UNIT-3

Linked Lists

* A linked list is a collection of nodes (or) collection of inter-connection nodes

* The linked list is a linear data structure in which the nodes are connected sequencially by the establishing connection from one node to another node.

* Basically the linked lists are classified into 3 types they are
  - i) single linked list
  - ii) Doubly linked list
  - iii) Circular linked list.

Single linked list:-

In the single linked list each node contains two fields they are 1) Data fields 2) link (or) next field

Node structure:- in the single linked list:-

Node

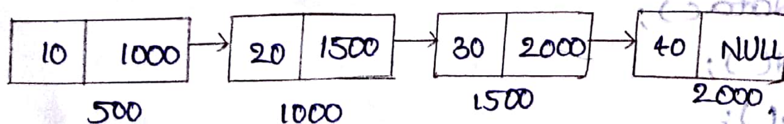
Data field	Link (or) next field

Data field:- The data field can be used to store the data (or) an element.

Link (or) next field:- The link field can be used to store the address of next node in the single linked list

* In the single linked linked list the link field of last node is NULL.

eg:- 10, 20, 30, 40



Operations of Single linked lists:-

On the single linked lists we perform four basic operations they are
1) create
2) Insert
3) Delete
4) Display

Create:- The create operation is used to create a node in order to form a single linked list.

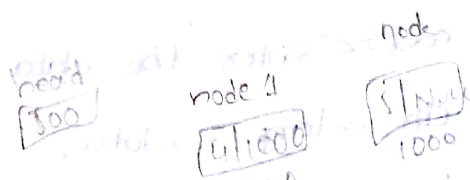
Insert:- This operation is used to insert a node either front (or) middle (or) last to the existing node(s) in the single linked list.

Delete:- In this operation can be used to delete a specific node in the single linked list

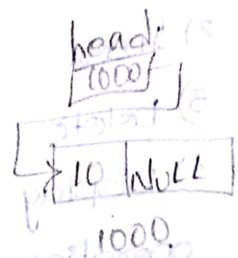
Display:- This operation can be used to display the list of elements in the single linked list.

Ex:- Program to implement single linked list.

```
#include<iostream.h>
# include <conio.h>
# include <process.h>
  class slink
  {
  Struct node  --> structure name
  {
  int data;
  node *next;
  } * head;
public:
    slink()  --> default Constructor
    {
    head = NULL;
    }
```

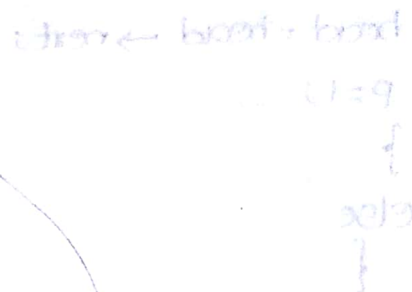


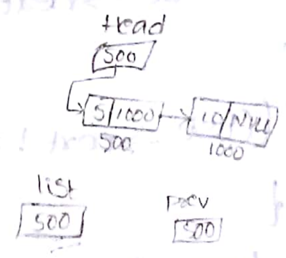
```
void  create ();
void  insert ();
void  delnode ();      member functions
void  display ();
};
```



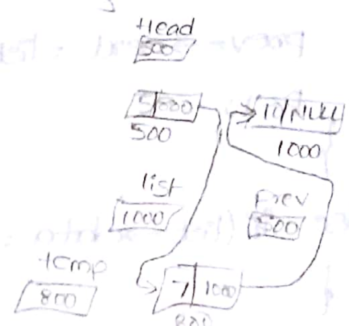

```
void slink::create()
{
 head = new node;      (operator)
 cout<<"Enter the data for the node:";
 cin>>head->data;
 head->next =NULL;
}

void slink::insert()
{
 int n;
 char p;
 node *list, *prev, *temp;
 list =head;
 cout<<"Insert the node at front/middle/last? f/m/l:";
 cin>>p;
 if (p=='f')
 {
 temp = new node;
 cout<<"Enter the data for the node:";
 cin >>temp ->data;
 temp ->next =head;
 head =temp;
 }
 if (p=='m')
 {
 cout<<"Enter the data for the node before to insert new
                                     node at middle";
 cin>>n;
```

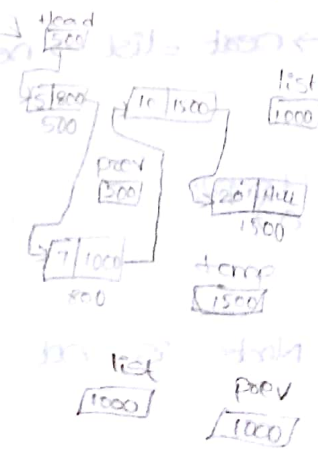
```
while(list -> data != n)
  {
    prev = list;
    list = list -> next;
  }
  temp = new node;
  cout<<"Enter the data for the node";
  cin>>temp -> data;
  prev -> next = temp;
  temp -> next = list;
 }
```



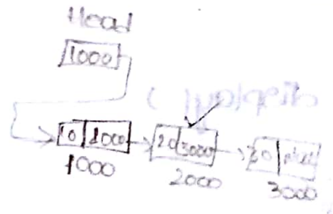



```
if (p=='l')
 {
  while (list -> next != NULL)
   {
    list = list -> next;
   }
   temp = new node;
   cout<<"Enter the data for the node:";
   cin>>temp -> data;
   list -> next = temp;
   temp -> next = NULL;
  }
}
```

