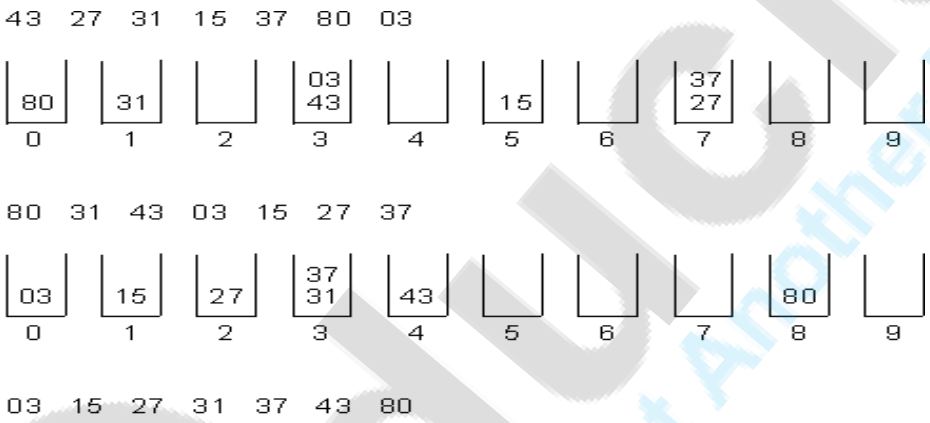
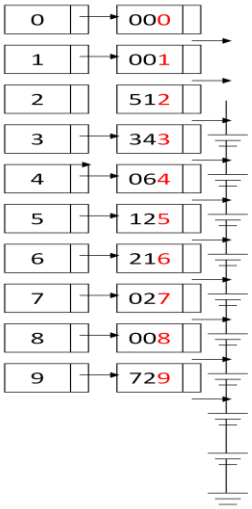
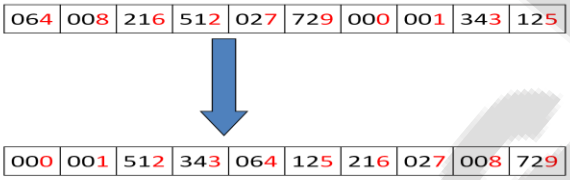
Radix Sort

- Radix sort is a sorting algorithm that sorts integers by processing individual digits.
- Two classifications of radix sorts
  - Least significant digit process the integer representations starting from the least significant digit and move towards the most significant digit.
  - Most significant digit process the integer representations starting from the most significant digit and move towards the least significant digit. This is also known as radix exchange sort
- The steps in Least significant digit (LSD) radix sort algorithm are as follows:
  - Take the least significant digit of each key.
  - Sort the list of elements based on that digit.
  - Repeat the sort with the immediate more significant digit.

