Advantages of Array storage:

Simple to create and easy to use

Direct accessing

Memory allocated for all items at the same time(Allocation and deallocationis easier)

Drawbacks of Array Storage:

Size is fixed, same array can't be grown

Contiguous block of memory allocation

Insertion and deletion operations are costlier

Deleting is done by overwriting, memory allocated for the item will not be deleted. Memory is deleted when the life time of the array ends.

Linked Storage:

Linked list It is a collection of Nodes. Each node contains two fields , one to store information and other to store address of some other node in the collection.

Information filed	Address/link field

Node Representation

Types of Linked lists:

Single Linked liIst

Double Linked liIst

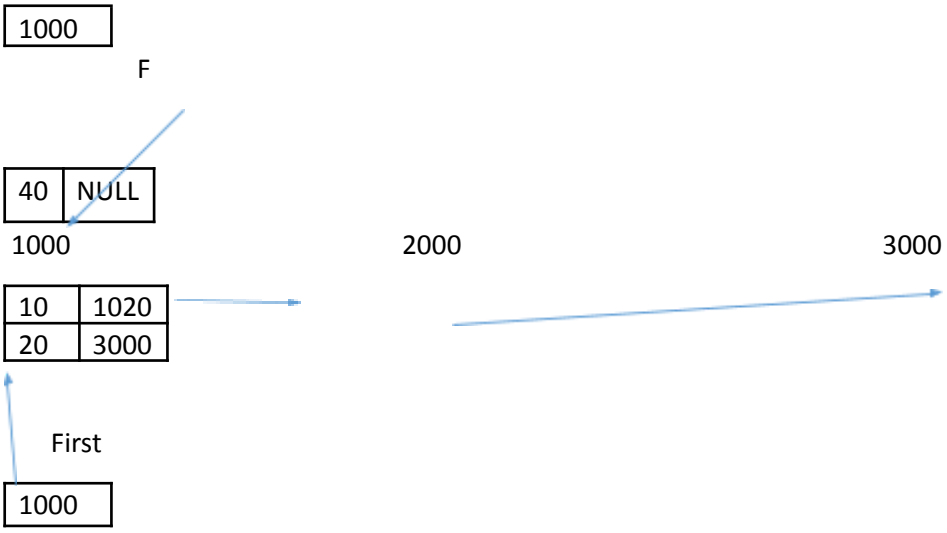
Circular Linked liIst

linked List with Header nodes

Single Linked List Implementation

Node structure:

```
Struct   NODE
{
   Int   info;   //  STUD  info
      Struct NODE   *Link;
};    /// Self  referential structures
```



Operations on Linked lIst:

Traversal

Insertion

Deletion

Updating

Searching

Sorting

Copying

Merging

Implementation of single linked list

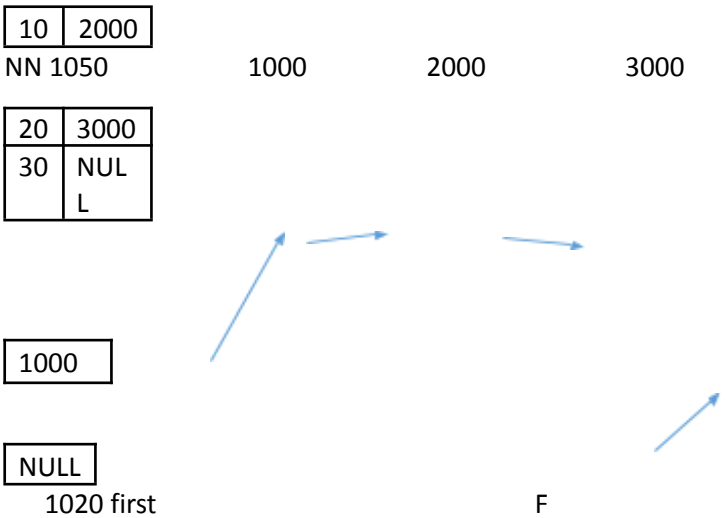
```
#include<stdio.h>
#include<stdlib.h>
struct NODE
{
  int info;
  struct NODE *link;
};
typedef struct NODE * node;
node InsertFront(node);
```

```
node InsertRear(node);
node DeleteFront(node);
node DeleteRear(node);
node InsertPos(node);
node DeletePos(node);
node InsertByOrder(node);
node DeleteKey(node);
void Search(node);
node CreateCopy(node);
node Reverse(node);
void Disp(node);
int NE=0;
void main()
{
   int ch;
   node first= NULL, L2=NULL;
   clrscr();
   for(;;)
   {
      printf("\nenter
choice:\n1:InsertFront\n2:Disply\n3:InsertRear\n4:DeleteFront\n5:DeleteRear\n6
:InsertByposition\n7:DeleteByposition\n8:InsertByOrder\n9:DeleteByKey\n10:Sear
ch Key\n11:Create Copy\n12:Reverse\n");
      scanf("%d",&ch);
      switch(ch)
      {
     case 1: first =InsertFront(first);
          printf("in main %d\n", first->info);
          Disp(first);
          break;
     case 2: Disp(first); break;
     case 3: first=InsertRear(first); Disp(first); break;
     case 4: first = DeleteFront(first); Disp(first); break;
     case 5:
     case 6:
     case 7:
     case 8:
     case 9:
     case 10:
     case 11:
     case 12:
     default:exit(0);
     }
 }
 }
 node InsertFront(node F)
    {
       node NN;
       //int info;
       NN = (node)malloc(sizeof(struct NODE));
       printf("enter info field:");
       scanf("%d",&NN->info);
       NN->link=F;
       NE++;
       return NN;
```

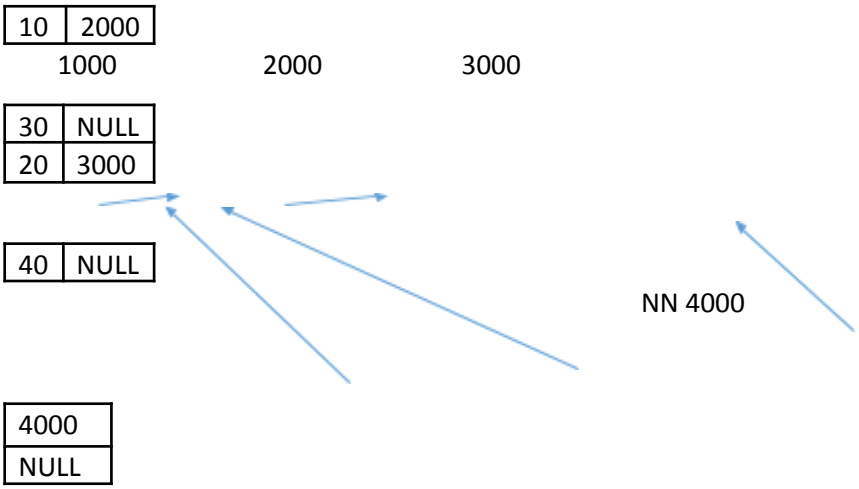
```
    }
```

```
node InsertRear(node F)
    {
       node NN,LN;
       int info;
       NN = (node)malloc(sizeof(struct NODE)); // getnode()
       printf("enter info field:");
       scanf("%d",&NN->info);
       NN->link=NULL;
       NE++;

       if(F== NULL)
       return NN;
       LN=F;
       while(LN->link!=NULL)
       LN=LN->link;
       LN->link=NN;
       return F;
    }
```

Insert Front



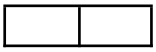
Insert Rear



1020 first F

```
node InsertRear(node F)
    {
       node NN,LN;
       int info;
       NN = (node)malloc(sizeof(struct NODE)); // getnode()
       printf("enter info field:");
       scanf("%d",&NN->info);
       NN->link=NULL;
       NE++;

       if(F== NULL)
       return NN;
       LN=F;
       while(LN->link!=NULL)
       LN=LN->link;
       LN->link=NN;
```