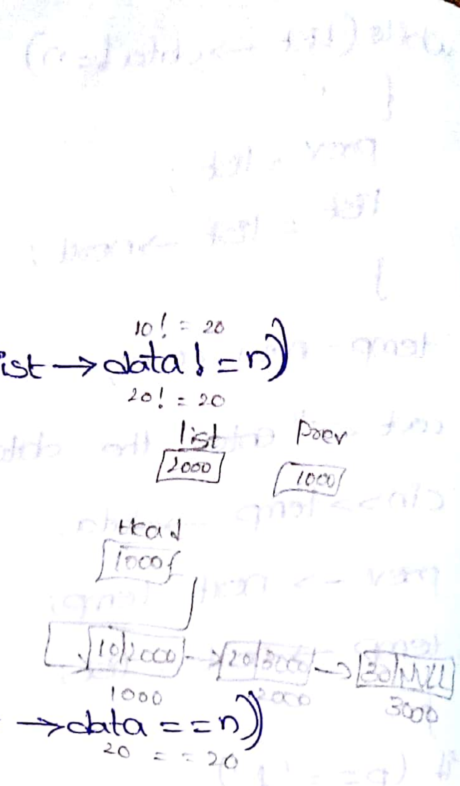


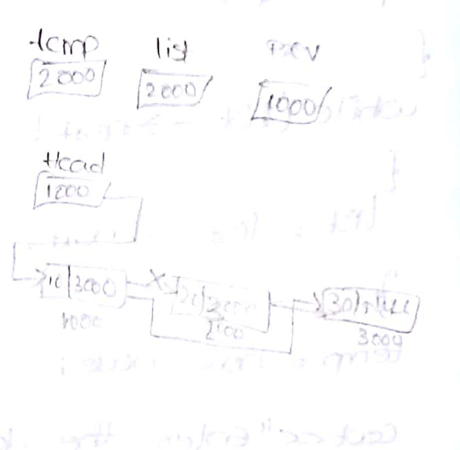


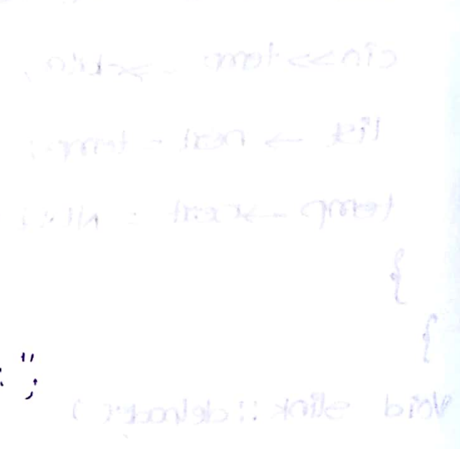

```
Void slink::delnode()
{
 int n, p=0;
 node *list, *prev, *temp;
 list = head;
 cout<<"Enter the data of the node to be deleted:";
 cin>>n;
 if (head->data==n)
  {
```

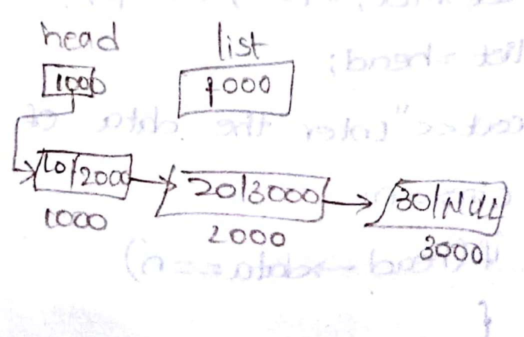


```
head = head -> next;
p=1;
}
else
{
while ((list -> next ! = NULL) && (list -> data ! = n))
{
prev = list ;
list = list -> next ;
}
if ((list -> next ! = NULL) && (list -> data == n))
{
temp = list;
prev -> next = list -> next;
p=1;
}
else if (list -> data == n)
{
temp = list;
prev -> next = list -> next;
p=1;
}
}
delete (temp);
if (p==0)
{
cout<<"Node is not present :";
}
}
void slink :: display( )
{
node *list;
list = head;
if (list == NULL)
{
cout<<"list is empty:";
```

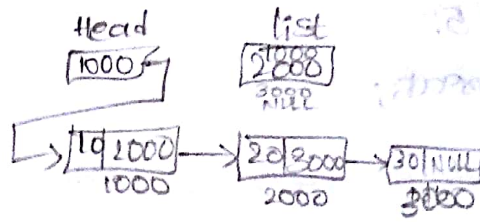








```
    }
else
  {
  while (list != NULL)
                1000 != NULL
                2000 != NULL
                3000 != NULL
   {
    cout<<list->data<<"<=>";
    list = list->next;
                1000 -> 2000
                2000 -> 3000
   }
   cout<<"NULL";
   } //else over
  } //display over
  void main( )
  {        -> class obj
   slink s;
   int option;
   clrscr( );
   cout<<"1.create( )"<<endl;
   cout<<"2.insert( )"<<endl;
   cout<<"3.delnode( )"<<endl;
   cout<<"4.display( )"<<endl;
   cout<<"5.exit"<<endl;
   do
    {
     cout<<"\n Enter your choice:";
     cin>>option;
     switch (option)
      {
 case 1:
     s.create( );
     break;
 case 2:
     s.insert( );
     break;
 case 3:
    s.delnode( );
    break;
```



O/P

10<=>20<=>30<=>NULL

```
case 4:
   s.display();
   break;
case 5:
   break;
   }
}
while (option!=5);
getch();
}
```

1!=5
2!=5
3!=5
4!=5
5!=5 → false

Double linked list:-

In the doubly linked list each node contains 3 fields they are

1. left link field
2. data field
3. Right link field

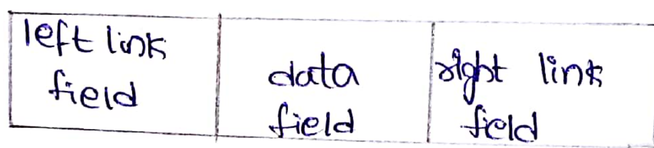
1. **left link field:-** The left link field is used to store the address of previous node

2. **Data field:-** The data field is used to store the data (or) an element

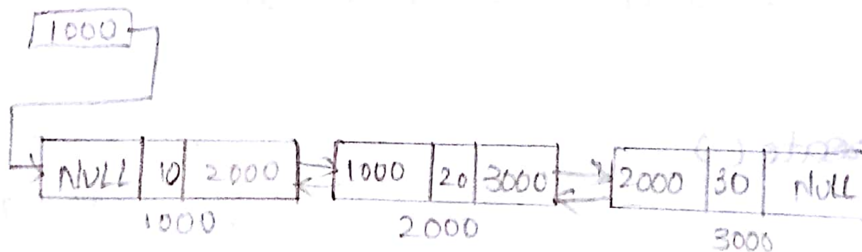
3. **Right link field:-** the right link field is used to store the address of next node.

* In the doubly linked list the left link field of the first node is NULL and also the right link field of the last node is NULL.

