NOTES FROM 27-07-2024 TO 30-07-2024

PRIORITY QUEUE =>

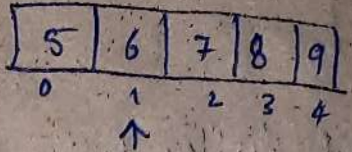
```
-> # define MAX 5
 int q [max], front = -1, rear = -1;
 void insertion ()
 {
    int element;
    if (rear == max-1)
    {
       printf ("FULL");
    }
    else
    {
       printf ("enter element");
       scanf ("%d", & element);
       if ( front == -1)
       {
          front = rear = 0;
           q [rear ]= element;
       }
       else
       {
          int i, j;
          for ( i = front ; i <= rear; i++)
          {
             if (q[i] > element )
             break;
          }
```

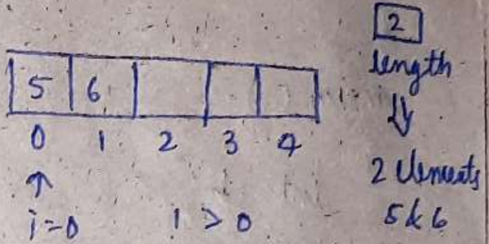
```
        for( j = rear; j>=i; j--)
              q[j+1] = q[j];
                  q[i] = element;
                  rear ++;
        }
    }
}
void deletion ()
{
    int k;
    if (front = = -1)
    {   printf ("EMPTY");
    }
    else
    {
        k = q[front];
        if ( front = = rear )
          front = rear = -1;
          else
           front ++;
           printf (" deleted element is %d", k);
       }
}
void display ()
{
     int i;
       for (i = front; i< = rear; i++)
         printf ("%d", q[i])
   }
```

```
main ()
{
  int ch;
   while (1)
   {
     printf ("Enter 1 ~~for~~ to insert \n 2 ~~for~~ to delete \n
                3 to display \n 4 to exit ");
     scanf ("%d", & ch);
    Switch (ch)
    {
      case 1: insertion (); break;
      case 2: deletion (); break;
      case 3: display (); break;
      case 4: exit (0);
    }
  }
}
```

```
-> # define size 5
 int l [size], length = 0;
 void insertion ()
 {
   int index, element;
   printf ("Enter index");
   scanf ("%d", & index);
   if (length == size)
   {
     printf ("full");
   }
   else if (index < 0 || index > length)
   {
     printf ("Insertion not possible");
   }
```

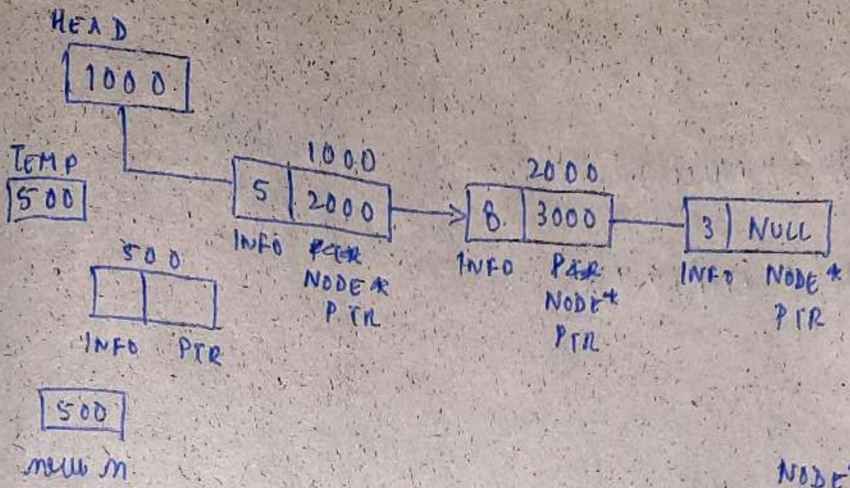
```
else
{ printf ("Enter element.");
  scanf ("%d", & element);
  for (i = length - 1; i >= index; i--)
  {
      l[i+1] = l[i];
  }
    l[index] = element;
    length ++;
  }
}
void deletion ()
{ int index;
  printf (" enter index ");
  Scanf ("%d" & index);
  if (length == 0)
  {
     printf ("EMPTY");
  }
  else if (index < 0 || index >= length)
  {
     printf (" NOT POSSIBLE ");
  }
  else
  {
     printf (" Deleted element is %d", l[index]);
     for (i = index; i < length - 1; i++)
     {
        l[i] = l[i+1];
     }
```





```
    length--;
  }
}
void display ()
{   int i;
    for (i=0; i< length; i++)
    printf ("%d", l[i]);
}
main ()
 {
    int ch;
    while (1)
    {
       printf ("Enter 1 to insert\n 2 to Delete\n 3 to display
              \n 4 to Exit ");
       scanf ("%d", &ch);
        Switch (ch)
        {
          case 1: insertion (); break;
          case 2: Deletion (); break;
          case 3: display (); break;
          case 4: exit (0);
          default:
        }
     }
 }
```