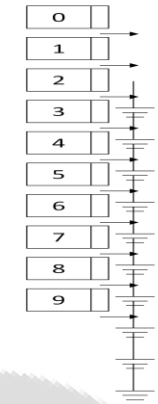


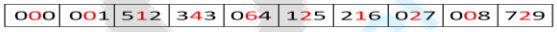
Pass 1

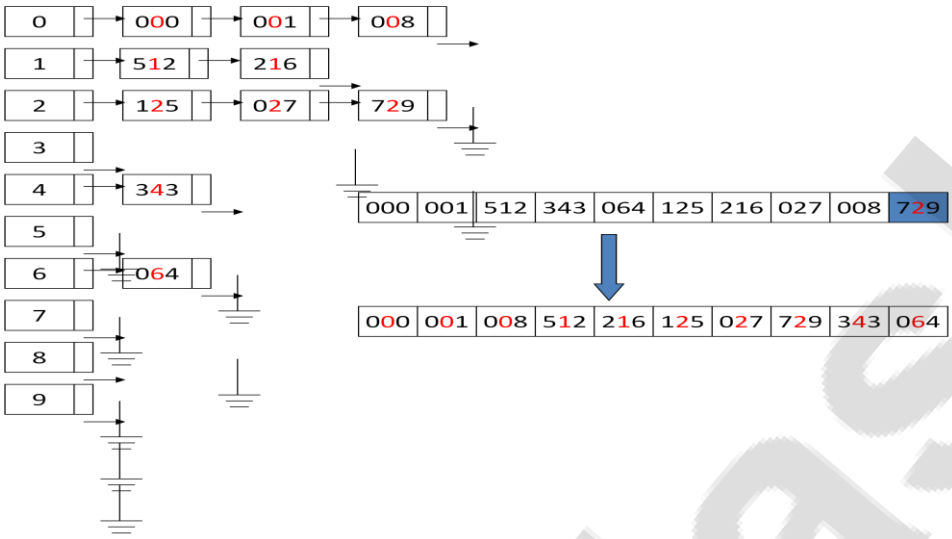




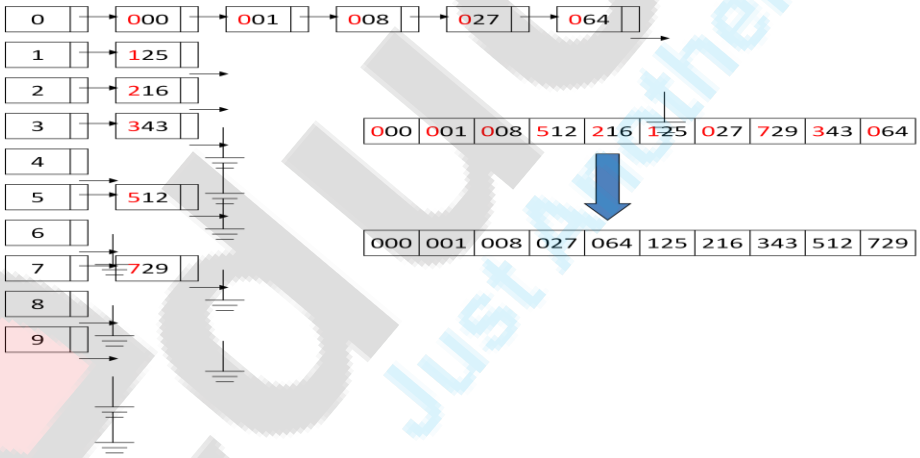
Pass 2






0
1
2
3
4
5
6
7
8
9
000
001
008
512
216
125
027
729
343
064
000 001 512 343 064 125 216 027 008 729
000 001 008 512 216 125 027 729 343 064

PASS 3


0
1
2
3
4
5
6
7
8
9
000
001
008
027
064
125
216
343
512
729
000 001 008 512 216 125 027 729 343 064
000 001 008 027 064 125 216 343 512 729

RADIX SORT

Algorithm radix (a, length)
// a is array to be sorted, length is number of elements in array
Pre: Unsorted list of length n.
Post: Sorted list in ascending order of length n

1. for k = lsd to msd do // k = no. of digits in data
    1. for i = 0 to (n-1) do
        1. y = a[i]
        2. j = k^{th} digit of y
        3. place y at rear of queue[j]
    2. for q = 0 to 9 do
        1. place elements of queue[q] in next sequential position of a

Reference: Tenenbaum , Forouzan (old)

Quick sort:

- Quicksort sorts by employing a divide and conquer strategy to divide a list into two sub-lists.
- The steps are:
    - Pick an element, called a pivot, from the list.
    - Reorder the list so that all elements which are less than the pivot come before the pivot and all elements greater than the pivot come after it (equal values can go either way). After this partitioning, the pivot is in its final position. This is called the partition operation.
    - Recursively sort the sub-list of lesser elements and the sub-list of greater elements.

44 33 11 55 77 90 40 60 99 22 88 66

↑
pivot

After partitioning

40 33 11 22 44 90 77 60 99 55 88 66

Less than 44 greater than 44

Quick sort – partition algorithm

Step 1: Repeatedly increase the pointer down by one position until a[down] > pivot

Step 2: Repeatedly decrease the pointer up by one position until a[up] <= pivot.

Step 3: if down < up, interchange a[down] and a[up]

Steps 1,2 ,3 are repeated until step 3 fails.

i.e. if up <= down, interchange pivot and a[up]

algorithm:

int partition (a, beg, end)

// Places pivot element piv at its proper position; elements

before it are less than it & after it are greater than it

1. piv = a[beg]
2. up = end
3. down = beg
4. while (down < up)
    1. while((a[down] <= piv) && (down<end))
        1. down=down + 1
    2. while(a[up]>piv)
        1. up=up-1
    3. if (down<up)
        1. swap (a[down], a[up])
5. swap(a[beg], a[up])
6. return up

Algorithm sort (a, beg, end)

// a - array to be sorted, beg - starting index of array to be sorted, end - ending index of array to be sorted

Pre: Unsorted list a of length n.