Node structure of the doubly linked list:-

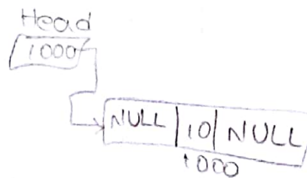


Eg:- head

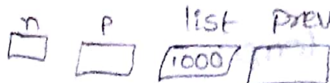


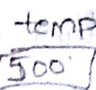
Program to implement doubly linked list:—

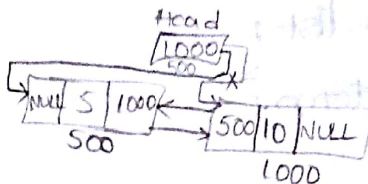
```
#include <iostream.h>
# include <conio.h>
# include <process.h>
class Dlink
{
struct node
{
node *llink;
int data;
node *rlink;
}* head;          node type (or) }; node *head;
public:
Dlink()
{
head = NULL;
}
void create();
void insert();
void delnode();
void display();
};
void Dlink::create()
{
head = new node;
cout<<"Enter the data for the node:";
cin >>head->data;
head->llink = NULL;
head->rlink = NULL;
}
```



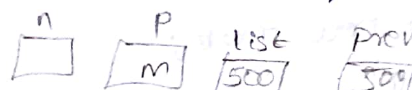
```
void Dlink::insert( )
{
int n;
char p;
node *list,*prev,*temp;
list = head;
cout<<"Insert the node at front/middle/last? f/m/l:";
cin >>p;
if (p=='f')
{
temp = new node;
cout <<"Enter the data for the node:";
cin >>temp ->data;
temp->rlink =head;
head -> llink = temp;
temp -> llink = NULL;
head =temp;
}
if (p=='m')
{
cout<<"Enter the data of a node,before to insert new
node. as middle:";
cin>>n;
while (list ->data! =n)
{
prev = list;
list = list ->rlink;
}
temp = new node;
cout <<"Enter the data for the node:";
cin>>temp->data;
```

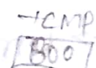


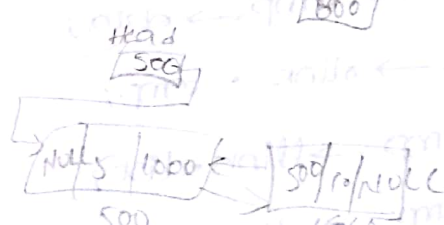




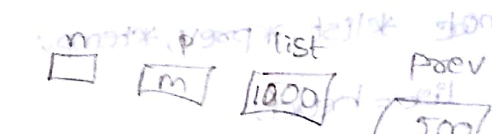
↳ node-type pointer variables

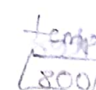


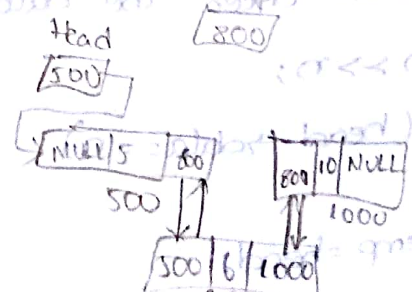




5!=10
10!=10

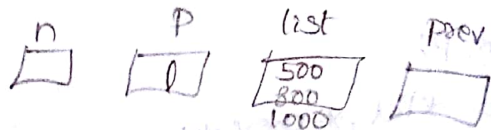


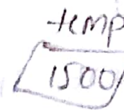




Scanned by CamScanner

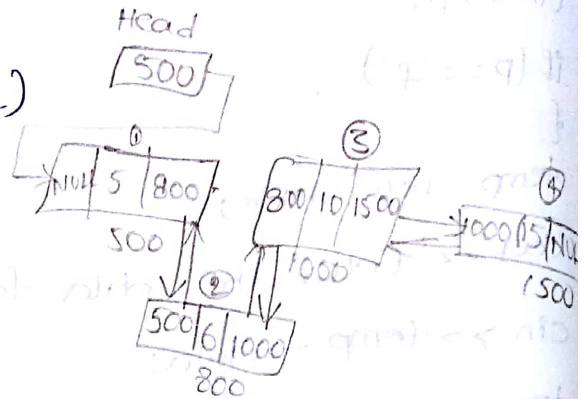
```
prev -> rlink = temp;
temp -> llink = prev;
temp -> rlink = list;
list -> llink = temp;
}
```





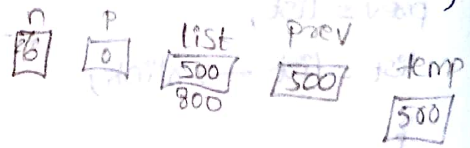

```
if (p == 'l')
{
while (list -> rlink != NULL)
  {
  list = list -> rlink;
  }
 temp = new node;
 cout<<" Enter the data for the node:";
 cin >>temp-> data;
 list -> rlink = temp;
 temp -> llink = list;
 temp -> rlink = NULL;
}
```

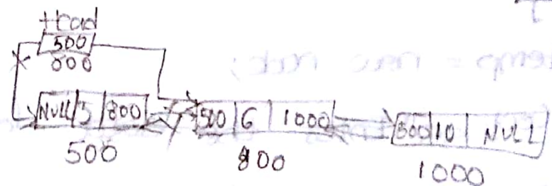
500 != NULL
800 != NULL
1000 != NULL
NULL != NULL



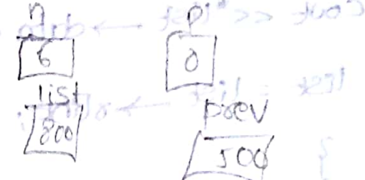

```
void Dlink::delnode()
{
 int n, p=0;
node *list, *prev, *temp;
 list=head;
 cout<<" Enter the data of a node to be deleted:";
 cin >>n;
 if ( head ->data==n)
{
 temp =head;
 head=head -> rlink;
 head -> llink = NULL;
 p=1;
delete (temp);
```

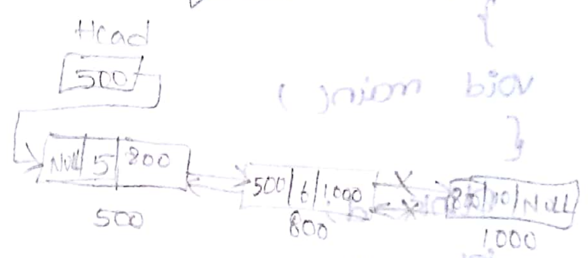
5 == 5
5 == 6

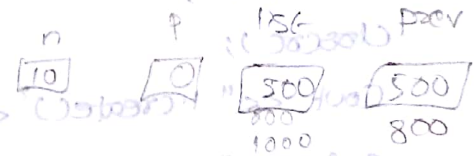


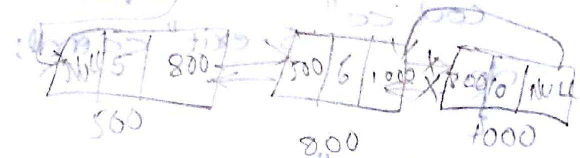


```
}
else
{
while((list->rlink!=NULL) && (list->data!=NULL))
{
prev=list;
list=list->rlink;
}
if((list->rlink!=NULL) && (list->data==n))
{
temp=list;
prev->rlink=list->rlink;
list->rlink->llink=prev;
p=1;
delete(temp);
}
else if(list->data==n)
{
temp=list;
prev->rlink=list->rlink;
p=1;
delete(temp);
}
}
if (p==0)
{
cout<<"Node is not Present:";
}
}
void Dlink::display()
{
node *list;
list=head;
if (list==NULL)
```

