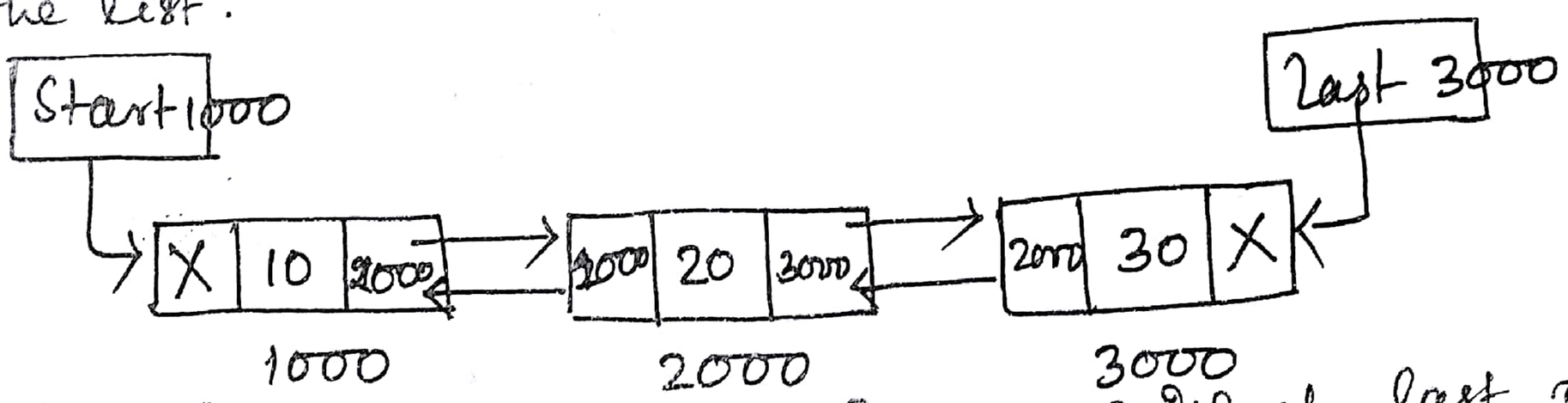
Double linked list

Disadvantage of single linked list :- The main disadvantage of single linked list is that the inability to traverse the list in the backward direction.

Def :-

→ A double linked list is one in which all nodes are linked together by multiple number of links which help in accessing both the sucessor node and predecessor node from the given node position.

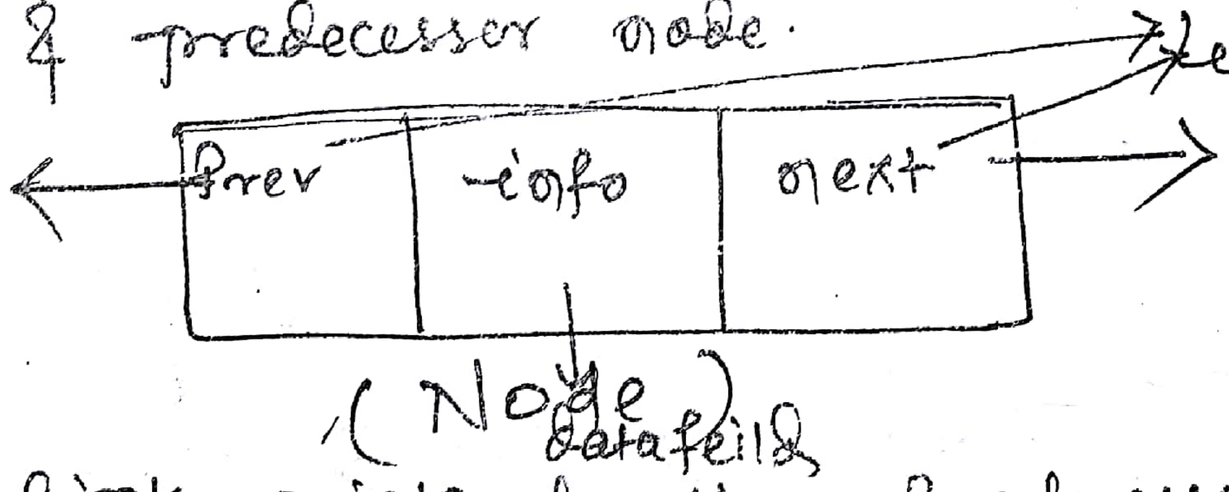
→ There are two external pointer start & last are used where start is used to store address of starting node & last is used to store address of last node in the list.



