Unit 3
Linked list

Dr.N P Karlekar

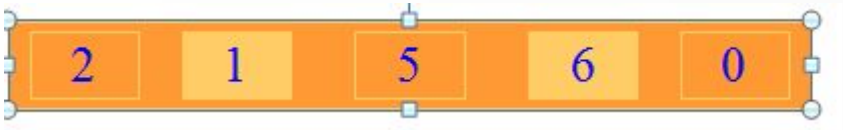
List

What is a List?

It is a sequence of ELEMENTS.

It is a List of elements of type T and it is a FINITE SEQUENCE of elements

Example: li
st integers or list of records.



2	1	5	6	0

Implementation of list

There are 2 main ways :

1. Using Contiguous storage

- an array in which the elements are physically next to one an another in adjacent memory locations

Disadvantage -

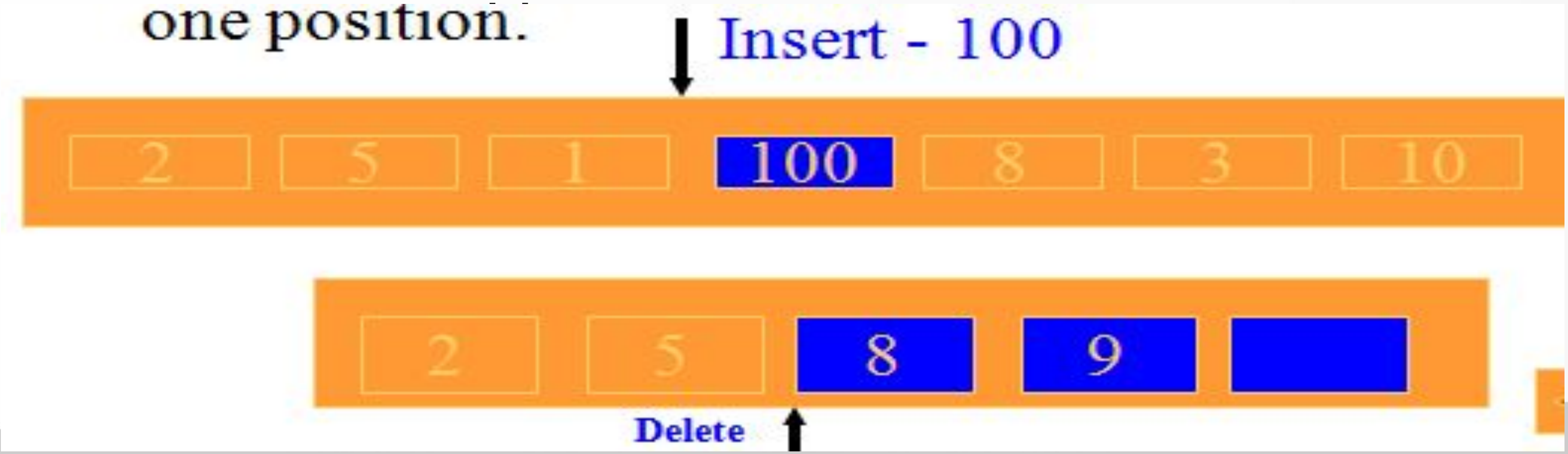
2. Non Contiguous storage(Linked List)

Disadvantage of Contiguous memory

Two Disadvantages,

1) Insertion in Position requires moving of elements ‘DOWN’ one position.

2) Deletion requires moving of elements ‘UP’



one position.

Linked list

Linked list eliminates the problem encountered in List.

What is a Linked List?

A linked list is a collection of **nodes**, where each node contains some data along with information about the next node.

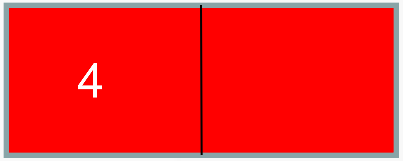
How it works?

A linked list uses non-contiguous memory locations and hence requires each node to remember where the next node is

Cont…

What is a NODE?

Node is a combination of DATA and LINK.



<u>What is Data?</u>

Is the part where the actual data is stored.

<u>What is a Link?</u>

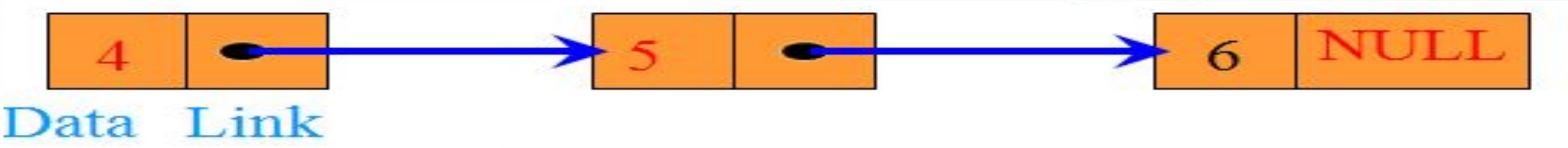
It is the link (pointer) to the next element of the list.

<u>Two ways of implementing</u>

It could be either an index to an array element (array implementation) OR

a pointer variable containing the address of the next element (pointer)

Representation



1) The above linked list contains 3 nodes located at different memory locations each node has a pointer that points to the next node.

2) The node that have a **NULL** indicates the it is the end of the list (**last node**).

In C/C++ programs either **NULL** or **0** can be used to indicate the end of the list.

Array Representation

Memory is to be allocated for

1) elements of the list
2) links.

when the elements are just integers

- int data[4]; // array to hold the data
- int link[4]; // array to hold the links

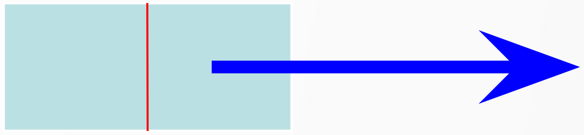
- We can now store a list of up to 4 elements and their links i.e. 4 nodes.

Index	Data	Link
0	**35**	**3**
1	**54**	**2**
2	**86**	**99**
3	**48**	**1**

Pointer Implementation

Consider the following,

```
struct node
{
int data;
node *next;
};
```



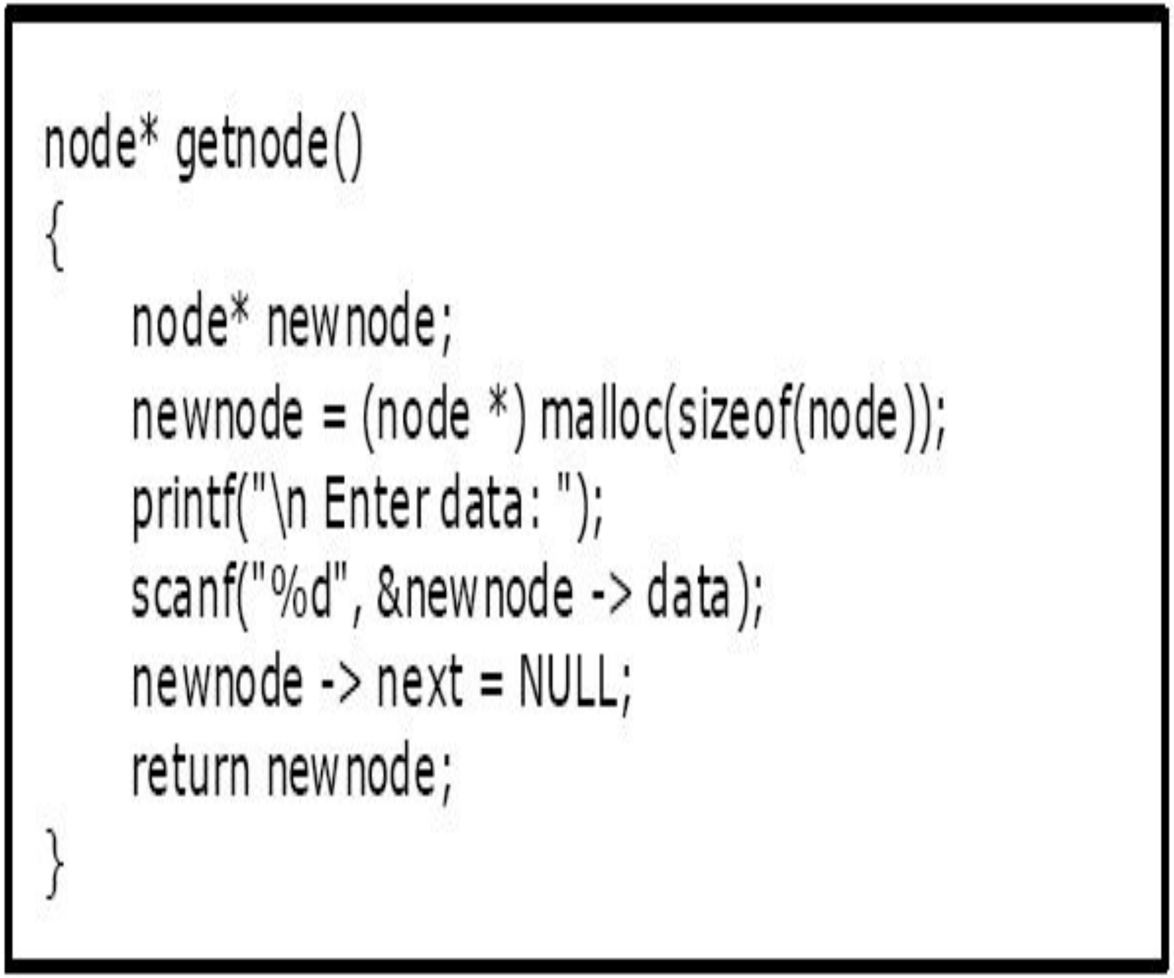
Node

- 1) **The structure contains an data and a pointer**
- **2) Data contains the data**
- **3) next is the pointer to the next node in the list**