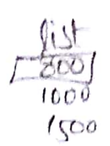


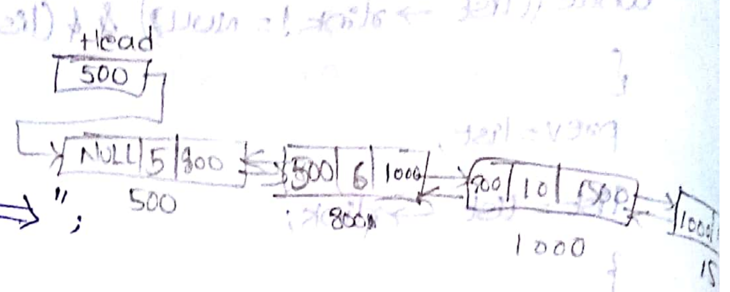


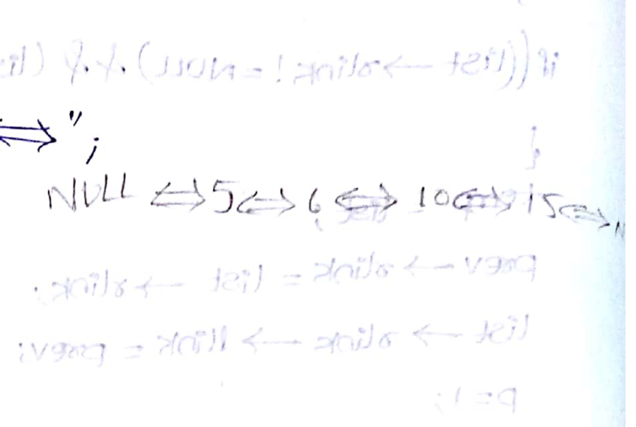


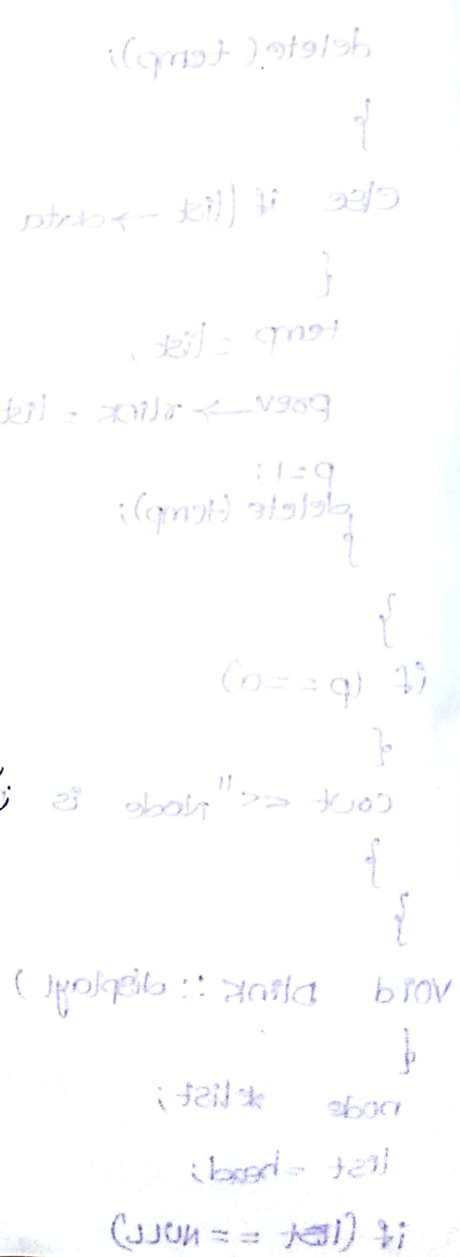


```
{
Cout << "list is empty";
}
else
{
cout << " NULL" << " <=> ";
while (list != NULL)
{
cout << list -> data << " <=> ";
list = list -> rlink;
}
cout << " NULL";
}
}
void main( )
{
Dlink d;
int option;
clrscr( );
Cout << " 1.create()" << endl;
Cout << " 2. insert( )" << endl;
cout << " 3. delnode( )" << endl;
cout << " 4. display( )" << endl;
cout << " 5. exit" << endl;
do
{
cout << "\n Enter your choice:";
cin >> option;
Switch (option)
{
case 1:
d. create( );
break;
case 2:
d. insert( );
```









```
        break;
case 3:
        d.delnode();
        break;
case 4:
        d.display();
        break;
case 5:
        break;
    }
  }
 while (option != 5);
 getch();
}
```

Circular linked list:-

The circular linked list has been classified into two types they are 1. circular single linked list
2. circular doubly linked list.

1. **Circular single linked list:-** In this each node contains two fields they are. 1. Data field
2. link (or) next field.

Node structure:-

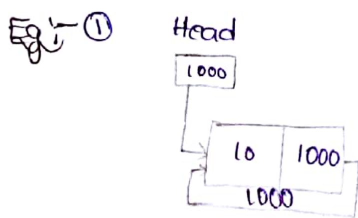
Data field	link (or) next field

Data field:- The Data field can be used to store the data (or) an element

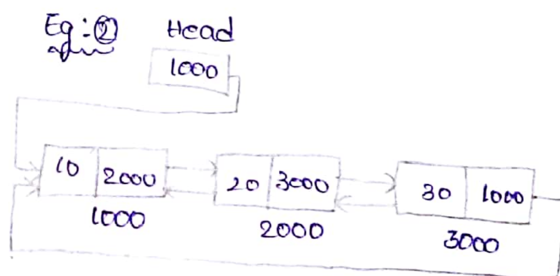
link (or) next field:- the link field can be used to store the address of next node.

* In the circular single linked list the last node link field holds the address of first node in order to make the linked list as circular

Eg:- ①



Eg:- ②



2. **Circular doubly linked list:-** In this each node contains 3 fields they are left link field, Data field, right link field.

Node structure:-

left link field	Data field	right link field

Left link field:- left link field holds the address of previous node.

Data field:- Data field can be used to store the data (or) an element.

Right link field:- Right link field holds the address of next node

* In the circular doubly linked list, the last node right link field stores the address of first node and the first node left link field stores the address of last node, In order to make the linked list has circular.