Post: Sorted list in ascending order of length n

1. if (beg < end)
    1. j = partition(a, beg, end)
    2. sort(a, beg, j-1)
    3. sort (a, j+1, end)
2. else
    1. return

Complexity of quick sort algorithm:

- Assume that array size n is power of 2 Let $n = 2^m$, so that $m = \log_2 n$
- Assume that proper position for the pivot always turn out to be middle of array.
- During first pass there will be n comparisons.Array is split into two subarrays.

  For each of the subarrays n/2 comparisons are required.
- In next pass total 4 files are created each of size n/4

  Each file require n/4 comparisons yielding n/8 files.
- After m pass, there will be n files each of size 1.
- Total number of comparisons =

  $= n + 2(n/2) + 4 (n/4) + 8 (n/8)+\ldots\ldots$

  $= n + n + n + n+\ldots.+n$ (m times)

  $= n.m$

  $= n \log_2 n$
- Complexity of quicksort algorithm = $O(n \log_2 n)$

Q. Sort array using quick sort.65, 21, 14, 97, 87, 78, 74, 76, 45, 84, 22

Solution:

Stack

Q. Define stack.show the working of stack with suitable example . give algorithm for push,pop,isfull,isempty,count,display,peep change operation.

- An stack is an ordered collection of items into which new items may inserted and item may be deleted at one end called TOP of the stack.
- A stack is a homogeneous collection of items of any one type.
- Data can be added or removed from only the top.
- Last In, First Out (LIFO).

Stack Operations:

1. **Push :**Place an item onto the stack. If there is no place for new item, stack is in overflow state.
2. **Pop :**Return the item at the top of the stack and then remove it. If pop is called when stack is empty, it is in an underflow state.
3. **Top:** Access the item at the top of the stack, but do not remove it.
4. **isfull:** Tells if the stack is full or not.
5. **isEmpty** : Tells if the stack is empty or not.
6. **Count:** The number of items in the stack.
7. **Peep:** Access the item at the i^{th} position

8. **Change:**change the item at i^{th} posititon
9. **Display:**print the element from stack

Push():

- Algorithm :push(S, TOP, X): This algorithm insert element x to the top of the stack which is represented by array S containing N elements with pointer TOP denoting the top most element in the stack.

1. [check for stack overflow]
   if TOP = =N-1
   write[stack overflow]
   return
2. [Increment TOP]
   TOP = TOP+1
3. [Insert element]
   S[TOP]=X
4. [finished]

Pop():

- Algorithm :pop(S, TOP): This algorithm remove top most element from top of the stack which is represented by array S containing N elements with pointer TOP denoting the top most element in the stack.

1. [check for stack underflow]
   if TOP = -1
   write[stack underflow on POP]
   return
2. [Decrement TOP Pointer]
   TOP = TOP-1
3. [return top element from stack]
   return(S[TOP+1]

Peep();

- Algorithm :Top(S, TOP): This algorithm returns the value of i^{th} element of the stack which is represented by array S containing N elements with pointer TOP denoting the top most element in the stack. The element is not deleted by this function.

1. [check for stack underflow]
   if TOP-i+1 < 0
   write[stack underflow on TOP]
   return
2. [return i^{th} element from stack]
   return(S[TOP-i+1]

Top():

- **Algorithm :Top(S, TOP): This algorithm returns the value of top most element from stack. Stack S containing N elements** with pointer TOP denoting the top most element in the stack. The element is not deleted by this function.

1. [check for stack underflow]
   if TOP < 0
   write[stack underflow on TOP]
   return
2. [return top most element from stack]
   return(S[TOP]

change():

- **Algorithm :change(S, TOP,X i):** This algorithm change the value of ith element from the top of the stack with x . stack S containing N elements with pointer TOP denoting the top most element in the stack.

1. [check for stack underflow]
   if TOP-i+1 < 0
   write[stack underflow on change]
   return
2. [change the return i^{th} element from top of the stack]
   return(S[TOP-i+1] = X
3. [finish]

return

isfull():

- **Algorithm :isfull(S, TOP, N):** This algorithm check whether stack is full . stack S containing N elements with pointer TOP denoting the top most element in the stack.