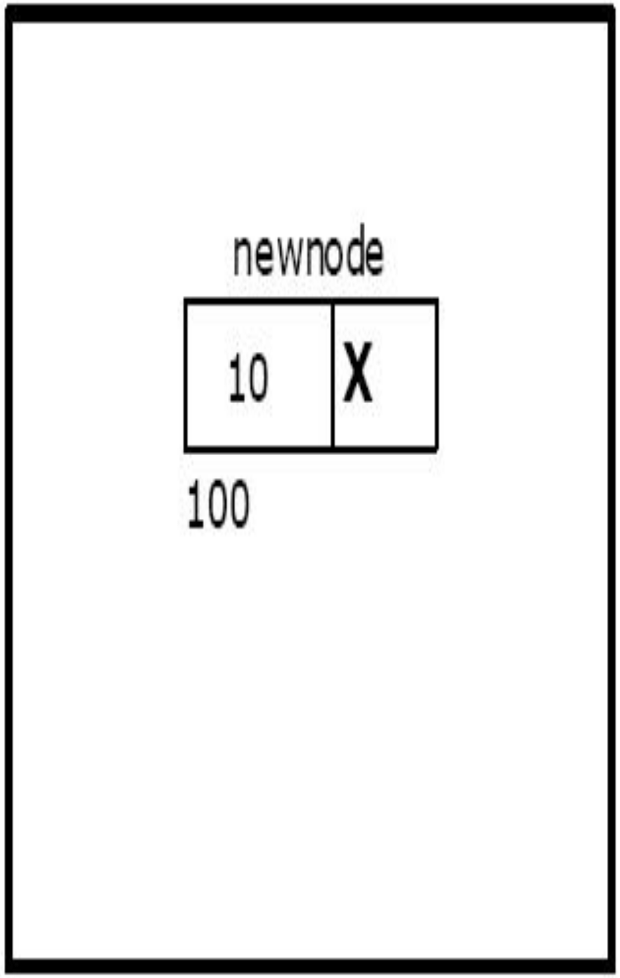
Operations on linked list

- Insertion
- Deletion
- search
- Creation
- Traverse

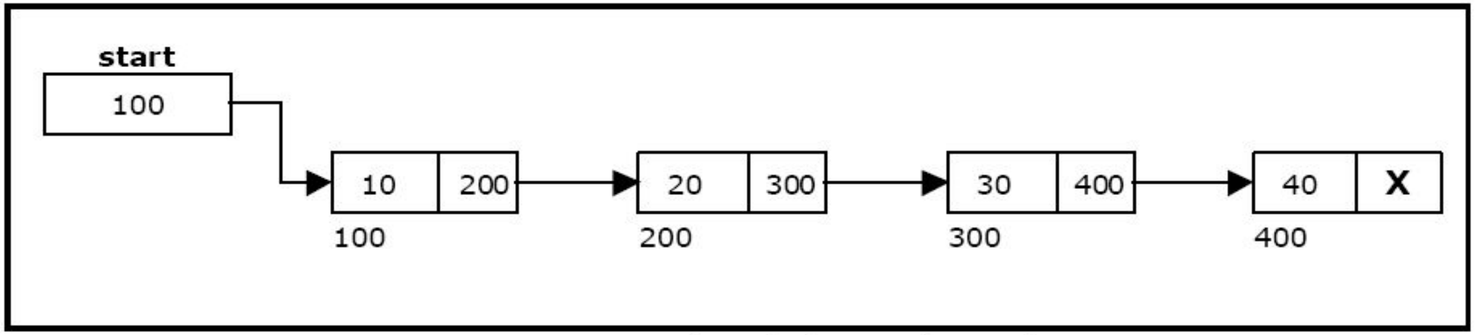
Creation of SLL

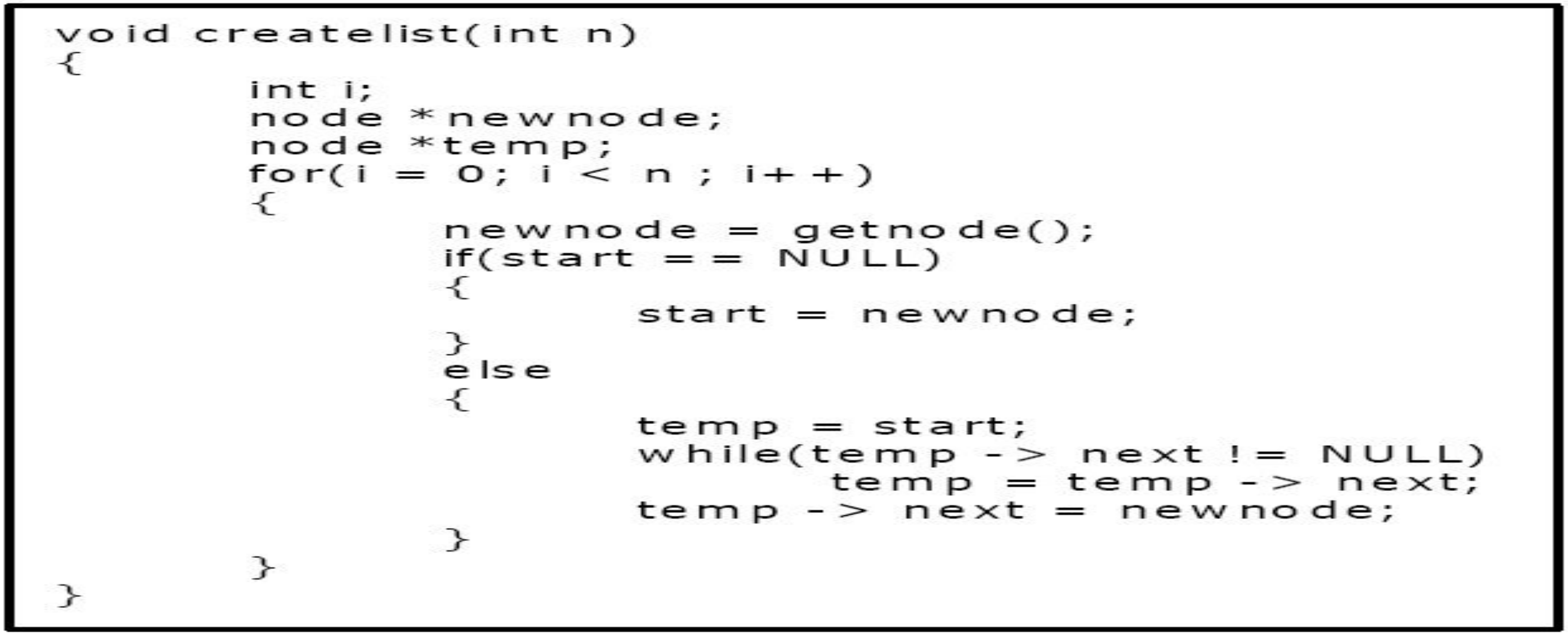


```
node* getnode()
{
      node* newnode;
      newnode = (node *) malloc(sizeof(node));
      printf("\n Enter data: ");
      scanf("%d", &newnode -> data);
      newnode -> next = NULL;
      return newnode;
}
```



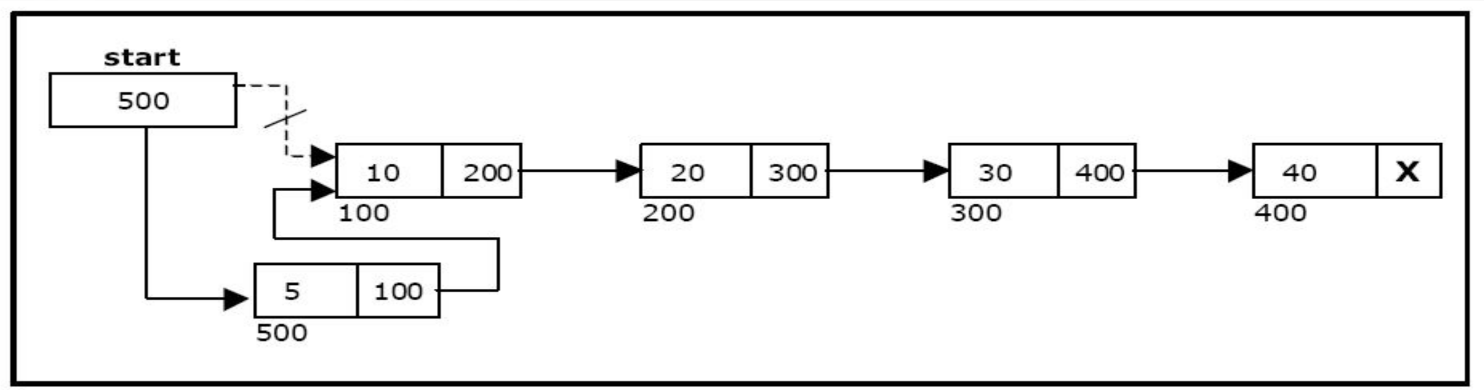
Creation of SLL of ‘n’ nodes

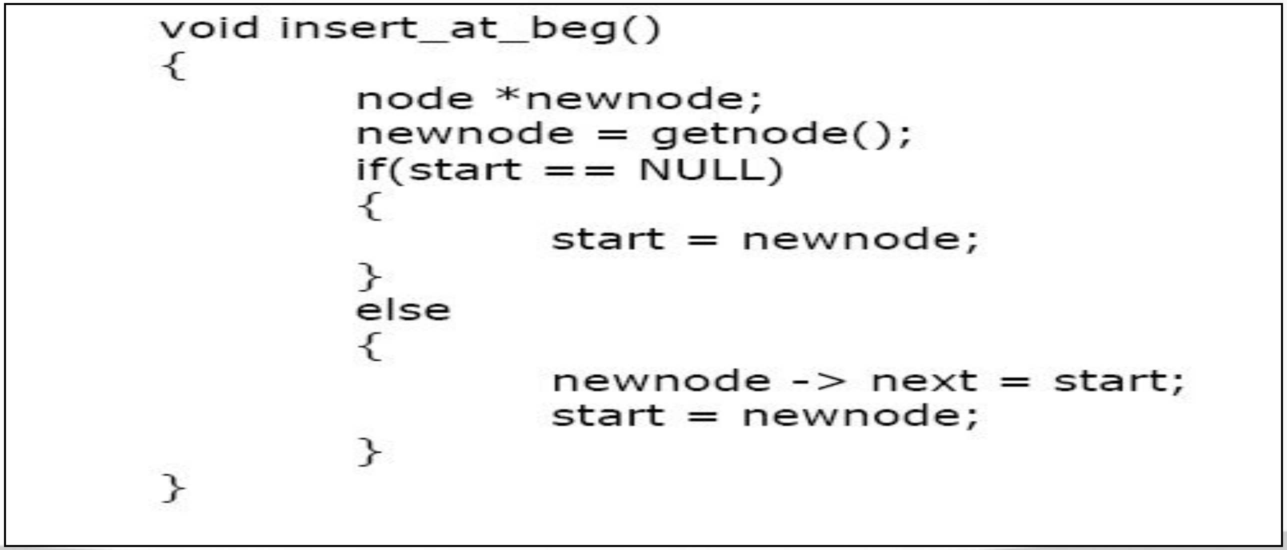




```
void createlist(int n)
{
        int i;
        node *newnode;
        node *temp;
        for(i = 0; i < n ; i++)
        {
                newnode = getnode();
                if(start == NULL)
                {
                        start = newnode;
                }
                else
                {
                        temp = start;
                        while(temp -> next != NULL)
                                temp = temp -> next;
                        temp -> next = newnode;
                }
        }
}
```

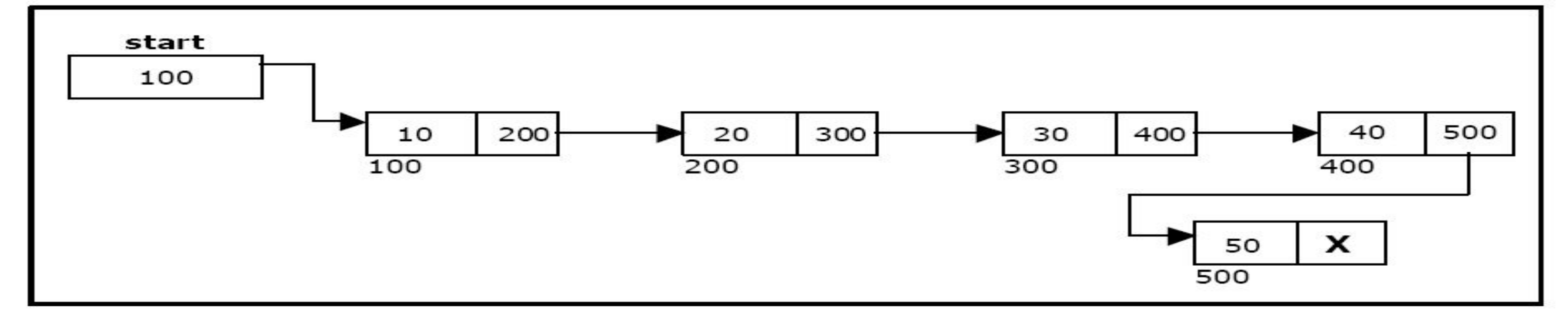
Insertion at beginning in SLL

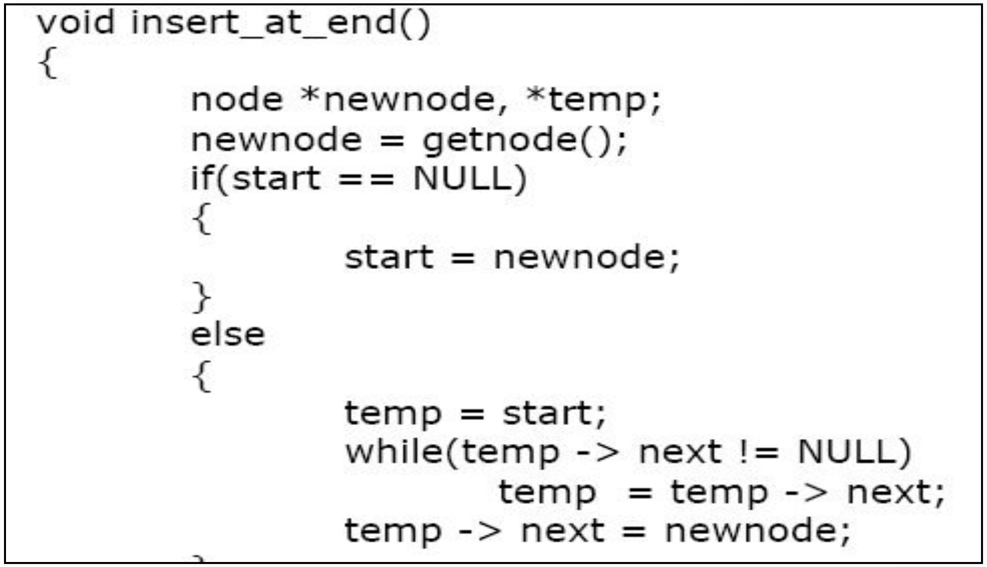




```
void insert_at_beg()
{
        node *newnode;
        newnode = getnode();
        if(start == NULL)
        {
                start = newnode;
        }
        else
        {
                newnode -> next = start;
                start = newnode;
        }
}
```