Eg: ①

head
1000

1000	10	1000

1000

Eg ②

head
1000

3000	10	2000

1000

1000	20	3000

2000

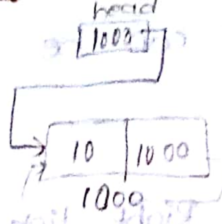
2000	30	1000

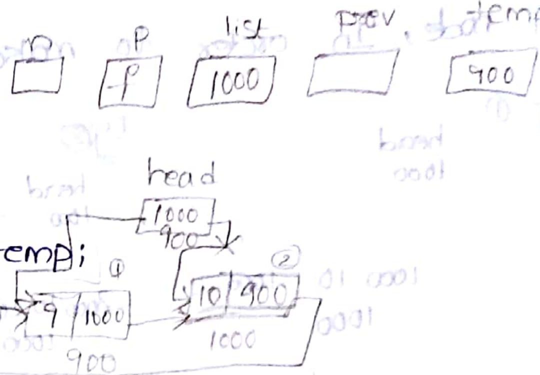
3000

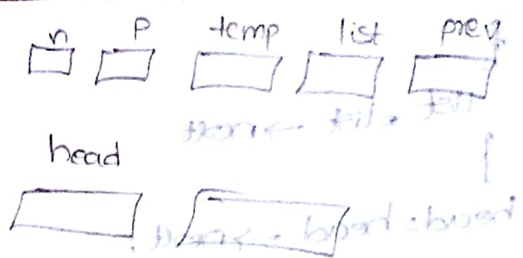
program to illustrate circular single linked list:-

In linked list it forms nodes. the nodes of addresses are processing that is the reason we are using <process.h>

```
#include <iostream.h>
# include <conio.h>
# include <process.h>
  class clink
  {
  struct node
  {
   int data;
   node *next;
  }*head;
  public:
    cslink ( )
    {
      head = NULL;
    }
   void create ( );
   void insert ( );
   void delnode ( );
   void display ( );
 };
```

```
void cslink::create()
{
 head = new node;
 cout<<"Enter the data for the node:";
 cin>>head->data;
 head->next = head;
}
```


```
void cslink::insert()
{
 int n;
 char p;
 node *list, *prev, *temp;
 list = head;
 cout<<"Insert the node at front/middle/last ? f/m/l:";
 cin>>p;
 if (p=='f')
 {
 temp = new node;
 cout<<"Enter the data for the node:";
 cin>>temp->data;
 temp->next = head;
 head->next = temp;
 head = temp;
 }
 if (p=='m')
 {
 cout<<"Enter the data for the node, before to insert
 new node as middle:";
 cin>>n;
 while(list->data!=n)
 {
```




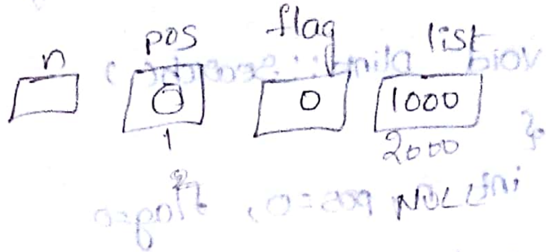

```
prev=list;
list = list -> next;
}
temp = new node;
cout<<"Enter the data for the node:";
cin>> temp -> data;
prev ->next =temp;
temp ->next = list;
}
if (p=='l')
{
while (list ->next != head)
{
list =list -> next;
}
temp = new node;
cout <<"Enter the data for the node:";
cin >> temp ->data;
list ->next =temp;
temp -> next = head;
}
}
void cslinks::delnode()
{
int n, p=0;
node *list, *prev, *temp;
list=head;
cout<<"Enter the data of a node to be deleted:";
cin>>n;
if (head->data==n)
{
temp=head;
while(list ->next!=head)
```

```
{
list = list → next
}
head = head → next;
list → next = head;      // first node
p = 1;
delete ( temp);
}
// if
else
{
while ((list → next ! = head) && (list → data ! = n))
{
prev = list;
list = list → next;
}
if ((list → next ! = head) && (list → data = = n))
{
temp = list;
prev → next = list → next;
p = 1;                   // middle node
delete (temp);
}
else if (list → data = = n)
{
temp = list;
prev → next = list → next;     // last node
p = 1;
delete (temp);
}
} //else
if ( p = = 0)
{
```

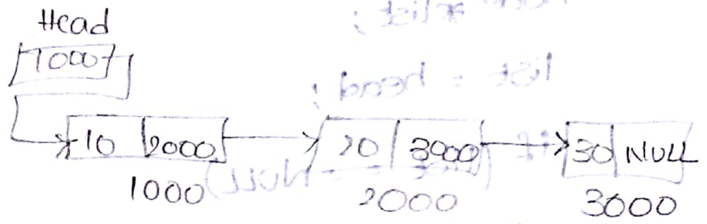
```
cout <<"Node is not present:";
 }
}

 void cslink::display( )
 {
 node *list;
 list =head;
 if (list = =NULL)
  {
  cout<<"list is empty:";
  }
 else
  {
  while(list ->next!=head)
   {
   cout<<list ->data <<"<=>";
   list = list ->next ;
   }
  cout<<list->data;
}
void main( )
{
 cslink cs;
 int option;
 clrscr( );
 cout <<"1.create ( )"<<endl;
 cout <<" 2. insert ( )"<<endl;
 cout <<" 3. delnode ( )"<<endl;
 cout <<" 4. display( )"<<endl;
 cout <<"5. exit "<<endl;
do
 {
 cout <<"\n Enter your choice:";
 cin >> option;
 switch (option)
```

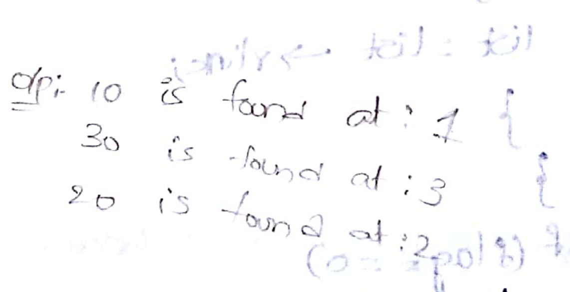
```
{
case 1:
    cs.create();
    break;
case 2:
    cs.insert( );
    break;
case 3:
    cs.delnode( );
    break;
case 4:
    cs.display( );
    break;
case 5:
    break;
  }
}
while (option != 5);
getch( );
}
```