1. [check for stack overflow]
   if TOP = =N-1
   write[stack full]
   return

isempty():

- **Algorithm :isempty l(S, TOP, N):** This algorithm check whether stack is full . stack S containing N elements with pointer TOP denoting the top most element in the stack.

1. [check for stack overflow]
   if TOP ==-1
   write[stack empty]
   return

count():

- **Algorithm :count(S, TOP):** This algorithm count number of elements present in stack which is represented by array S containing N elements with pointer TOP denoting the top most element in the stack.

1. [check for empty stack]
   if TOP==-1
   count=0
   return count
2. For i=0 to top
   Count=count+1
3. Return count

Display():

- **Algorithm :display(S, TOP):** This algorithm display elementsof stack which is represented by array S containing N elements with pointer TOP denoting the top most element in the stack.

2. [check for empty stack]
   if TOP==-1
   write("underflow");
3. For i=0 to top
   Write (s[i])

Applications of Satck

Q. write an algorithm to convert Infix to post fix

Infix to postfix conversion rules:

- While no error occurs and end of infix expression has not been reached, do:

a) Get next input *Token* (constant, variable, arithmetic operator, left and right parenthesis) from the infix expression.
b) If *Token* is

- A left parenthesis: Push onto stack
- A right parenthesis: Pop and display stack elements until a left parenthesis is popped, but do not display left parenthesis.
- An operator: If stack is empty or *Token* has higher priority than stack top, push *Token* onto stack. Else, repeatedly pop and display stack top which has same or higher precedence than *Token* and at the end push token on stack. Left parenthesis has lower priority than any operators.
- An operand: Display it.

c) On end of infix expression, pop and display stack item until stack is empty.

Q. Covert following infix expression into postfix expression:

A*B +(C+(D-E))

TOKEN	Stack	Display
A		A
*	*	A
B	*	A B
+	+	A B *
(	+ (	A B *
C	+ (	A B * C
+	+ (+	A B * C
(	+ (+ (	A B * C
D	+ (+ (	A B * C D
-	+ (+ (-	A B * C D
E	+ (+ (-	A B * C D E
)	+ (+	A B * C D E -
)	+	A B * C D E - +
End		A B * C D E - + +

Q. Change following infix expressions to postfix expressions using stack:

1. D-B+C
2. A*B+C*D
3. (A+B)*C-D*E+F
4. (A-2*(B+C)-D*E)*F
5. A^B*C-D+E/F/(G+H)
6. ((A+B)*C)/(D+E)
7. (A+B^C^D)*(E+F/D)
8. ((A+B)*C)-(D-E)^(F+G)
9. (P+Q*(R/S)-T)*(U/(V+W-Z))
10. ((A+B*(C/D)-E)*(F/(G+H-J))

Q.**Evaluation of postfix expression:**

- Create an empty stack that will contain operands.
- Take one by one token from the left to right.

- If token is an operand push it onto the stack.
- If token is an operator op
  - Pop the top item from the stack as operand2.
  - Pop again the top item from the stack as operand1.
  - Perform operation operand1 op operand2.
  - Push result back to stack.

- When all tokens in input expression are processed stack should contain a single item, which is the value of expression.

Evaluate 7 5 - 4 *

Token	Action Taken	Stack Content After Processing the Token
7	Push 7	7
5	Push 5	7 5
-	Pop 5 as second operand, pop 7 as first operand, 7-5 is 2. Push 2.	2
4	Push 4	2 4
*	Pop 4 as second operand, pop 2 as first operand, 2*4 is 8. Push 8.	8