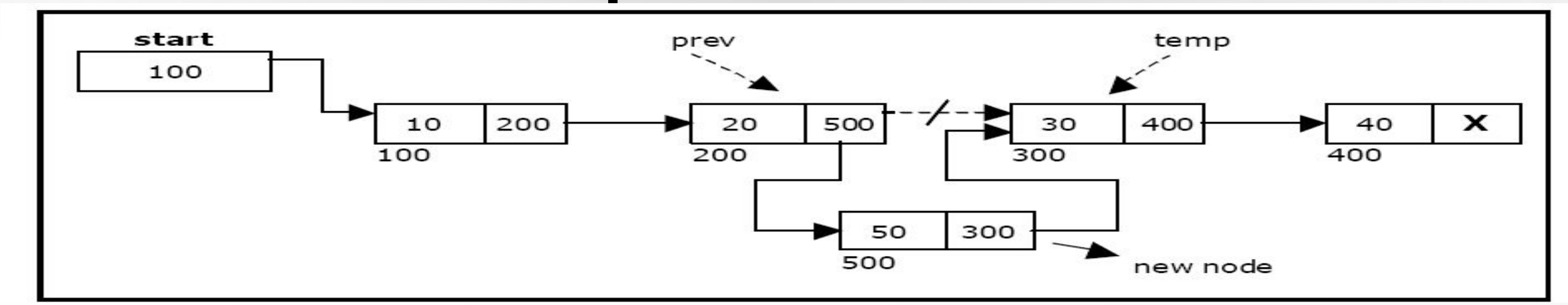
Inserting a node at end in SLL

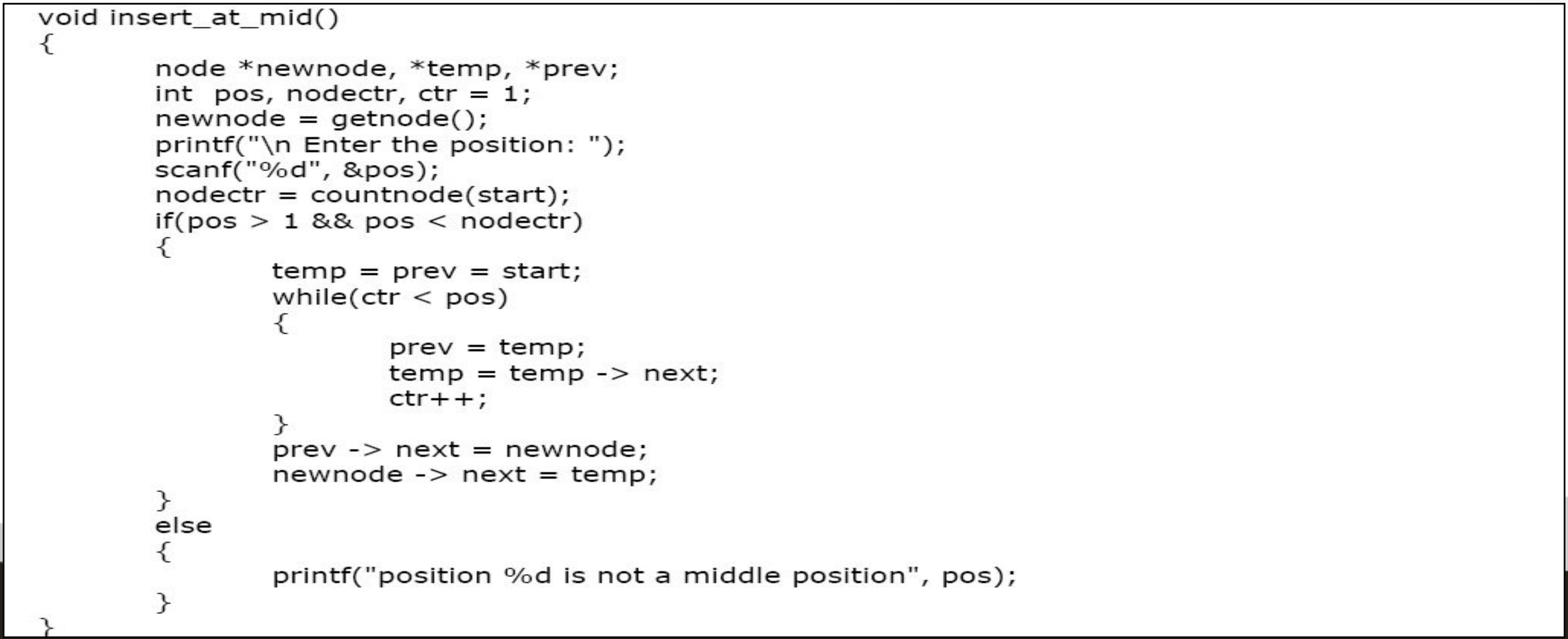




```
void insert_at_end()
{
        node *newnode, *temp;
        newnode = getnode();
        if(start == NULL)
        {
                start = newnode;
        }
        else
        {
                temp = start;
                while(temp -> next != NULL)
                        temp  = temp -> next;
                temp -> next = newnode;
```

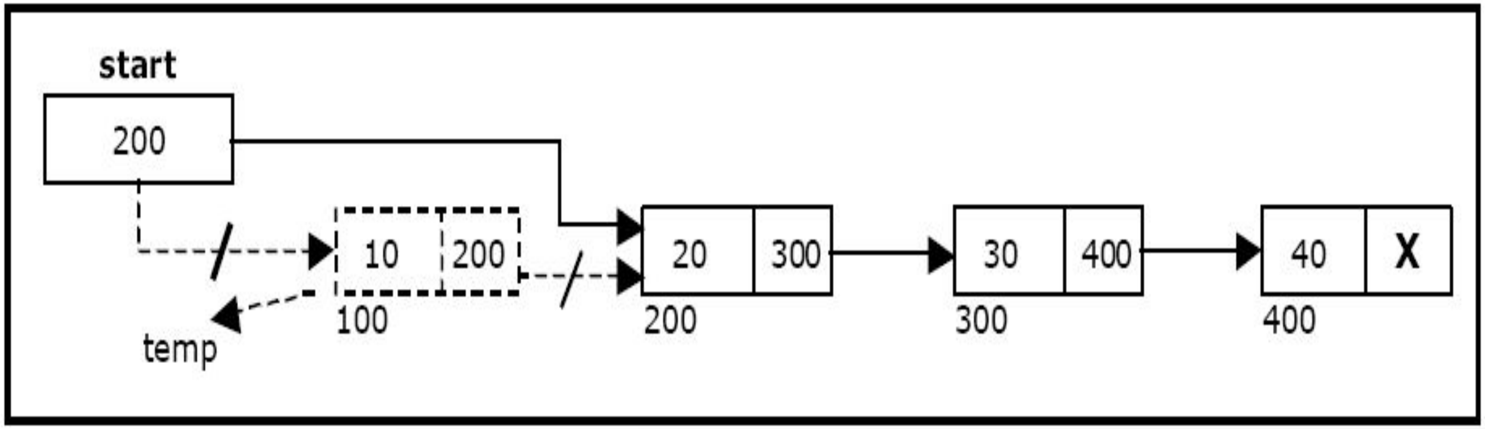
Inserting a node at intermediate position

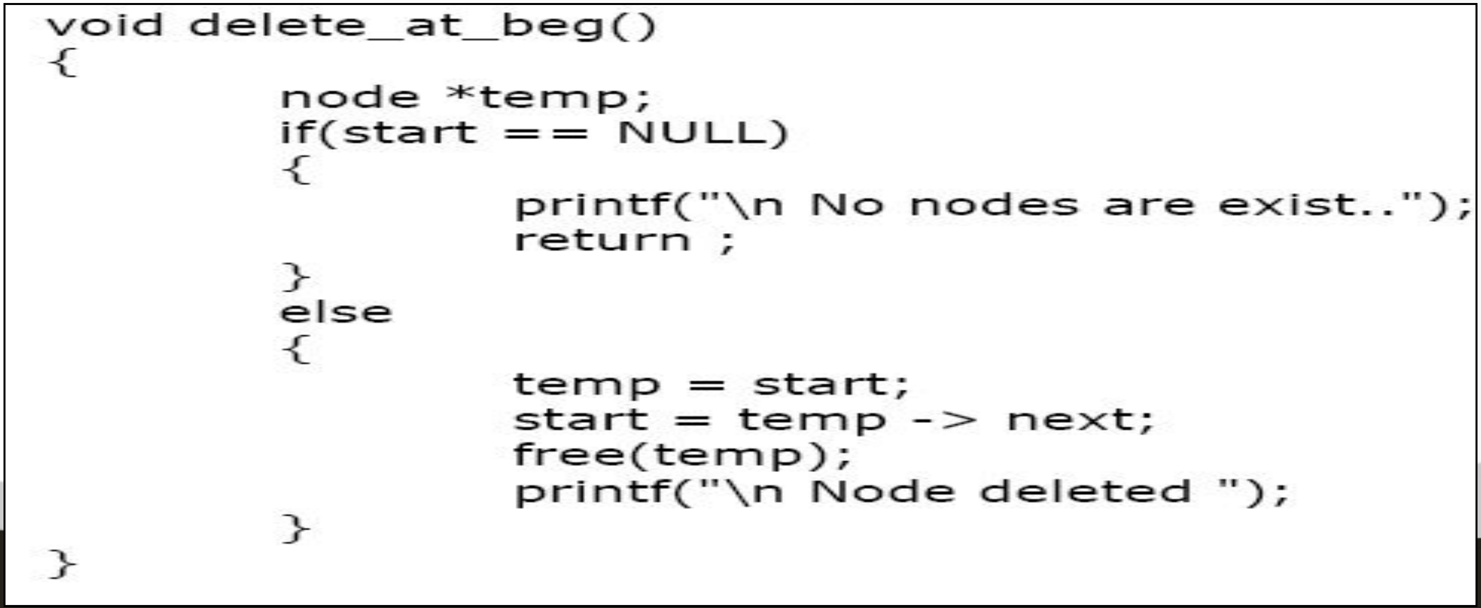




```
void insert_at_mid()
{
        node *newnode, *temp, *prev;
        int  pos, nodectr, ctr = 1;
        newnode = getnode();
        printf("\n Enter the position: ");
        scanf("%d", &pos);
        nodectr = countnode(start);
        if(pos > 1 && pos < nodectr)
        {
                temp = prev = start;
                while(ctr < pos)
                {
                        prev = temp;
                        temp = temp -> next;
                        ctr++;
                }
                prev -> next = newnode;
                newnode -> next = temp;
        }
        else
        {
                printf("position %d is not a middle position", pos);
        }
}
```