STACK USING LINKED LIST :-



NODE* ptr ⇒ NODE POINTER

LINKED LIST :-

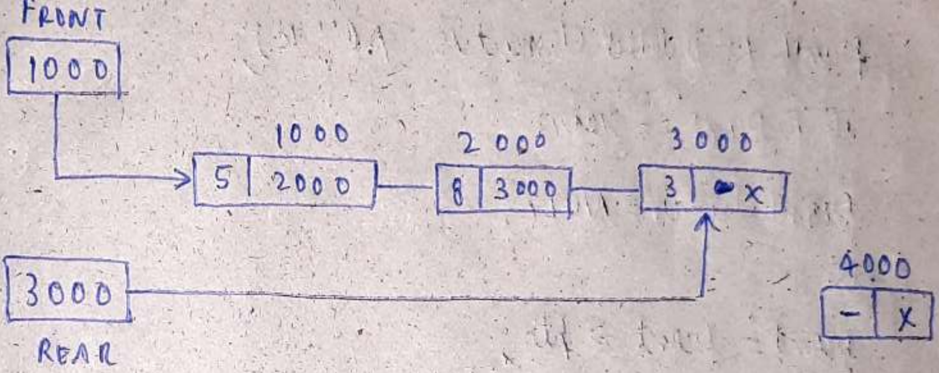
It is logical linear data structure.

It is also known as Self referential structure.

```
=> struct node
   {
     int info;
     struct node * ptr;
   };
   typedef struct node node;
   node *top = NULL, *temp , *newnode;
   void push ( )
   {
     int element;
     printf (" Enter element ");
     scanf ("%d", &element );
     newnode = (node *) malloc ( sizeof ( node ));
     newnode -> info = element;
     newnode -> ptr = top;
         top = newnode;
   }
```

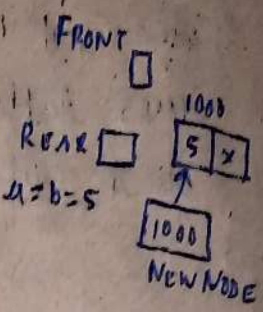
```
void pop( )
{
  int k;
   if (top == NULL)
   {
      printf ("EMPTY");
   }
   else
   {
      k = top -> info;
      top = top -> ptr;
      printf ("Deleted element is %d", k);
   }
}
void Display ( )
{
   temp = top;
   while ( temp ! = NULL)
   {
      printf ("%d", temp -> info);
       temp = temp -> ptr;
   }
}
main ( )
{ int ch;
   while (1)
   {
      printf ("Enter 1 to PUSH \n 2 to POP \n 3 to DISPLAY
                 \n 4 to EXIT");
      scanf ("%d", & ch);
       switch (ch)
{
  case 1: push( ); break;
  case 2: pop ( ); break;
  case 3: display ( ); break;
  case 4: exit (0);
 }
}
}
```