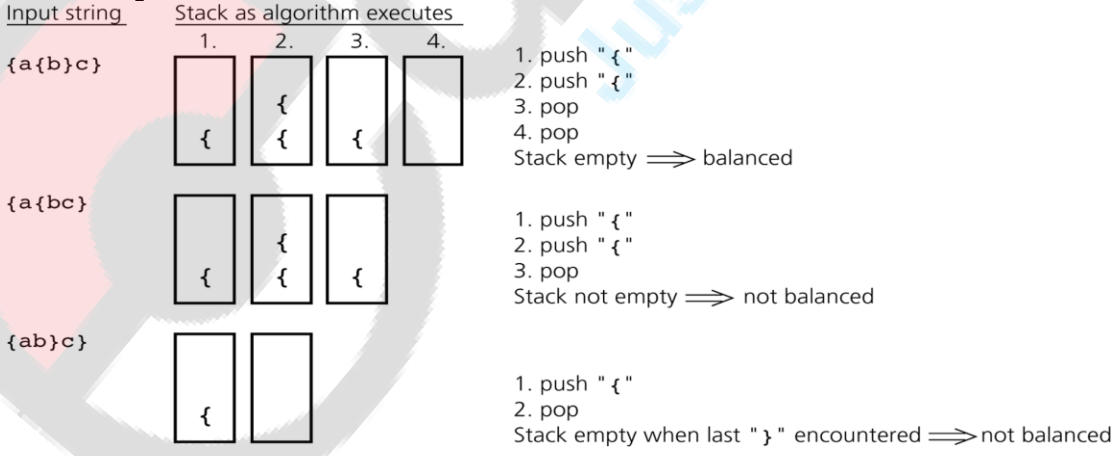
Q. Determine value of following postfix expressions when A =2, B=3, C=4, D=5, E=6, F=7

- AB*C-D+
- ABC+*D-
- AB+C*DE*-F+

Q. Write a short note on **Balancing of parenthesis?**

- A stack can be used to verify whether a program contains balanced braces
  - An example of balanced braces
    abc{defg{ijk}{l{mn}}op}qr
  - An example of unbalanced braces
    abc{def}}{ghij{kl}m

Example:

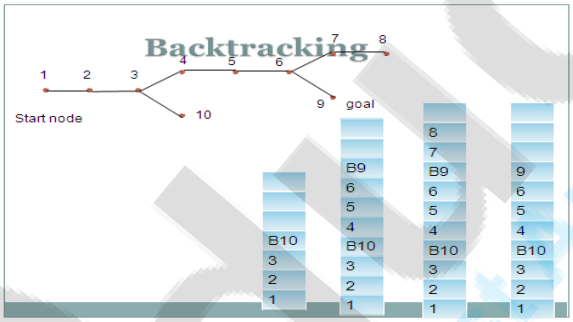


- **Algorithm for Balancing of parenthesis**

1) Declare a character stack S.
2) Now traverse the expression string exp.
a) If the current character is a starting bracket (‘(‘ or ‘{‘ or ‘[‘) then push it to stack.
b) If the current character is a closing bracket (‘)’ or ‘}’ or ‘]’) then pop from stack and if the popped character is the matching starting bracket then fine else parenthesis are not balanced.
3) After complete traversal, if there is some starting bracket left in stack then “not balanced”

Q. application of stack:

- Expression evaluation
  - Convert infix expression into postfix form
  - Evaluate postfix expression
- Reversing data
  - Read series of numbers and pushes them on stack.
  - Pop the stack and prints number in reverse order.
- Backtracking
  - Making decision between two or more paths.



Enter branch point into stack if network contain more than one path.

- Match the braces in a source program

- Recursion:
  In recursive function,
  - Save parameters, local variables, and return address on stack.
  - If the base criteria(termination condition) has been reached, perform computation and go to step 3. Otherwise perform partial computation and go to step 1.
  - Restore most recently saved parameters, local variables, and return address from stack. Go to this return address.

// stack using linked list

```
#include<conio.h>
#include<iostream.h>
#include<process.h>
class stack
{
        int info, ele;
        stack *node,*link,*top;
public:
        stack()
        {
                top=NULL;
        }
        void insert();
        void del();
        void dis();
};

void stack::insert()
{
        node=new stack;
        cout<<"\nEnter Info:";
        cin>>ele;
        node->info=ele;
        node->link=NULL;
        if(top==NULL)
        {
                top=node;
        }
        else
        {
                node->link=top;
                top=node;
        }
}
void stack::del()
{
        if(top==NULL)
        {
                cout<<"\n Underflow";
        }
        else
        {
```

```
                cout<<"\nDeleted Element is :"<<top->info;
                top=top->link;
        }
}

void stack::dis()
{
        stack *move;
        move=top;
        while(move!=NULL)
        {
                cout<<"\t"<<move->info;
                move=move->link;
        }
}

void main()
{
        clrscr();
        int ch;
        stack s;
        cout<<"\n1.Insert 2.Show 3.Delete 4.Exit";
        while(ch!=4)
        {
                cout<<"\nEnter Choice";
                cin>>ch;
                switch(ch)
                {
                        case 1: s.insert(); break;
                        case 2: s.dis(); break;
                        case 3: s.del(); break;
                        case 4:exit(0);
                }
        }
getch();
}
```