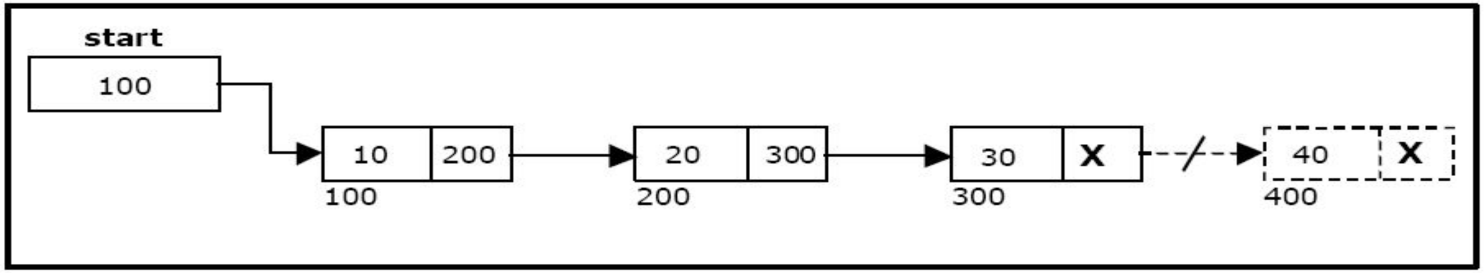
Deletion of node at the begining

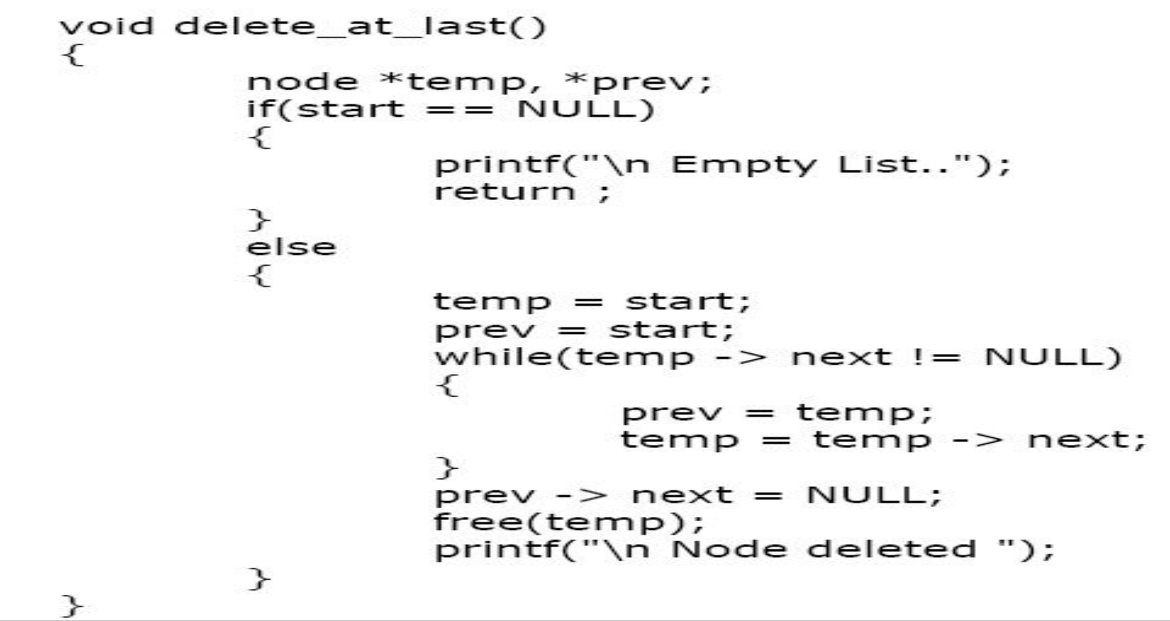




```
void delete_at_beg()
{
          node *temp;
          if(start == NULL)
          {
                    printf("\n No nodes are exist..");
                    return ;
          }
          else
          {
                    temp = start;
                    start = temp -> next;
                    free(temp);
                    printf("\n Node deleted ");
          }
}
```

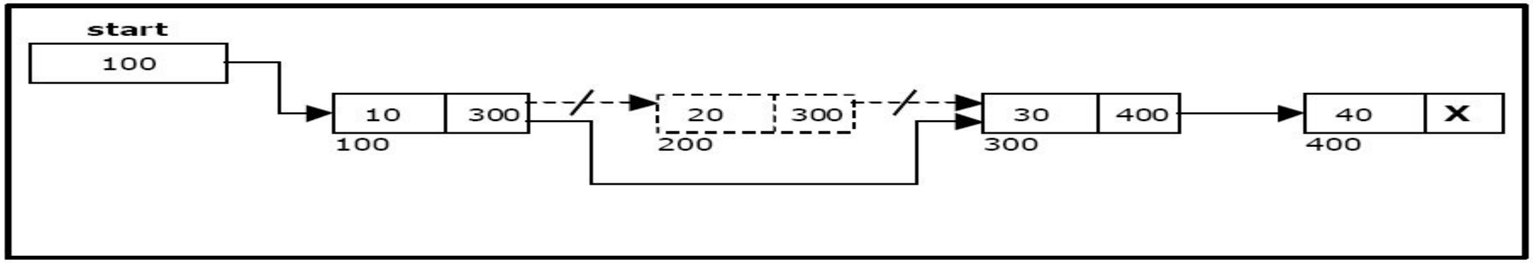
Deletion of node at end





```
void delete_at_last()
{
        node *temp, *prev;
        if(start == NULL)
        {
                printf("\n Empty List..");
                return ;
        }
        else
        {
                temp = start;
                prev = start;
                while(temp -> next != NULL)
                {
                        prev = temp;
                        temp = temp -> next;
                }
                prev -> next = NULL;
                free(temp);
                printf("\n Node deleted ");
        }
}
```

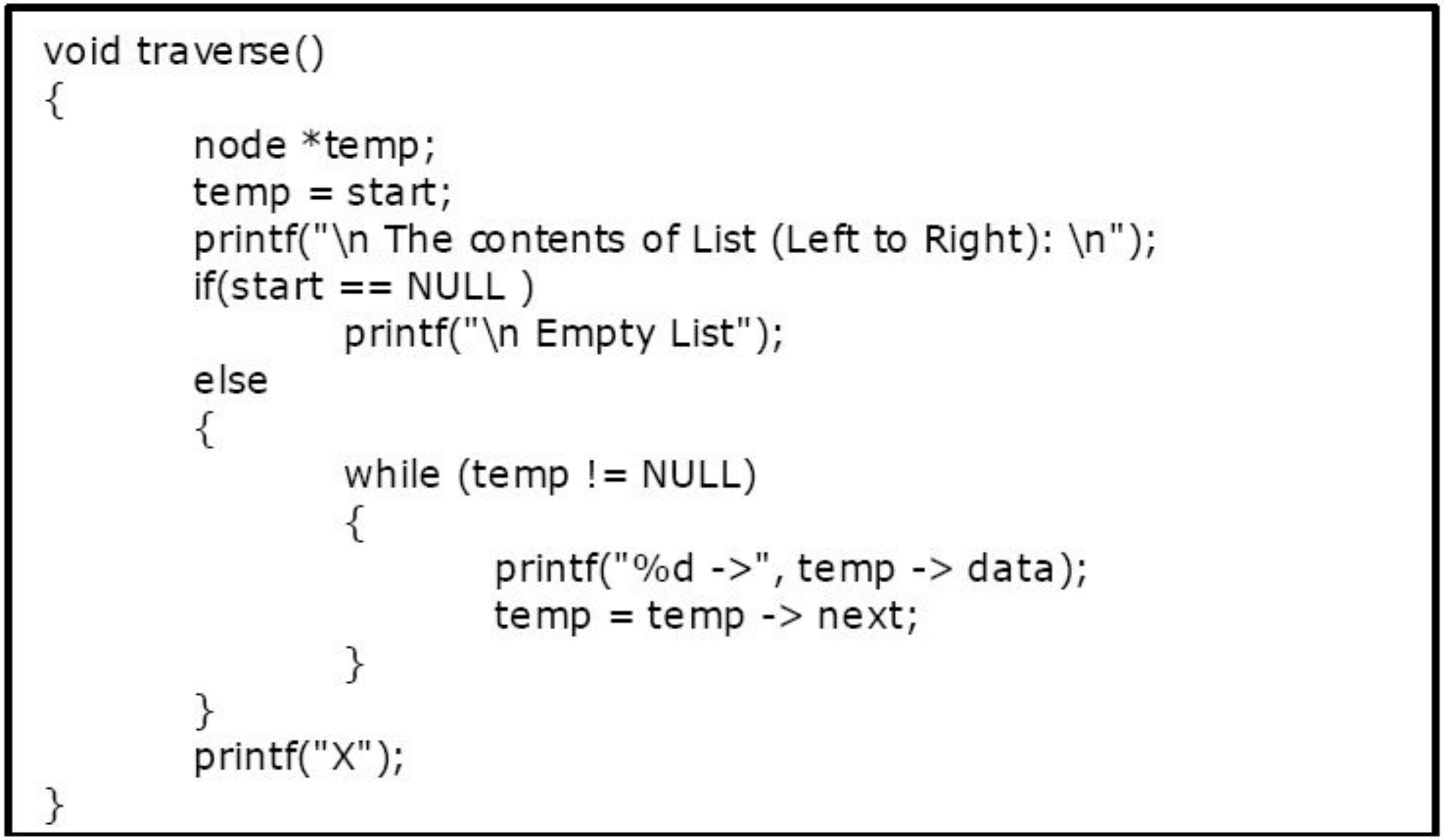
deleting a node at intermediate position



```
void delete_at_mid()
{
        int ctr = 1, pos, nodectr;
        node *temp, *prev;
        if(start == NULL)
        {
                printf("\n Empty List..");
                return ;
        }
        else
        {
                printf("\n Enter position of node to delete: ");
                scanf("%d", &pos);
                nodectr = countnode(start);
                if(pos > nodectr)
                {
                        printf("\nThis node doesnot exist");
                }
```

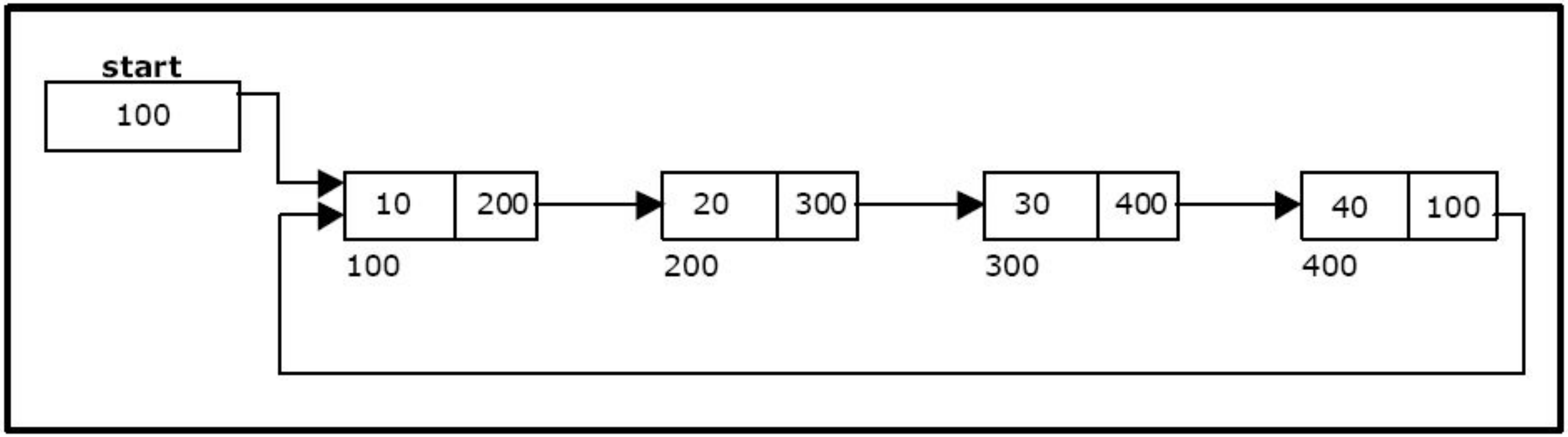
```
                        if(pos > 1 && pos < nodectr)
                        {
                                temp = prev = start;
                                while(ctr < pos)
                                {
                                        prev = temp;
                                        temp = temp -> next;
                                        ctr ++;
                                }
                                prev -> next = temp -> next;
                                free(temp);
                                printf("\n Node deleted..");
                        }
                        else
                        {
                                printf("\n Invalid position..");
                                getch();
                        }

                }
        }
```