QUEUE USING LINKED LIST =>



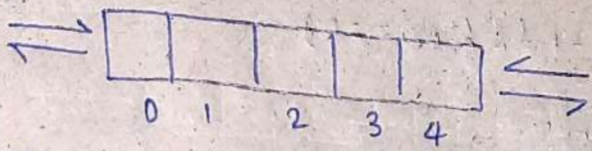
```
-> struct node
   {
     int info;
     struct node *ptr;
   };
   typedef struct node node;
   node = *front = *rear = NULL, *temp, *newnode;
   void insert ( )
   { int element;
     printf ("Enter element: ");
     scanf (" %d", &element);
     newnode = (node *) malloc (sizeof (node));
     newnode -> info = element;
     newnode -> ptr = NULL;
```

```
 if (front = = NULL)
     front = rear = new node ;
 else
     rear = rear -> ptr = newnode;
}
void delete ()
{
  int k;
   k = front -> info ;
   printf ("Deleted element is %d", k);
   if (front == rear)
   front = rear = NULL;
   else
   front = front -> ptr;
 }
 void Display ()
 {
  temp = front;
  while (temp != NULL)
  {
    printf ("%d", temp -> info);
    temp = temp -> ptr;
  }
 }
 void main ()
 {
  int ch ;
   while (1)
   {
     printf (" Enter 1 to INSERT \n 2 to DELETE \n
               3 to DISPLAY \n 4 to EXIT ");
     scanf ("%d" & ch);
```



```
switch (ch)
 {
   case 1: insert(); break;
   case 2: delete(); break;
   case 3: display(); break;
   case 4: exit(0);
 }
}
}
```

DOUBLE ENDED QUEUE :-



insertf ()
INSERT FRONT
insertr ()
INSERT REAR

```
-> #include <stdio.h>
int q[max], front = -1, rear = -1;
void insertf ()
{
  int element;
  if (front == 0 && rear == max-1)
  {
    printf ("FULL");
  }
  else
  {
    printf ("enter element");
    scanf ("%d", &element);
    if (front == -1)
      front = rear = 0;
    else if (front > 0)
      front--;
    else
    {
```

```
    for ( i = rear ; i >= front ; i -- )
    {
        q [ i + 1 ] = q [ i ];
    }
    rear ++
  }
  q [ front ] = element;
 }
}
void insertr ( )                                r(1) -> REAR
{
  if ( rear = = max - 1 && front = = 0 )
  { printf ( "FULL" );
  }
  else
  { printf ( "Enter element" );
    scanf ( "%d", & element );
    if ( front = = -1 )
    {
      front = rear = 0;
       q [ rear ] = element;
    }
    else if ( rear < max - 1 )
    {
      rear ++;
      q [ rear ] = element;
    }
    else
    {
      for ( i = front ; i <= rear ; i ++ )
      {
        q [ i - 1 ] = q [ i ];
        q [ rear ] = element;
      }
    front --;
   }
  }
}                                              f() -> FRONT
void deletef ()
{
   if (front == -1)
   {
     printf ("EMPTY");
   }
   else
   {
     if (front == rear)
     {
        printf ("Deleted element is %d ", q[front]);
         front = rear = -1;
     }
     ~~else~~
     ~~{~~
        ~~printf ("Deleted element is %d ", q[front]);~~
        ~~front = 0;~~
     ~~}~~
     else
     {
         printf ("Deleted element is %d", q[front]);
         front = front + 1;
      }
   }
}
void deleter ()                                r() -> REAR
{
   if (front == -1)
     printf ("EMPTY");
```

```
else
{
   if (front == rear)
   {
      printf ("Deleted element is %d", q[rear]);
      front = rear = -1;
   }
   else if (rear == 0)
   {
      printf ("Deleted element is %d", q[rear]);
      rear = max - 1;
   }
   else ~~if( re~~
   {
      printf ("Deleted element is %d", q[rear]);
      rear = rear - 1;
   }
}
void display ()
{  int i;
   for (i = front; i <= rear; i++)
     printf ("%d", q[i]);
}
void main ()
{  int ch;
   while (1)
   {
    printf ("Enter\n 1 to
   insert front \n
 2 to insert from rear \n
 3 to delete from front \n
 4 to delete from rear \n
 5 to display \n
 6 to exit ");
 scanf ("%d", & ch);
```

```
   switch (ch)
   {
     case 1: insertf(); break;
     case 2: insertr(); break;
     case 3: deletef(); break;
     case 4: deleter(); break;
     case 5: display(); break;
     case6 : exit(0);
   }
}
```