
Printf ("%d", queue[rear]);
}
}
```



2! = 1

(2+1) % 5 = 3

(3+1) % 5 = 4

5 % 5 = 0

O/P

5, 6, 7, 8, 10

DEQ :- DEQUE

DOUBLE ENDED QUEUE

A double ended queue (DEQUE) is a special type of queue that allows insertion and deletion of elements in both ends.

Types

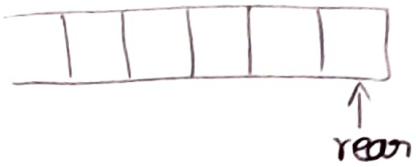
1. Input restricted deque - It allows deletion from both the ends but restrict the insertion at only one end.

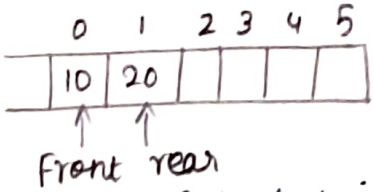
2. output restricted deque - It allows insertion at both ends but restrict the deletion at only one end.

Enqueue (insertion) at rear position

```
void DEQUE_ enqueue - rear (int x)
{
  if (((front == rear+1) || ((front == 0) && (rear ==
                                    max - 1)))
    Printf ("Queue is full");
  else
    {
      if (rear == max - 1)
```

```
        rear = 0;
    else
        rear = rear + 1;
        DEQUE [rear] = x;
    }
}
```

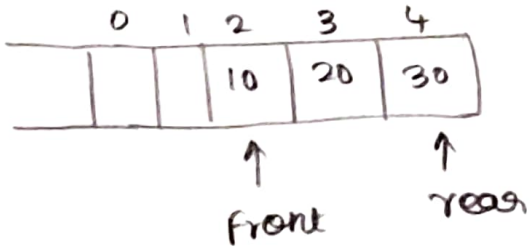




It is used in ~~Multiple~~ ~~Queue~~ task scheduling for multiple Processors

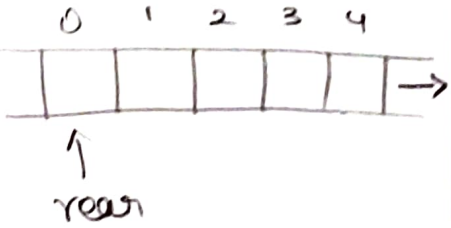
Enqueue(insertion) at front Position

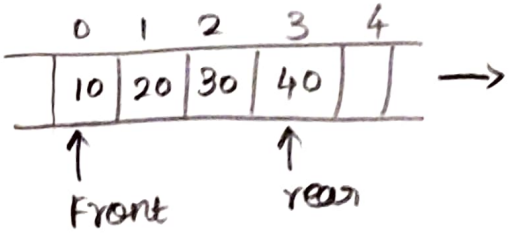
```
void DEQUE - enqueue - front (int x)
{
  if (((front == rear+1) || ((front == 0) &&
                              (rear == max-1)))
     Printf ("Queue is full");
  else
    {
      if (front == 0)
          front = max-1;
      else {
          front = front - 1;
          DEQUE [front] = x;
          }
    }
}
```



Dequeue (deletion) at front Position

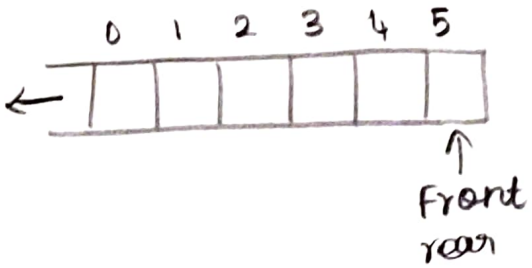
```
int DEQUE_ dequeue_ front ()
{
  if ( front == -1)
     Printf ("QUEUE empty ");
  else
    if (front == max -1)
       front =0 ;
    else
      {
       x = DEQUE [front];
       front = front + 1;
       return x;
      }
}
```

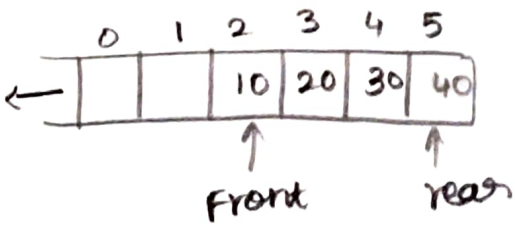




Dequeue (deletion) at rear Position

```
int DEQUE_ dequeue_ rear ()
{
   if ( front == -1 )
   Printf ("QUEUE empty");
   else
```





```
    {
        x = DEQUE [rear];
        rear = rear - 1;
        return x;
      }
    }
}
```

APPLICATIONS OF QUEUE

Applications of queue :

Priority queue :- implement Priority queues Priority queue is a Priority Queue in which inserting an item or removing an item can be Performed from any position based on some Priority

Need for PQ (Priority queue) :-

In operating System. Jobs are initially Placed at the end of the queue. The scheduler will repeatedly take the first job on the queue, run it until either it finishes or its time limit is up and Place it at the end of the queue. if it does not finish

This strategy is not appropriate because very short jobs to take a long time because of the wait involved running.

This Problem can be solved by Priority queue

SJF (Shortest Job First)

Computer networks :-

* Computer networks where the Server takes the job of the client as Per the queue Starategy.

Simulation :-

* When Jobs are Submitted to the Printer they are arranged in order of arrival.

* Batch Processsing in an OS

Example Email of using S/w (software)

Running S/w (software)

It is a method of running software Programs called jobs in batches automatically.

* Queuing theory :-

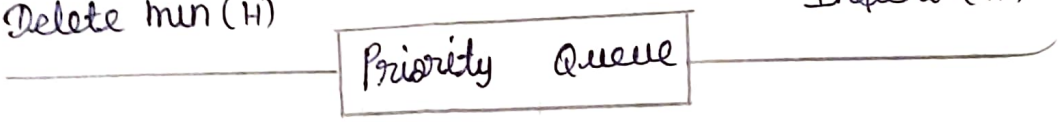
↳ how long users expect to wait on a line

↳ The answer depends on arrive

↳ how frequently users arrive.

↳ how long it takes to Process

Node model of a Priority queue :-



Two basic operations performed by Priority queue are insertion and deletion.

Insertion operation is used to insert an element in to the Priority queue, it is similar to enqueue operation. Deletion operation is used to delete minimum element from the Priority queue. It is similar to dequeue operation.

Simple Implementation :-

There are several ways of implement a Priority queue.

- Linked list
- Binary search tree
- Binary heap

Queue applications can be summarized as follows.

* To implement a Priority queue.
* Priority queue can be used to Sort the element using heap Sort.
* Simulation.
* Batch Processing in an operating System.
* compter networks where the server takes job of the client as per the queue strategy.
* Mathematics uses queuing theory.