→ Previous field of first node & next field of last node holds NULL.

→ It provides bidirectional traversing.

→ Each node in a doubly linked list has two linking fields. These are used to point to the sucessor node & predecessor node.



→ left link points to the predecessor node where as right link points to the sucessor node.

» Structure for double linked list :-

```
typedef struct NODE
{ struct NODE *prev;
  int info;
  struct NODE *next;
} node;
```

» Empty list creation :-

```
node *start = NULL, *last = NULL;
```

operation in double linked list :-

1) Traverse
   - Traverse in order
   - Traverse in reverse order
2) creation of double linked list.
3) insertion
4) deletion
5) searching

Traverse :-

(i) order :-

```
void traverse_in_or( )
{
 node *ptr;
 ptr = start;
 while(ptr != NULL)
  {pf("%d", Ptr->info);
   ptr = ptr->next;
  }
}
```

(ii) reverse order :-

```
void traverse_in_re()
{
  node *ptr;
  ptr = last;
  while (ptr != NULL)
   {
   pf("%d", Ptr->info);
   ptr = ptr->prev;
   }
}
```

ii) Creation :-

```
void create( )
 {node *ptr;
  ptr = (node *) malloc (sizeof(node));
  if(ptr == NULL)
   {
   {pf("Memory full");
    getch();
   exit(0);
  }
}
else{ printf("Enter info");
    scanf("%d", &ptr->info);
    if(start==NULL)
      {
      ptr->next = NULL;
       ptr->prev = NULL;
```

```
start = ptr;
last = ptr;
}
else
    {
      ptr -> next = NULL;
      last -> next = ptr;
      last = ptr;
    }
}
```

Insertion :-

a) insertion at beginning

c code

```
void dl_ins_beg()
{
node *ptr;
ptr = (node *) malloc (sizeof(node));
if(ptr==NULL)
 {
  pf("Memory full");
  getch();
  exit(0);
 }
else
  { pf("Enter info");
  sf("%d", &ptr->info);
if(start==NULL)
 {
  ptr->prev = NULL;
    ptr->next = NULL;
 start=ptr;
 last=ptr;
 }
```

```
else
  {
    ptr->next=start;
    ptr->prev=NULL;
    start->prev=ptr;
     start=ptr;
  }
}
}
```

ii) insertion at end :-

```
void dl_ins_end()
 {node *ptr;
  ptr=(node *)malloc(sizeof(node));
  if(ptr==NULL)
    {printf("Memory full");
     getch();
     exit(0);
    }
  else
    {printf("Enter info");
     scanf("%d",&ptr->info);
     ptr->next=NULL;
     if(start==NULL)
       {ptr->prev=NULL;
        start=ptr;
         last=ptr;
       }
     else
       {ptr->prev=last;
        last->next=ptr;
        last=ptr;
       }
    }
}
```

4

iii) Insertion at specific position :-

```
void dl_ins_spc()
{ int pos, i;
  node *ptr, *loc;
  ptr = (node *) malloc(sizeof(node));
  if(ptr == NULL)
  {
   printf("Memory full");
    getch();
    exit(0);
  }
else
{
 pf("Enter pos");
  sf("%d", &pos);
  sf("%d", &ptr->info);
  if(pos == 0)
   {
        ptr->prev = NULL;
        ptr->next = NULL;
         start = ptr;
         last = ptr;
    }
else
 { loc = start;
  for(i=0; i<pos; i++)
   { loc = loc->next;
     if(loc == NULL)
      {pf("Pos Not found");
       getch();
        exit(0);
    }
  }

 ptr->next = loc
 ptr->prev = loc->prev;
(loc->prev)->next = ptr;
 loc->prev = ptr;
}}
```

9) insertion before —:

```
void dl_ins_bef()
{ int befel;
{ node *ptr, *bep;
  ptr=(node *)malloc(sizeof(node));
  if(ptr==NULL)
    {
    { printf("Memory fell");
      getch();
      exit(0);
    }
else
  {
  { printf("Enter element");
    scanf("%d", &Ptr->info);
   printf("Enter the element before
          which element to be inserted");
scanf("%d", &befel);
   loc=start;
 if(start->info==befel)
    {
      loc->prev=ptr;
      ptr->next=start;
      ptr->prev=NULL;
      start=ptr;
    }
else
    {while((loc!=NULL)&&(loc->info!=befel))
       loc=loc->next;
   if(loc==NULL)
     {
      {pf("element Not found");
       getch();
       exit(0);
     }
else
     {
      {ptr->next=loc;
      ptr->prev=loc->prev;
      (loc->prev)->next=ptr;
       loc->prev=ptr;
}
}
}
```