Traversing a list

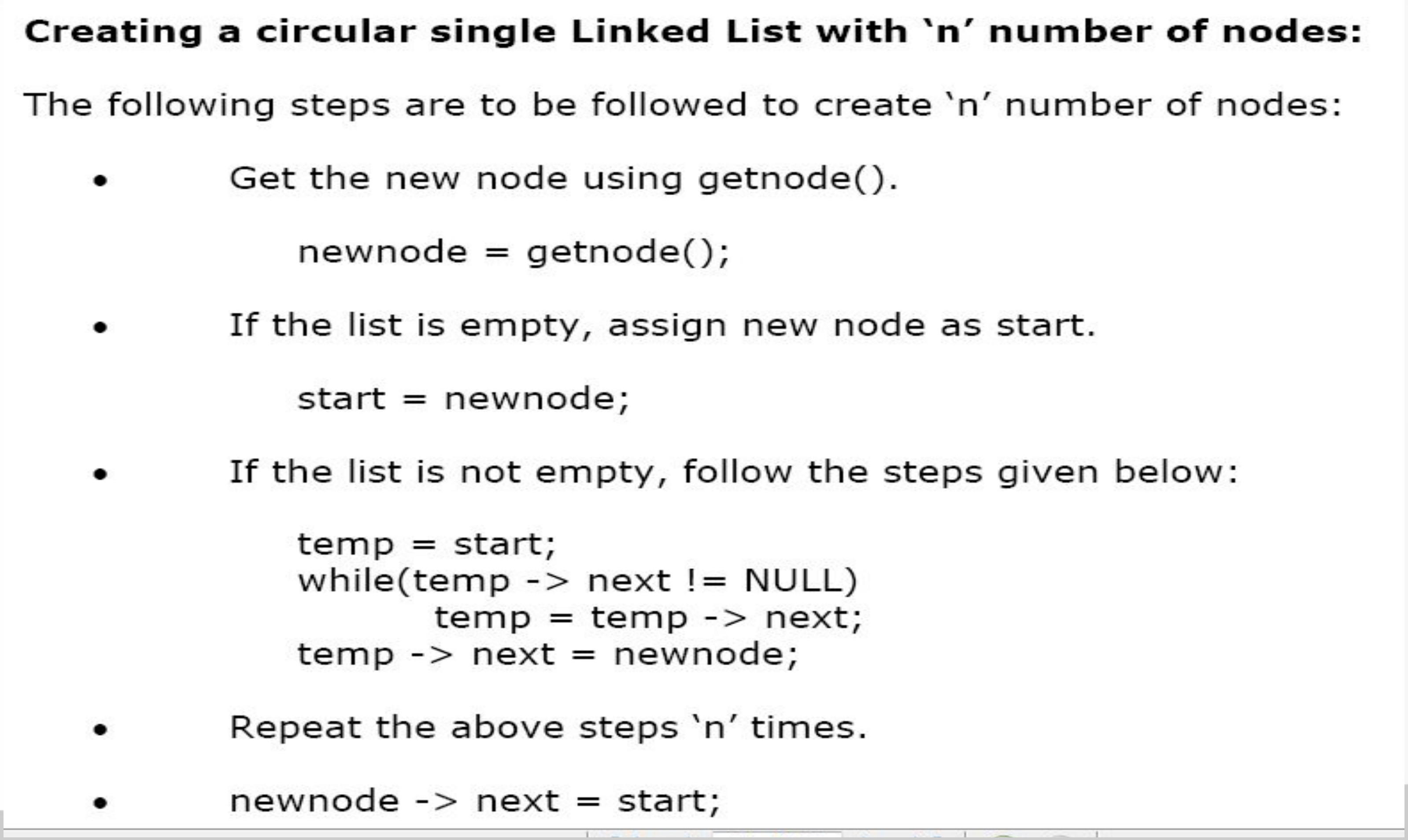


```
void traverse()
{
        node *temp;
        temp = start;
        printf("\n The contents of List (Left to Right): \n");
        if(start == NULL )
                printf("\n Empty List");
        else
        {
                while (temp != NULL)
                {
                        printf("%d ->", temp -> data);
                        temp = temp -> next;
                }
        }
        printf("X");
}
```

Circular singly linked list



Creation of CLL



Creating a circular single Linked List with 'n' number of nodes:

The following steps are to be followed to create 'n' number of nodes:

- Get the new node using getnode().

```
newnode = getnode();
```

- If the list is empty, assign new node as start.

```
start = newnode;
```

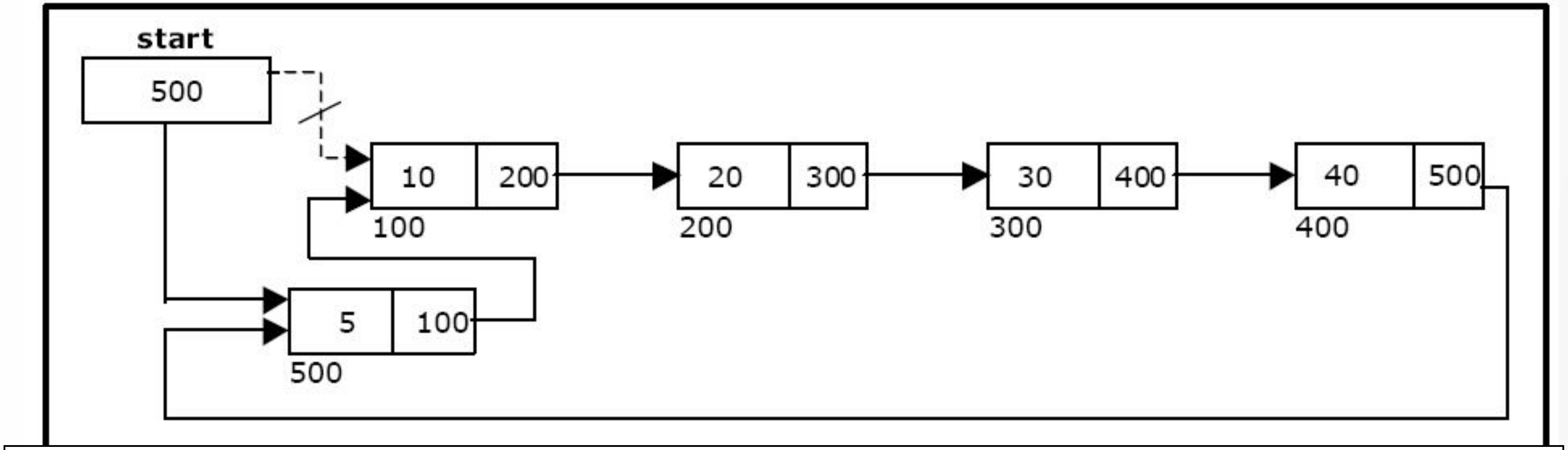
- If the list is not empty, follow the steps given below:

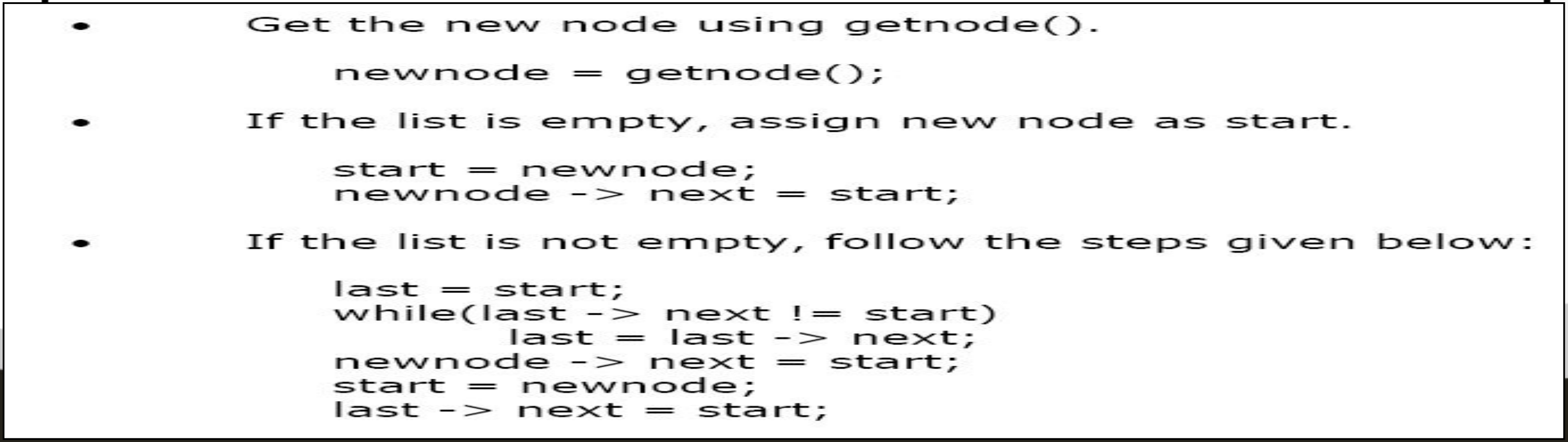
```
temp = start;
while(temp -> next != NULL)
        temp = temp -> next;
temp -> next = newnode;
```

- Repeat the above steps 'n' times.