Search( ) operation for the single linked list:-

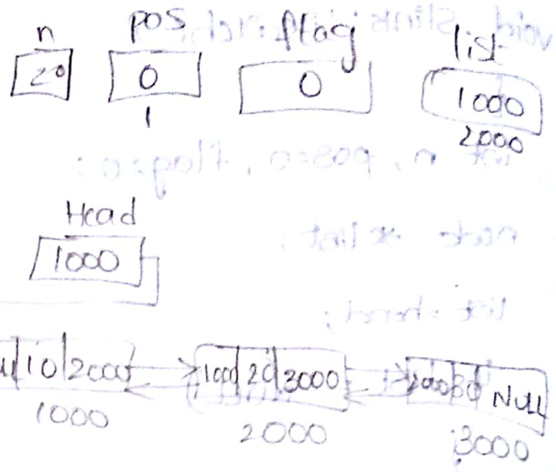



```
void slink::search( )
{
int n, pos=0, flag=0;
node *list;
list =head;
if (list = =NULL)
{
cout<<"list is empty";
}
else
{
cout<<" Enter the data of a node to be searched:";
cin>>n;
while(list! =NULL)
{
pos++;
if (list → data= =n)
{
cout<<list→data <<" is found at:"<<pos<<endl;
flag=1;
}
list = list →next;
}//while
}//else
if (flag==0)
{
cout<<"Node is not present with the given element:";
}
}//search( )
```





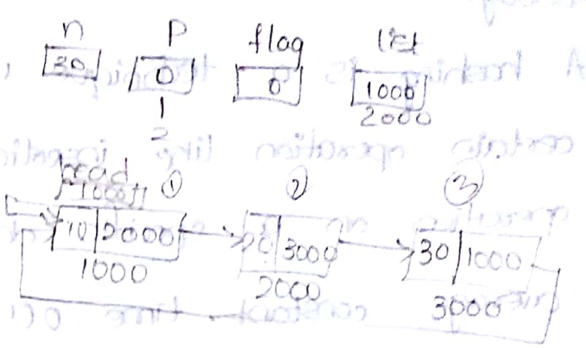
Search() operation for doubly linked list:-




```
void Dlink :: Search( )
{
 int n, pos=0, flag=0
 node *list;
 list = head;
 if (list = = NULL)
  {
  Cout<<"list is empty!";
  }
 else
  {
  Cout<<"enter the data of a node to be searched:
  cin>>n;
  while (list! =NULL)
  {
   pos++;
   if (list ->data = =n)
   {
   Cout<<list ->data <<"is found at:"<<pos <<endl;
   flag=1;
   }
  list =list ->rlink;
  }
 }
if (flag= =0)
{
Cout<<"Node is not present with the given element:";
}
}
```

o/p:- 20 is found at : 2

search() operation for circular linked list:-




```
Void CSlink::search( )
{
int n, p=0, flag=0;
node *list;
list =head;
if (list == NULL)
{
cout<<"list is empty:";
}
else
{
cout<<"Enter the data of a node to be searched:";
cin>>n;
while (list->next!=head)
{
pos++;
if (list->data==n)
{
cout<<list->data<<" is found at node position:"<<pos<<endl;
flag=1;
}
list=list->next;
} //while
if (list->data==n)
{
pos++;
cout<<list->data<<" is found at node position:"<<pos<<endl;
flag=1;
}
} //else
if (flag==0)
{
cout<<"Node is not present with the given element:";
}
```

o/p : 20 is found at node position: 2

Hashing:-

* A hashing is a technique which is used to perform certain operation like insertion, deletion and search operation on a specific data structure with an average constant time O(1) [order of 1].

* Using the hashing technique the search operation can be performed very efficiently when compare with some of the technique like linear, binary search and fibonacci search etc.

* The hashing technique can be used usually to implement a special data structure is known as dictionary. A dictionary is set of elements (or) contain.

* The hashing technique contains are uses three key components in order to perform insertion, deletion and search on a data structure. They are.

1. Hash table
2. Hash function mapping
3. collision

Hash table:-

* A hash table is a data structure which contains a fixed length array with set of index values to store (or) search (or) retrie an element into (or) from the hash table.

* The index of the hash table starts from 0 to N-1 where 'N' is the size of the hash table.

Hash table (6)

0 to N-1

0
1
2
3
4
5

* In the hash table each key element is mapped (or) associated with the corresponding index value of the hash table in order to perform the operation insertion deletion and search on the hash table.

Hash function:-

* the hash function is a function which can be used to map the key elements with the corresponding index values of the hash-table to perform the operation (insertion, deletion and search)

* The value which is generated by the hash function can be called hash code (or) hash address (or) index of the hash table.

The hash function is broadly classified into 4 types based

1. Division method.
2. Mid-square method
3. folding method
4. Multiplication method

*. The hash function can be denoted by 'H'.

Division Method:-

* In this method, each key element will be modulo divided by hash table size and the remainder value can be treated as hash address (or) location address of the hash table. where we can insert (or) delete (or) search the particular key element into (or) from hash table.

* The general form of this Method is

$$H(k) = k \bmod m$$

where

K = key

M = hash table size.

Ex:-

$m = 10$

keys = 20, 28, 32, 44

$H(20) = 20 \% 10$
$= 0$

$H(28) = 28 \% 10 = 8$

$H(32) = 32 \% 10 = 2$

$H(44) = 44/10 = 4$

Hash table

0	20
1	
2	32
3	
4	44
5	
6	
7	
8	28
9	

Mid-Square Method:-

* In this method each key element is squared and middile bits are extracted from the squared number depending up on the size of the hash table and also the middle bits will be modulo divided by hash table size and the remainder value can be considered as hash address (or) location address of the hash table.

$m = 10$

keys = 6, 72, 15, 38

$H(K) = K^2 = 6^2$

$= 36 \% 10 = 6$

$(72)^2 = 5184 \% 100 = 18 \% 10 = 8$

$(15)^2 = 225 \% 100 = 22 \% 10 = 2$

$(38)^2 = 1444 \% 100 = 44 \% 10 = 4$

0	
1	
2	15
3	
4	38
5	
6	36
7	
8	72
9	

Folding Method:- This method is divided into two types. They are 1. fold shifting method
2. fold boundary method

Fold shifting method:-

* In this Method the key element is divided into equal partitions based on the hash table size and all the partitions will be added and the sum will be considered as the location address of the hash table. If any carry is generated while performing addition operation between all the partitions, the carry will be ignored.

Eg:- $m = 10$

$K = 123456$

$= 1+2+3+4+5+6$

$= 21$ → location address of hash table

carry is ignored

Fold boundary Method:-

* The procedure of this method is exactly similar to the fold shifting method, except that the first and last partitions will be reversed and also all the partitions will be added and then the sum will be considered as hash address if any carry is generat

that will be ignored.