Queue

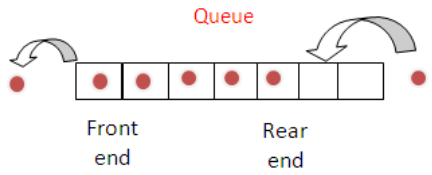
Q. Define Queue. Write algorithm for enqueue,Dequeue,isfull,isempty,display,queueu rear,queue front , Queue count operation.

Or

Explain queue data structure with suitable example? Write an algorithm for Enqueue , Dequeue ,Queuefront and queuerear.

Ordered homogeneous group of items in which the items are added at one end (rear) and are removed from the other end (front).

First In, First Out (FIFO)



Insertion in queue:

- QINSERT (Q,F,R,N,Y) : Given F and R , pointers to the front and rear elements of queue , a queue Q consisting of N elements and y is element which is inserted by this procedure at rear of queue . Initially F and R are set to -1.

1. [checking for overflow]

   If (R>=N-1)

   then write (" overflow")

   return

2. [Increment rear pointer]

   R←R+1

3. [Insert element]

   Q[R] ← Y

4. [Is front pointer properly set]

   if F=-1

   Then F← 0

   Return

Deletion from queue:

- QDELETE (Q,F,R) : Given F and R , pointers to the front and rear elements of queue , This procedure delete element at front of the queue . Y is temporary variable .

1. [checking for underflow]

   If (F==-1)

   then write (" underflow")

   return

2. [Delete element]

   Y← Q[F]

3. [Queue empty]

   if F=R

   Then F← R ← -1

   Else F← F + 1 (increment front pointer)

4. [Return element]

   Return(Y)

Queue front:

- QFRONT (Q,F) : Given F pointers to the front of queue , This procedure returns element at front of the queue. If queue is empty it returns 0

1. If (F==-1)

   then write (" underflow")

   return 0

2. else

   return (Q[F])

Queue Rear:

- QREAR (Q,R) : Given R pointers to the rear of queue , This procedure returns element at rear end of the queue. If queue is empty it returns 0

1. If (f==R==-1)

   then write (" underflow")

   return 0

2. else
   return (Q[R])

Queue empty:

- QEMPTY (Q,F) : Given F pointers to the front of queue , This procedure check the queue is empty or not

1. [checking for underflow]

   If (F==-1)

   then write (" underflow")

   return 0

Queue full:

- QFULL (Q,R,N) : Given R , pointers to the rear elements of queue , a queue Q consisting of N elements .This procedure check queue is full or not .

1. [checking for overflow]

   If (R>=N-1)

   then write (" overflow")

   return

Queue Count:

- QCOUNT (Q,F,R) : Given F and R , pointers to the front and rear elements of queue , a queue Q consisting of N elements and count is temporary variable contain no of elements in queue. This procedure return 0 if queue is empty otherwise returns count.

1. [checking for underflow]

   If (F==-1)

   then write (" Queue is empty ")

   count=0

   return count

2. else

for i=front to rear

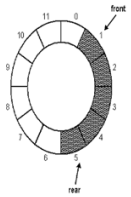
Count ← count + 1

return count

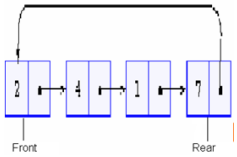
CIRCULAR QUEUE

Q. **Explain Circular queue data structure with suitable example? Write an algorithm for Enqueue , Dequeue ,Queuefront and queuerear**
Or
Q.What is queue? Explain the working of a circular queue and give algorithms for inserting an element and deleting an element from the circular queue.

Elements need not be shifted on insertion.Queue becomes full only when it has no empty space.