- newnode -> next = start;

Insertion at the beginning





- Get the new node using getnode().

```
newnode = getnode();
```

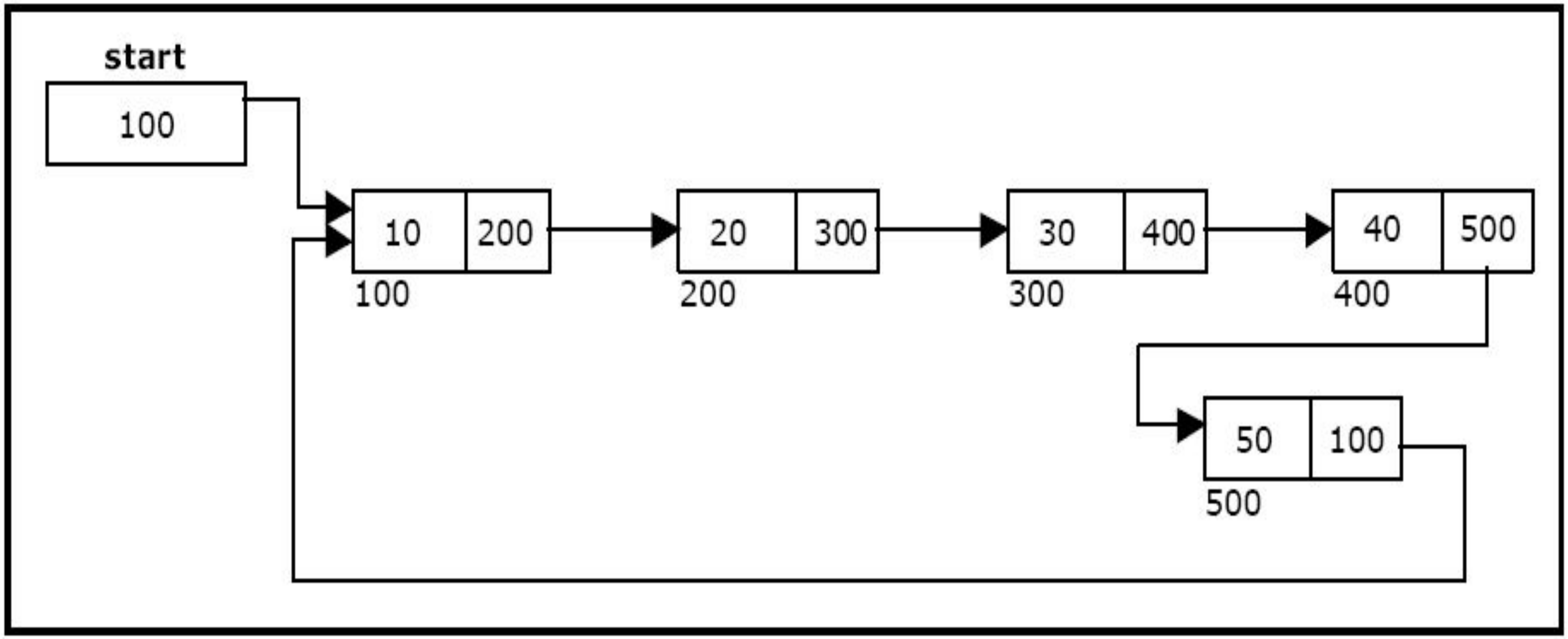
- If the list is empty, assign new node as start.

```
start = newnode;
newnode -> next = start;
```

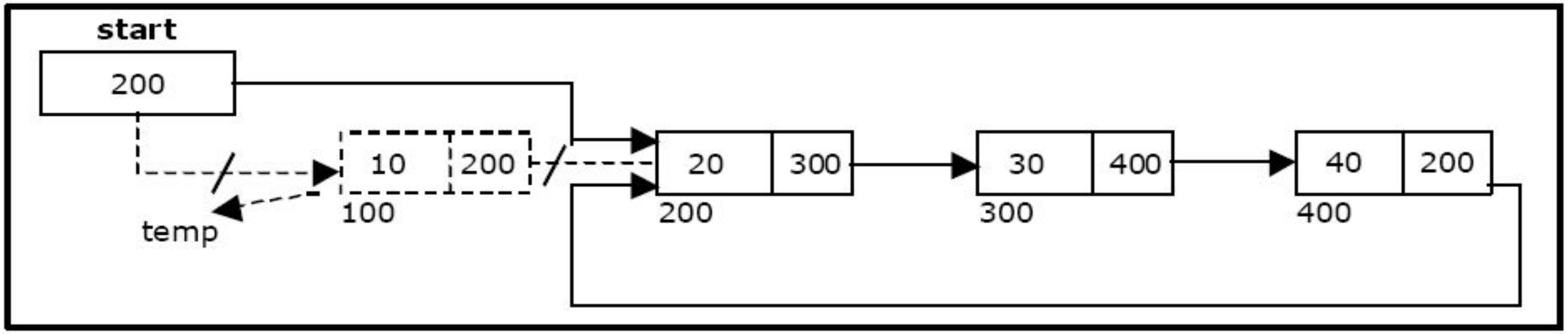
- If the list is not empty, follow the steps given below:

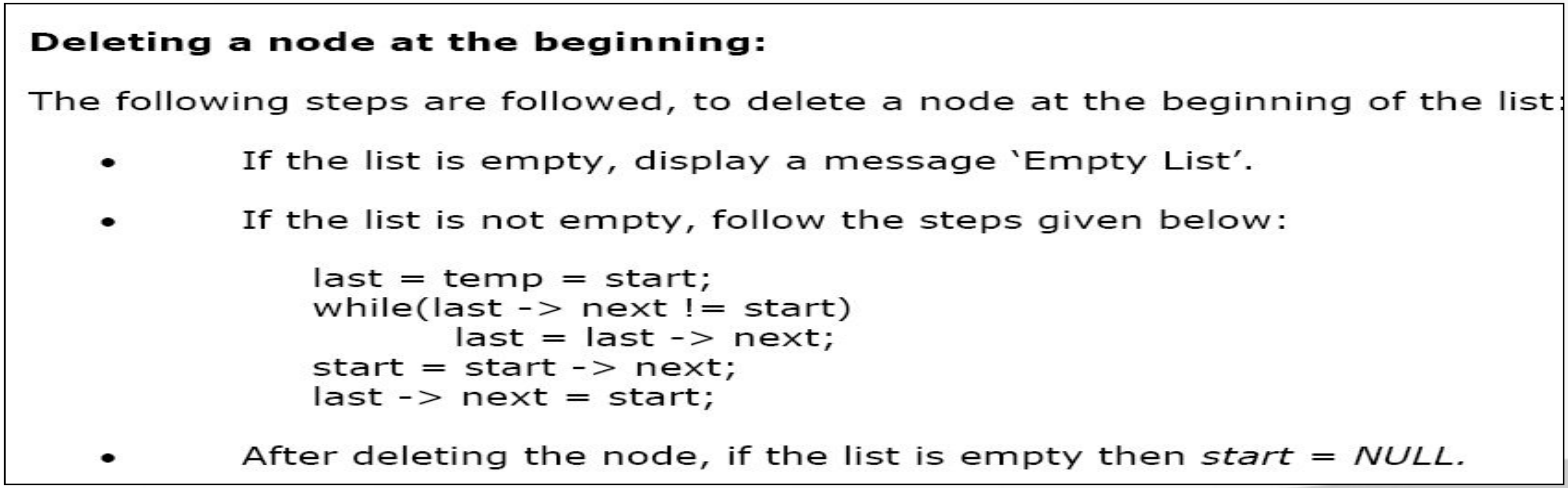
```
last = start;
while(last -> next != start)
        last = last -> next;
newnode -> next = start;
start = newnode;
last -> next = start;
```

Inserting a node at the end



Deletion of node at the beginning in CLL





Deleting a node at the beginning:

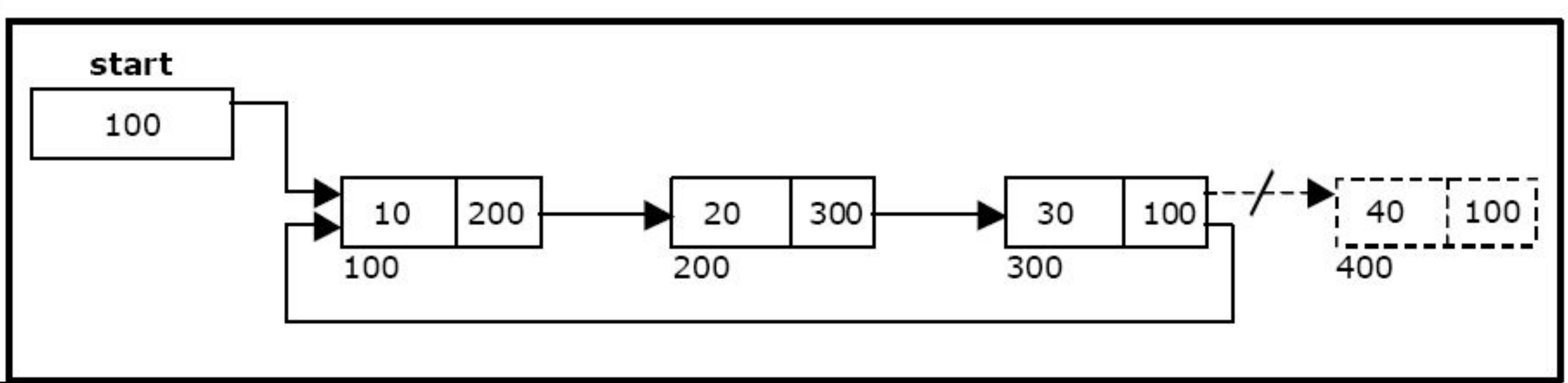
The following steps are followed, to delete a node at the beginning of the list:

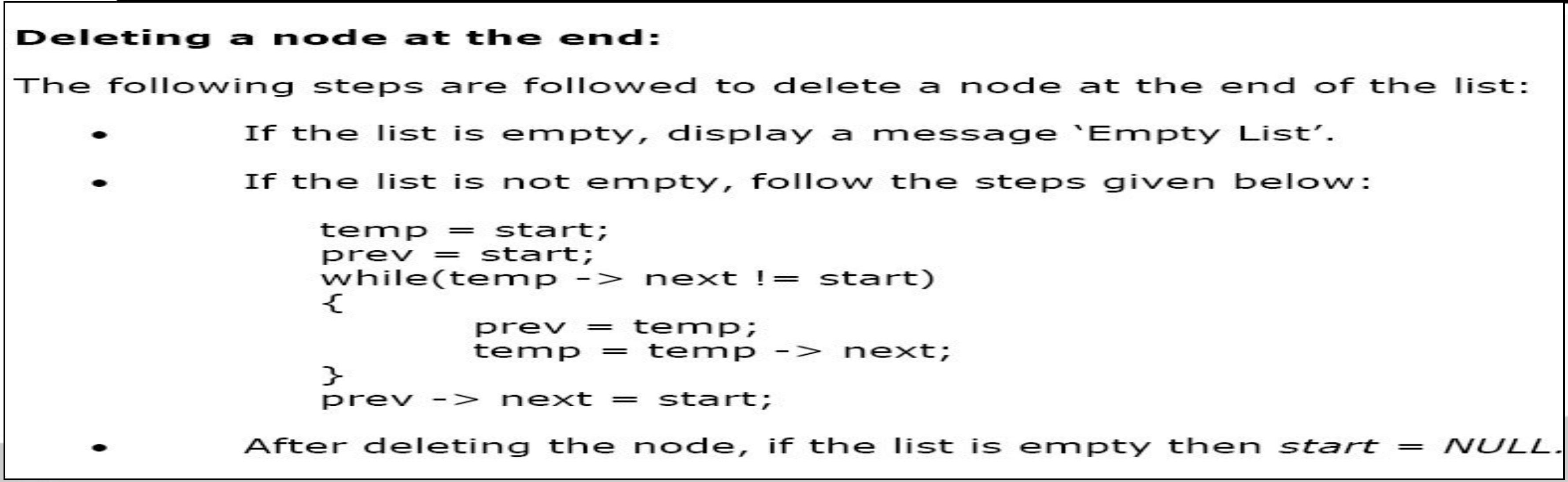
- If the list is empty, display a message 'Empty List'.
- If the list is not empty, follow the steps given below:

```
last = temp = start;
while(last -> next != start)
        last = last -> next;
start = start -> next;
last -> next = start;
```

- After deleting the node, if the list is empty then *start = NULL.*

Deleting a node at the end inCLL





Deleting a node at the end:

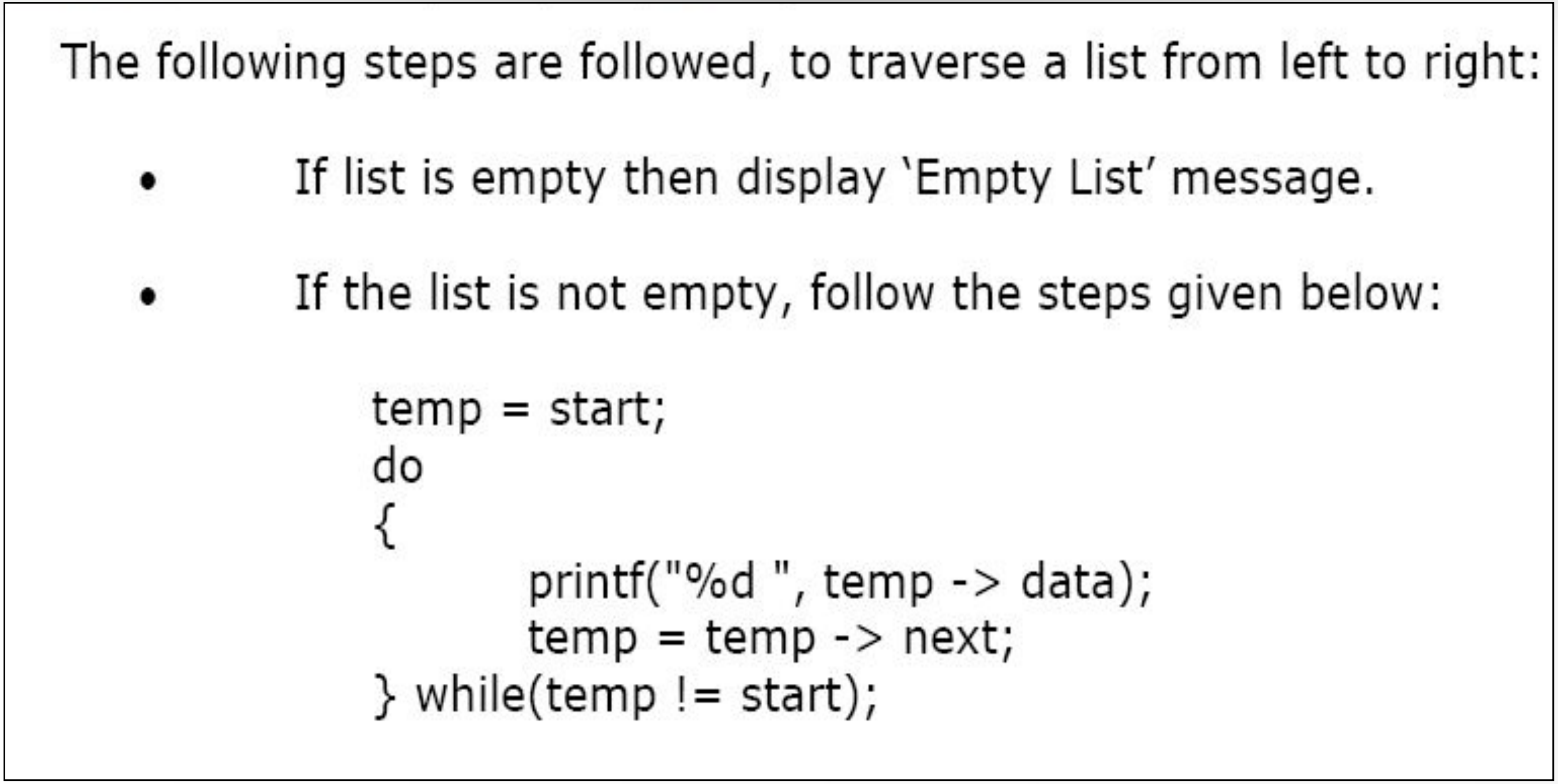
The following steps are followed to delete a node at the end of the list:

- If the list is empty, display a message 'Empty List'.
- If the list is not empty, follow the steps given below:

```
temp = start;
prev = start;
while(temp -> next != start)
{
          prev = temp;
          temp = temp -> next;
}
prev -> next = start;
```

- After deleting the node, if the list is empty then *start = NULL*.

Traversing CLL



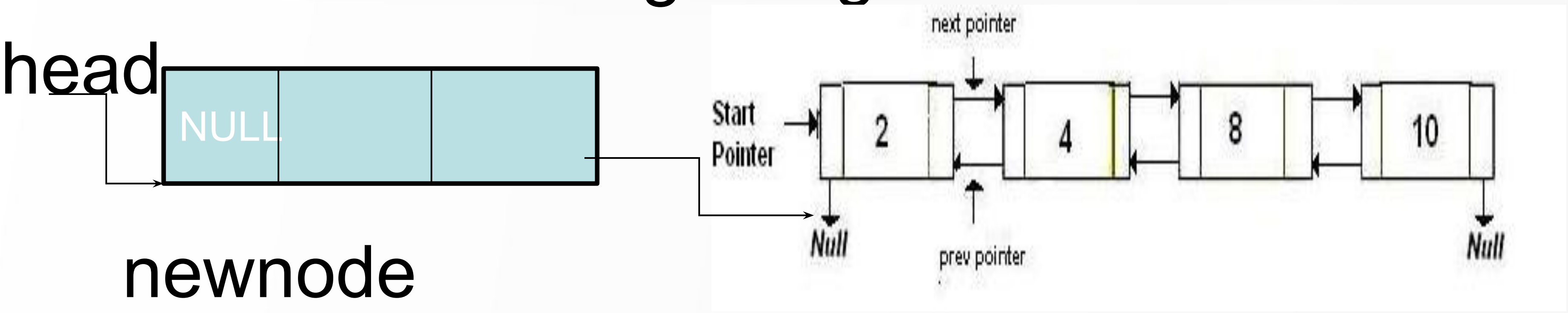
The following steps are followed, to traverse a list from left to right:

- If list is empty then display ‘Empty List’ message.
- If the list is not empty, follow the steps given below:

```
temp = start;
do
{
        printf("%d ", temp -> data);
        temp = temp -> next;
} while(temp != start);
```

Operations on DLL

- Insertion at the beginning



head

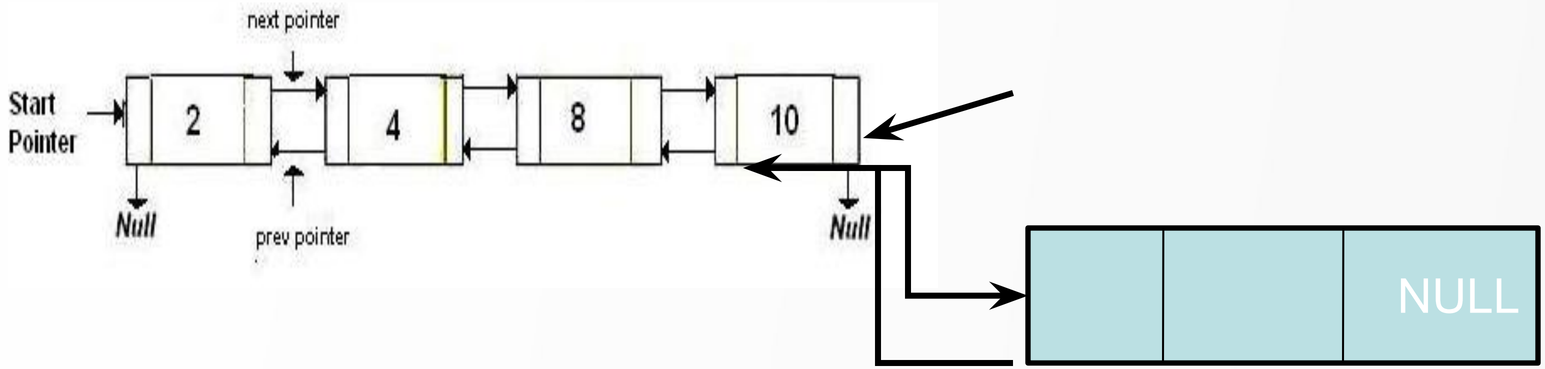
NULL

newnode

```
newnode->next=head;
head->prev=newnode;
head=newnode;
```

DLL

- Insertion at the end



newnode

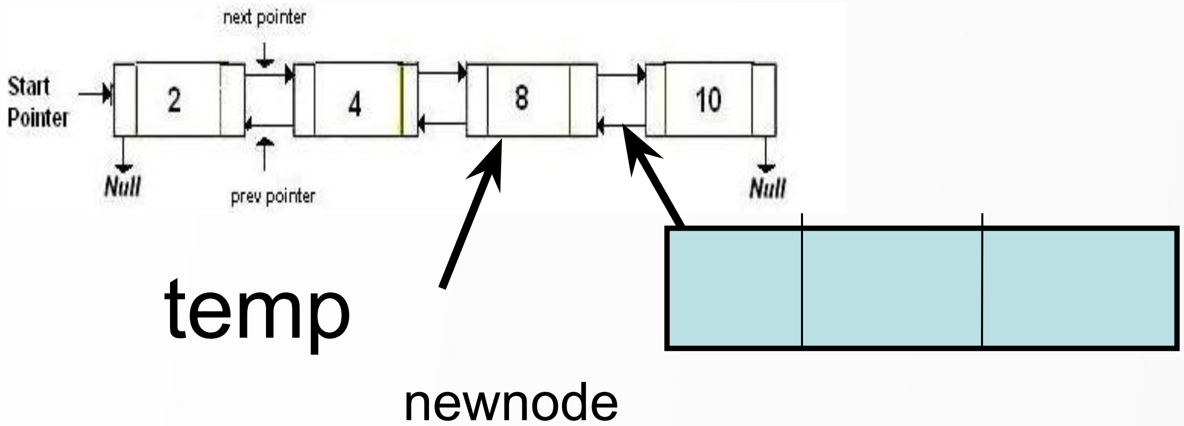
temp->next=newnode

newnode->prev=temp;

newnode->next= NULL;

DLL

- Insertion in between

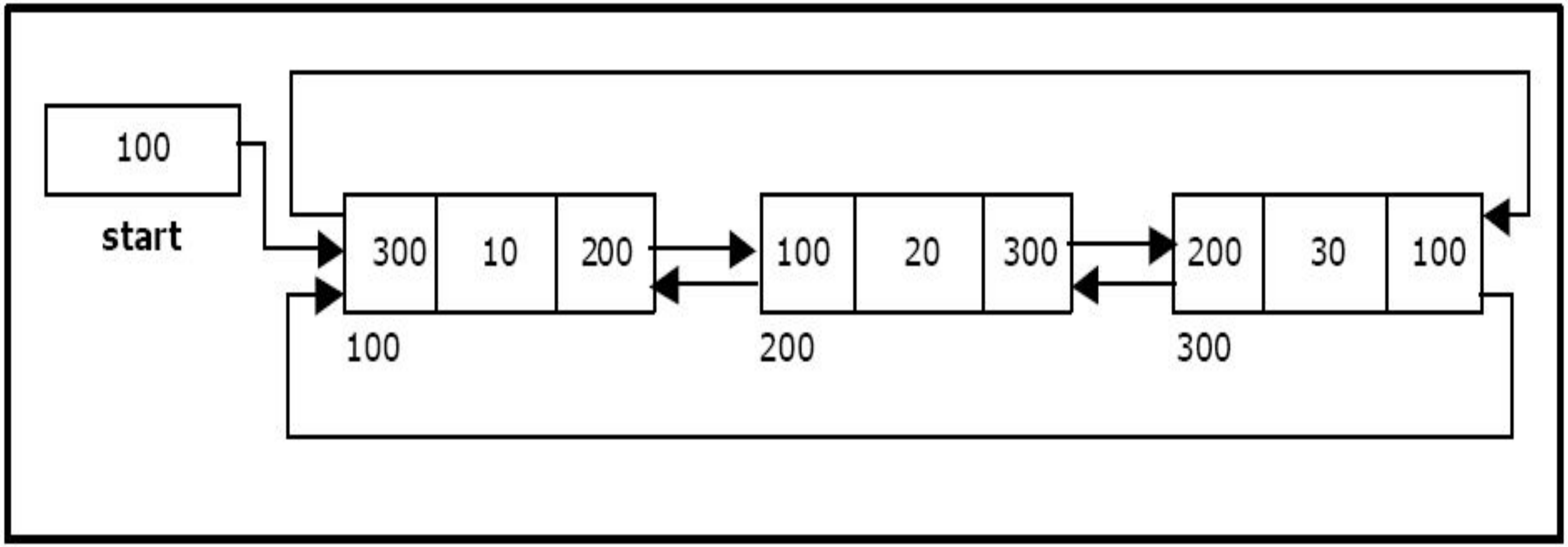


temp

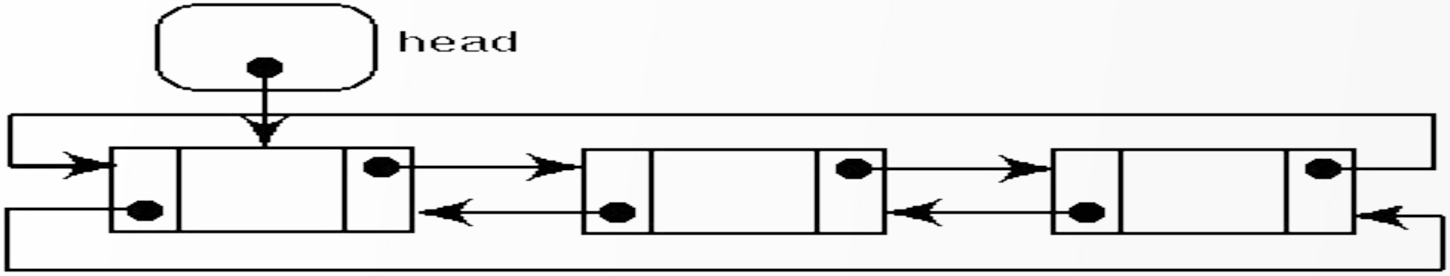
newnode

```
newnode->next=temp->next;
newnode->prev=temp;
temp->next->prev=newnode;
temp->next=newnode;
```