Eg:- $m = 100$

$K = 123456$

$= 12\ 34\ 56$

$= 21 + 34 + 65$

$= (1)20$ → sum

carry is ignored.

Multiplication Method:-

In this Method the key value is multiplied by a constant value and from the result the fraction part will be extracted and that will be multiplied by size of the hash table and from the final result the integer part will be considered as the hash address

$m = 10, \quad K = 20$

$K \times A$

$A = \frac{\sqrt{5} - 1}{2}$

$A = 0.618033$

$KA = 20 \times 0.618033$

$= 12.36066$

$m[KA] = 10 \times 0.36066$ (0.36066 → fractional part)

$= 3.6066$

Integer part $= 3$

The general form of this method is $H(K) = m[KA]$

This method will work efficiently by taking the constant value $A = \frac{\sqrt{5}-1}{2} = 0.618033$

$K \times A \quad m = 10 \quad A = \frac{\sqrt{5} + 1}{2}$

$= 0.618033$

20×0.618033

$= 12.36066$

0.36066 fraction part

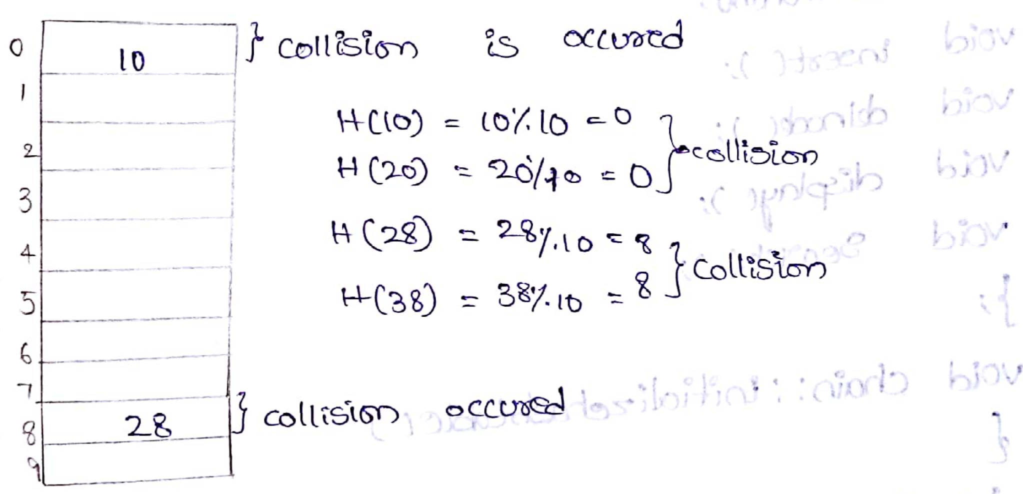
Collision:-

The collision is a problem (or) situation which will be occured when more than one element need to be stored into the same location address of the hash table.

The collision problem can be avoided by using two types of techniques there are 1. open hashing
2. closed hashing

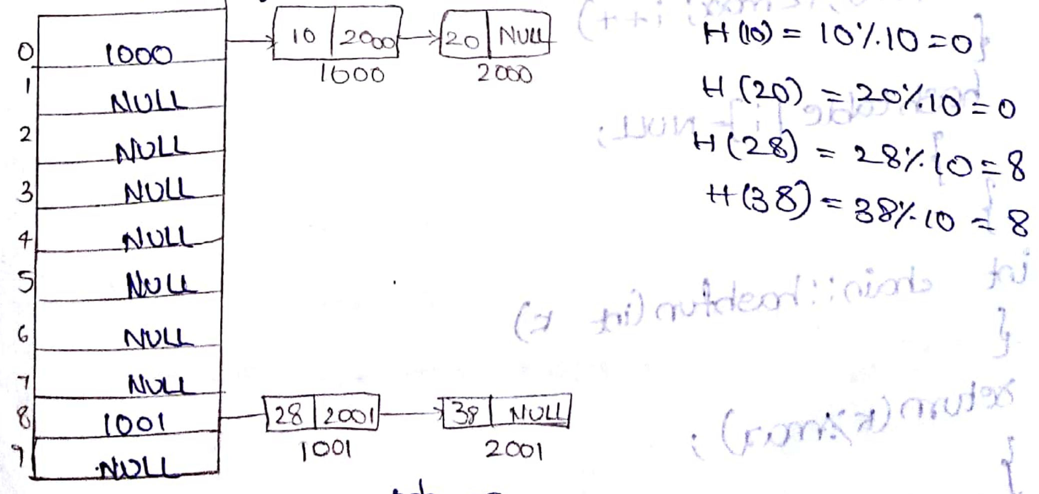
open hashing:- This technique can also be called as chaining (or) seperate chaining this technique can be used to resolve the collision problem by using linked lists