Array Implementation



List Implementation

Insertion in circular queue :

- CQINSERT (Q,F,R,N,Y) : Given F and R , pointers to the front and rear elements of circular queue , a queue Q consisting of N elements and y is element which is inserted by this procedure at rear of queue . Initially F and R are set to -1

1. If (R = max-1 && F =0) || (R+1= F))
   then write(" overflow")
   return
2. [Reset rear pointer]
   If (R= max-1)
   then R← 0
   else R← R + 1
3. [Insert element]
   Q[R] ← Y
4. [Is front pointer properly set]
   if F= -1
   Then F← 0
   Return

Deletion from circular queue:

- CQDELETE (Q,F,R) : Given F and R , pointers to the front and rear elements of circular queue , This procedure delete element at front of the queue . Y is temporary variable .

1. [checking for underflow]

If (F== -1)
then write (" underflow")

return

2. [Delete element]

Y← Q[F]

3. [Queue empty]

if F=R

Then F← R ← -1

return (Y)

4. [Increment front pointer]

if (F = max -1)

then F← 0

else F← F +1

Return(Y)

Queue front:

- CQfront (Q,F,R) : Given F and R , pointers to the front and rear elements of circular queue , This procedure gives e element at front of the queue .

2. [checking for underflow]

If (F== -1)

then write (" underflow")

return

3. [retrun front element]

Return(Q[F])

Queue Rear:

- CQrear (Q,F,R) : Given F and R , pointers to the front and rear elements of circular queue , This procedure gives e element at rear of the queue .

3. [checking for underflow]

If (F== -1)

then write (" underflow")

return

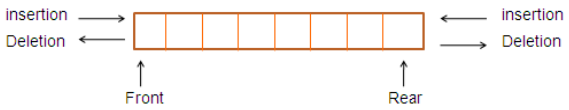
4. [retrun rear element]

Return(Q[R])

DEQUE

Q. Explain Deque

Double-ended queue - Elements can be added to or removed from the front or back.



Types:

1. **Output-restricted deque -** Insertion can be made at both ends, but output can be made from one end only

2. **Input-restricted deque** - Deletion can be made from both ends, but input can only be made at one end.

PRIORITY QUEUE

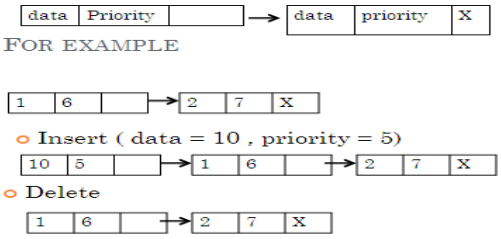
Q.Define and explain queue data structure with suitable example. Give an algorithm for insertion and deletion in priority queue?

- A modified queue in which elements are inserted arbitrarily with an associated priority. On deletion, element with the highest priority is removed from the queue
- Order of returned elements is not always FIFO.

If there are two elements with same priority then process them as per FIFO.

Examples

Processes scheduled by CPU



// Implement Queue using Link List

```
#include<conio.h>
#include<iostream.h>
#include<process.h>
class queue
{
        int info, ele,c;
        queue *node,*link,*start,*move;
public:
        queue()
        {
        start=NULL;
        c=0;
        }
        void insert();
        void del();
        void dis();
};

void queue::insert()
{
        node=new queue;
        if(c<3)
        {
                cout<<"\nEnter Info:";
                cin>>ele;
                node->info=ele;
                node->link=NULL;
                if(start==NULL)
                {
                        start=node;
                        c++;
                        return;
                }
                else
                {
                        move=start;
```

```
                while(move->link!=NULL)
                move=move->link;
                move->link=node;
                c++;
            }
        }
        else
            cout<<"\n Overflow";
}
void queue::del()
{       move=start;
        if(move!=NULL)
        {
            move=move->link;
            cout<<"\nDeleted Element is :"<<start->info;
            start=move;
        }
        else
            cout<<"\nUnderflow";
}
void queue::dis()
{       move=start;
        if(move==NULL)
        {
            cout<<"\n Queue is empty ";
            return;
        }
        else
        {
            while(move!=NULL)
            {
                cout<<move->info<<"\t";
                move=move->link;
            }
        }
}
void main()
{
        clrscr();
        int ch;
        queue s;
        cout<<"\n1.Insert 2.Show 3.Delete 4.Exit";
        while(ch!=4)
        {       cout<<"\nEnter Choice";
            cin>>ch;
```

```
                switch(ch)
                {
                        case 1: s.insert();break;
                        case 2: s.dis();break;
                        case 3: s.del();break;
                        case 4:exit(0);
                }
        }
getch();
}
```