23

v) Insert after an element :-

```
void al_ins_aft()
{ int aft;
 node *ptr,*loc;
 ptr = (node *) malloc(sizeof(node));
 if(ptr==NULL)
 {
  printf("memory full");
   getch();
   exit(0);
 }
else
 {
  printf(" after element");
   scanf("%d",&aft);
  printf("Enter value");
   scanf("%d",&ptr->info);
  if(last->info == after)
     {
       ptr->prev = last;
       ptr->next = NULL;
       last->next =ptr;
       last=ptr;
     }
else
     loc = start;
  while((loc!=NULL)&&(loc->info != aft))
     {
       loc = loc->next;
     }
  if(loc==NULL)
   pf("element Not present");
  else
   {ptr->prev = loc;
    ptr->next = loc->next;
    (loc->next)->prev = ptr;
     loc->next =ptr;
   }
 }
}
```

Deletion :-
deletion at beginning :-

```
void del_beg()
{ node *ptr;
  ptr = start;
  if(start == NULL)
  { printf("No element");
    getche();
    exit(0);
  }
else
 { printf("Element deleted is %d", ptr->info);
  if( start->next == NULL)
     { last = NULL;
       start = NULL;
     }
  else
     {
       (start->next)->prev = NULL;
        start = ptr->next;
     }
  }
```

Deletion at end :-

```
void del_end()
{
 node *ptr;
  ptr = last;
  if( last == NULL)
   { pf("List is empty");
     getch();
     exit(0);
   }
else
 { pf("element deleted is %d", ptr->info);
  if(last->prev = NULL)
    {
       last = NULL;
       start = NULL;
    }
else
    { last->next = ptr
      (last->prev)->next = NULL;
       last = last->prev;
    }
  free(ptr);
```

iii) Deletion at specific pos :-

```
void del_spec()
{
 node *ptr;
 int pos, i
 if(start==NULL)
  {
  printf("No element");
  getch();
  exit(0);
  }
 else
  { ptr=start;
printf("Enter pos");
scanf("%d", &pos);
if(pos==0),
 { start=1
 (ptr->next)->prev=NULL;
   start = start->next;
 }
else
 {
  ptr=start;
 for(i=0 ; i<pos ; i++)
   {
   ptr = ptr->next;
  if(ptr==NULL)
   {
   pf("pos not found");
   getch();
   exit(0);
   }  if(ptr->next==NULL)
        last=ptr;
 }
(ptr->next)->prev = ptr->prev;
(ptr->prev)->next=ptr->next;   if(ptr->next==NULL)
                                {
                                last=ptr->prev;
                                (ptr->prev)->next=ptr->next;
                                }
                               else
                                {
                                }
free(ptr);
}
}
```

9v) Deletion after an element :-

```
void dl_del_aftr()
{ int aftr;
 node *ptr;
   ptr = start;
 if( start == NULL)
   { pf("List is empty");
     getch();
     exit(0);
   }
else
   {
    pf("Enter after element");
    sf("%d", &aftr);
while(ptr != NULL) && (ptr->info != aftr))
    ptr = ptr->next;
if(ptr == NULL)
  {
  pf("Location Not found");
  }
else
   { loc = ptr->next;
    ptr->next =
     loc = ptr;
     ptr = ptr->next;
     loc->next = ptr->next;
     (ptr->next)->prev = ptr->prev;

   }free(ptr);
}
```

7 Deletion before an element :-

```
void dl-del-bef()
{ int bef
  node *ptr,*loc;
  ptr = start;
 if(start == NULL)
   {
  printf("List is empty");
  getche();
  exit(0);
   }
else
  { loc = start;
  printf("Enter before element");
  scanf("%d",&bef);
  while((loc != NULL)&&(loc->info != bef))
     { ptr = loc;
       loc = loc->next;
     }
if(loc == NULL)
  printf("element Not found");
else
   { loc->prev = ptr->prev;
    (ptr->prev)->next = ptr->next;
   }
  free(ptr);
}
```