Eg:- m = 10
k = 10, 20, 28, 38



(a) collision

m = 10, keys = 10, 20, 28, 38

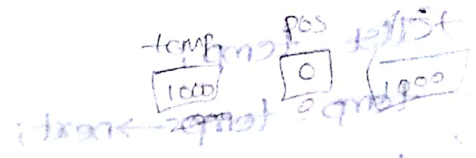


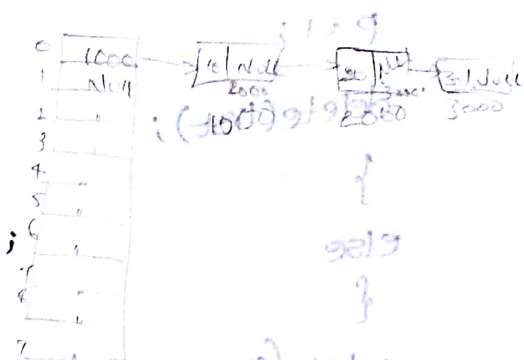
(b) collision avoidance

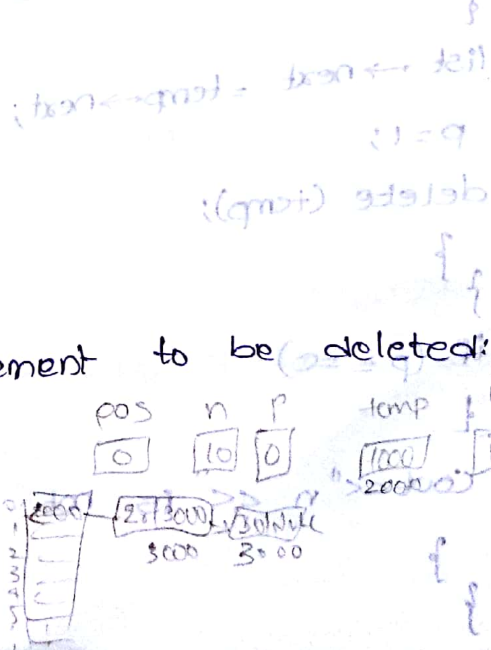
Program to implement chaining

```
#include <iostream.h>
# include < conio.h >
# define max
class chain
{
struct node
{
int data;
node *next;
}*hashtable[max];
public:
void initializehashtable( );
int hashfun(int);
void insert( );
void delnode( );
void display( );
void search( );
};
void chain::initializehashtable( )
{
int i;
for (i=0; i<max; i++)
{
hashtable [i]= NULL;
}
}
int chain::hashfun (int k)
{
return (k%max);
}
void chain::insert( )
{
```

```
int pos;
node *temp,*list;
temp =new node;
cout <<" Enter the data for the node:";
cin >> temp -> data;
temp->next = NULL;
pos = hashfun(temp ->data);
if(hashtable [pos] ==NULL)
{
hashtable [pos] = temp;
}
else
{
list=hashtable [pos];
while(list ->next != NULL)
{
list =list ->next;
}
if (list ->next == NULL)
{
list ->next = temp;
}
}
}

void chain::delnode( )
{
int pos, n, p =0;
node *temp,*list;
cout <<"Enter the element to be deleted:";
cin >>n;
pos=hashfun (n);
temp=hashtable [pos];
if(temp->data ==n)
```

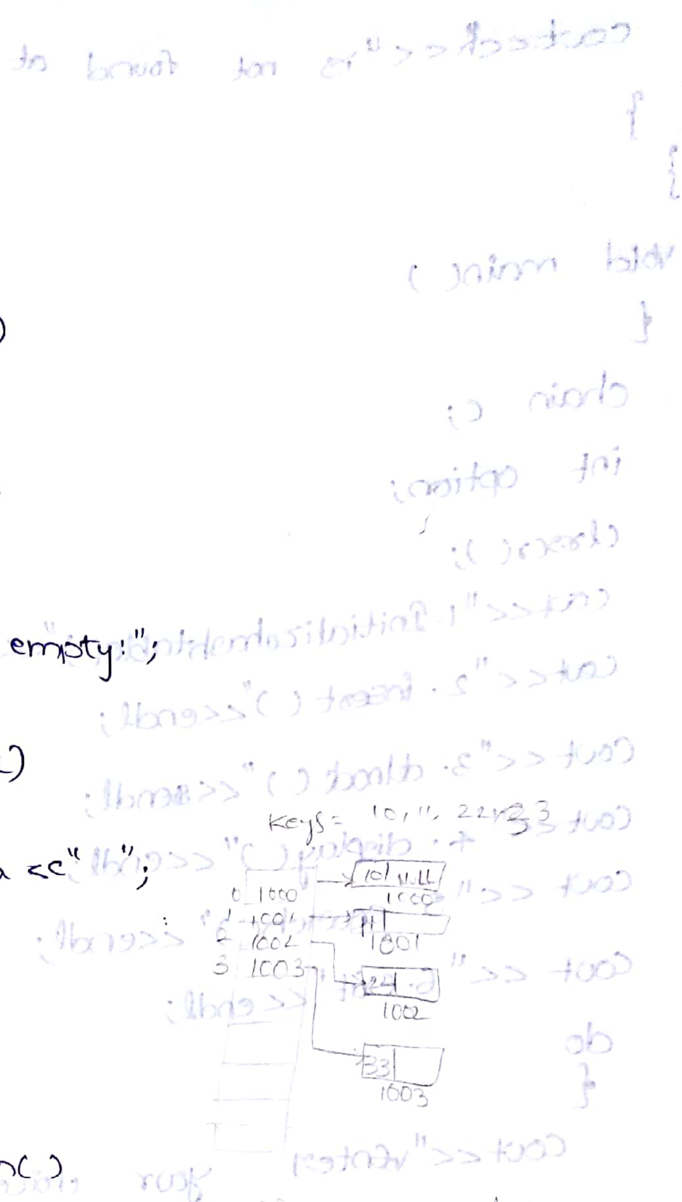




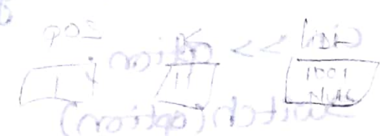


```
{
list = temp;
temp = temp->next;
hashtable [pos] = temp;
p = 1;
delete(list);
}
else
{
while((temp ->next! =NULL) && (temp->data! =n))
{
list = temp;
temp = temp ->next;
}
if((temp->next ! = NULL) && (temp->data = =n))
{
list -> next = temp ->next;
p=1;
delete (temp);
}
else if (temp->data = =n)
{
list -> next = temp->next;
p=1;
delete (temp);
}
}
if (p = =0)
{
cout<<"n <<" is not found at position: "<<pos<<endl
}
}
```

```
void chain::display()
{
int i;
node *list;
for (i=0; i<max; i++)
 {
 list = hashtable [i];
 if (list == NULL)
  {
  cout<<" list is empty:";
  }
  while(list != NULL)
   {
   cout<<list ->data <<" ";
   list=list ->next;
   }
  }
 }
```



```
void chain::search()
{
 int pos, k;
 node *list;
 cout<<"Enter the element to be searched:";
 cin>>k;
 pos = hashfun (k);
 list=hashtable [pos];
 while ((list != NULL) && (list ->data != k))
  {
  list = list ->next;
  }
 if ((list != NULL) && (list ->data == k))
 {
 cout<<list ->data <<" is found at position:"<<pos
                                            <<endl;
 }
 else
 {
```



```
cout<<ch<<"is not found at position"<<pos<<endl;
  }
}

void main( )
{
  chain c;
  int option;
  clrscr( );
  cout<<"1.Initializehashtable( )"<<endl;
  cout<<"2. insert ( )"<<endl;
  cout<<"3. delnode ( )"<<endl;
  cout<<" 4. display( )"<<endl;
  cout <<"5. search( )" <<endl;
  cout <<" 6. Exit"<< endl;
  do
  {
    cout<<"\nEnter your choice:";
    cin >> option;
    switch(option)
    {
  case 1:
      c.initializehashtable( );
      break;
  case 2:
      c. insert ( );
      break;
  case 3:
      c.delnode ( );
      break;
  case 4:
      c.display( );
      break;
```

```
case 5:
      c.search( );
      break;
case 6:
     break;
     }
    }
  while(option!=6);
  getch( );
}
```