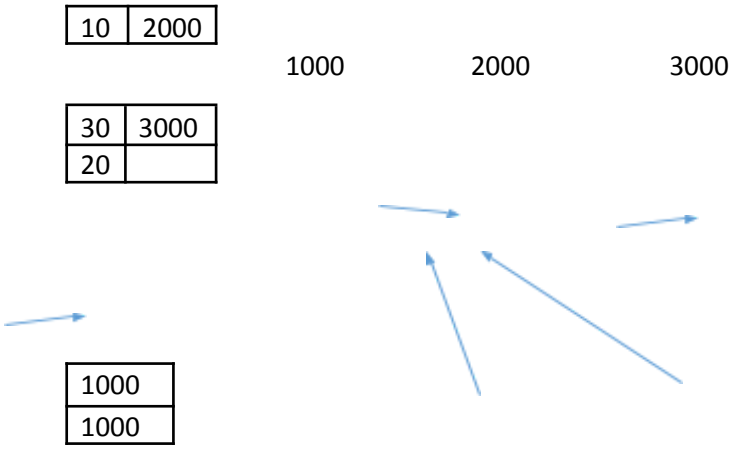
```
        return F;
     }


  void Disp(node F)
   {
       if(F==NULL)
       {
      printf("empty\n");
      return;
       }
       printf("\nList is:\n");
       while(F != NULL)
       {
      printf("%d ",F->info);
      F=F->link;
       }
  }
  node DeleteFront(node F)
  {  node ND;
     if(F==NULL)
       {
      printf("empty\n");
      return  NULL;
       }
       ND=F;
       printf("\ndeleted %d",ND->info);
       F=F->link;
       free(ND);
       NE--;
       return F;
  }



node DeleteRear(node F)
  {
      node LN,PL;


   //  Write the code

     }
```



```
void Search(node F)
 {
   node NS;
   int info, pos;
   if(F==NULL)

   {  printf("empty list");
       return ;
   }
```

```
   printf("\nenter the key to be searched");
   scanf("%d",&info);


//  Write the code to  Traverse the list to search using NS pointer
// Check for successful and unsuccessful



}
```

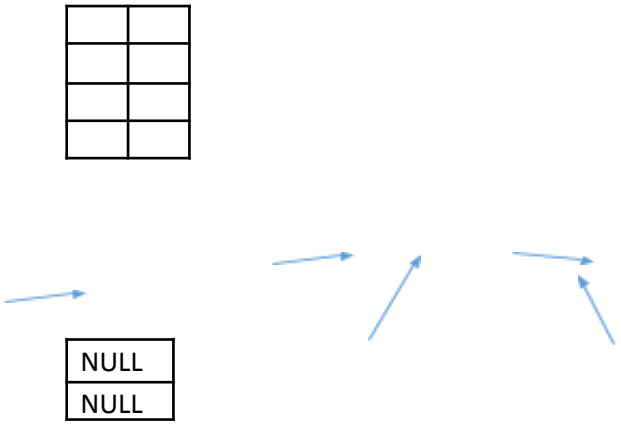
Delete Front



1020 first F

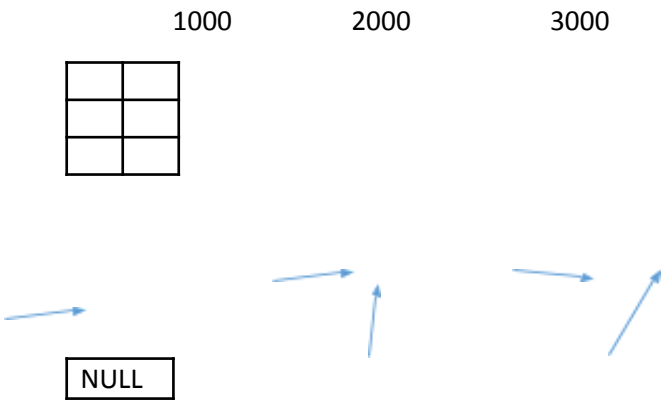
Insert By position (To be discussed)

1000 2000 3000



1020 first F

Delete By position (To be discussed)



NULL

1020 first F