Circular Doubly Linked list



CDLL

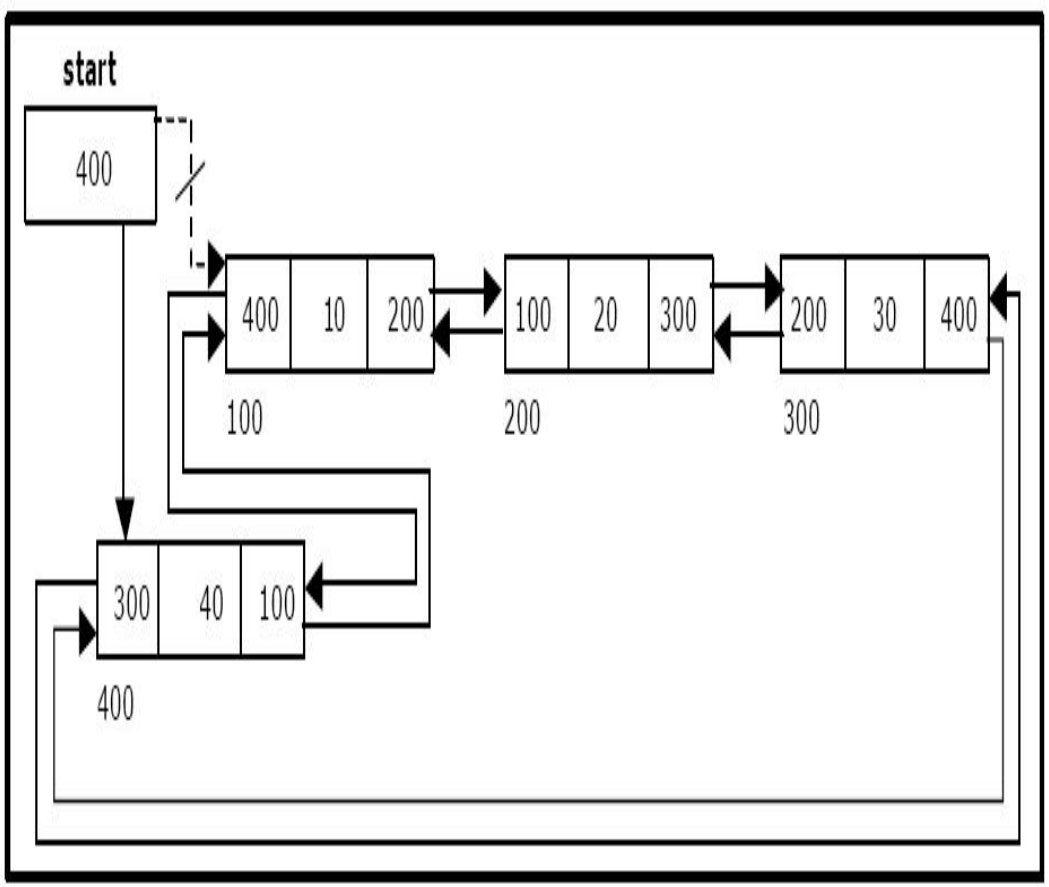


Doubly Linked Circular list

```
newnode->data=data;
newnode->next=NULL;
newnode->prev=NULL;
```

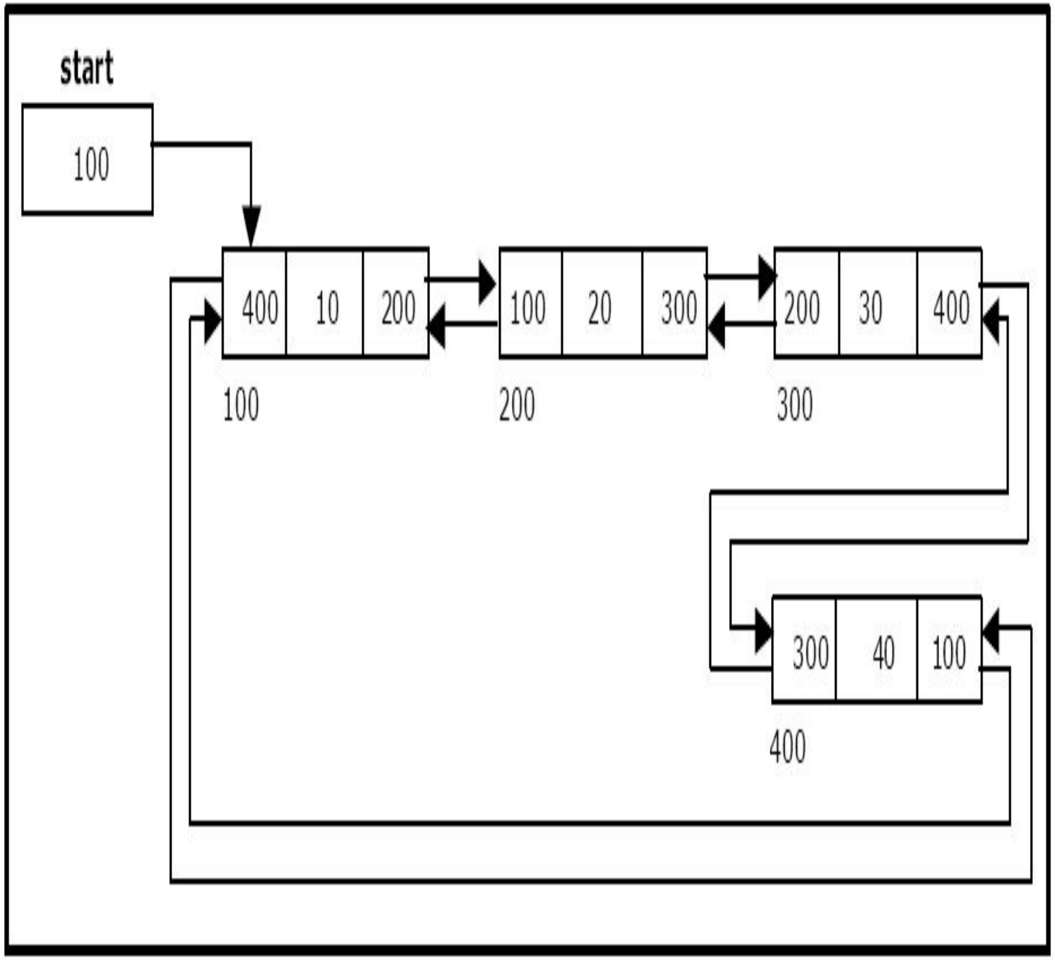
Insertion at first in CDLL

```
if(head==NULL) {
newnode->next=newnode;
newnode->prev=newnode;
    head=newnode; }
 else {
 Node *temp;
temp=head;
newnode->next=temp;
 newnode->prev=temp->prev
 temp->prev->next=newnode
temp->prev=newnode;
 head=newnode;
}
```



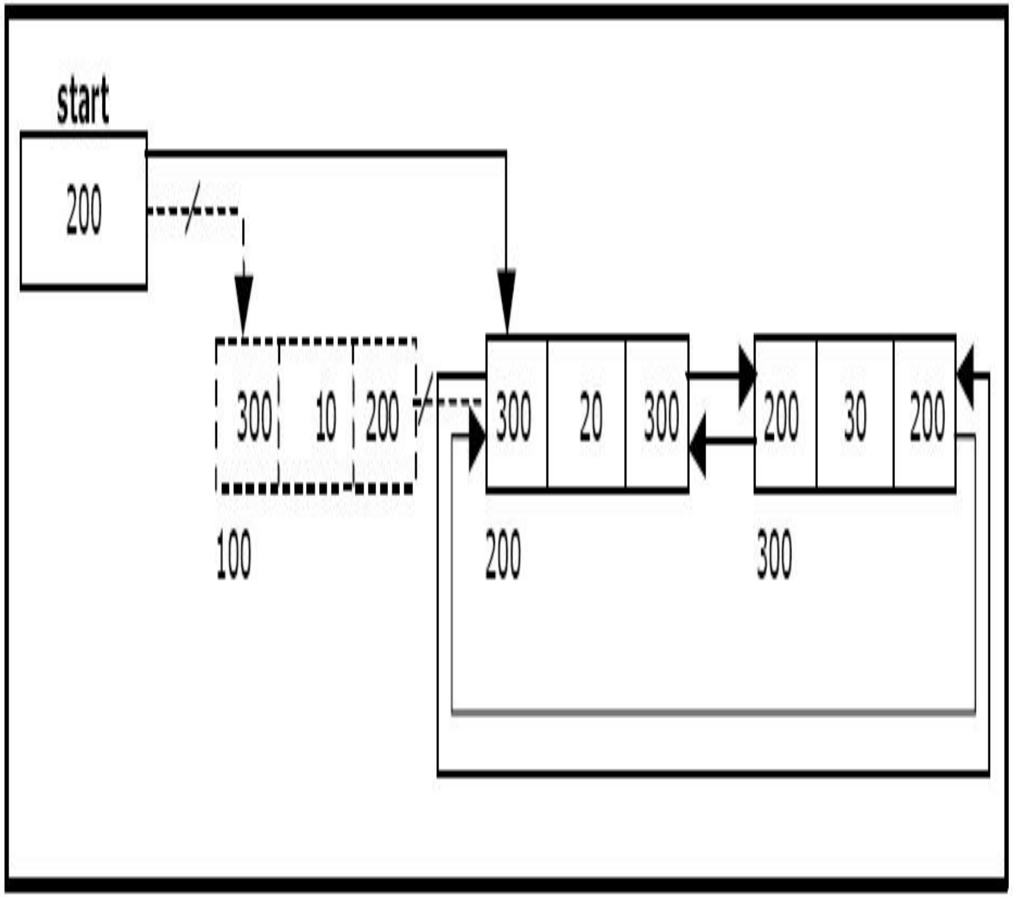
Insert at the end in CDLL

```
if(head==NULL) {
head=newnode;
 newnode->next=newnode;
 newnode->prev=newnode; }
else { Node *temp;
 temp=head;
temp=temp->prev;
 newnode->next=head;
newnode->prev=temp;
 temp->next=newnode;
 head->prev=newnode; }
```



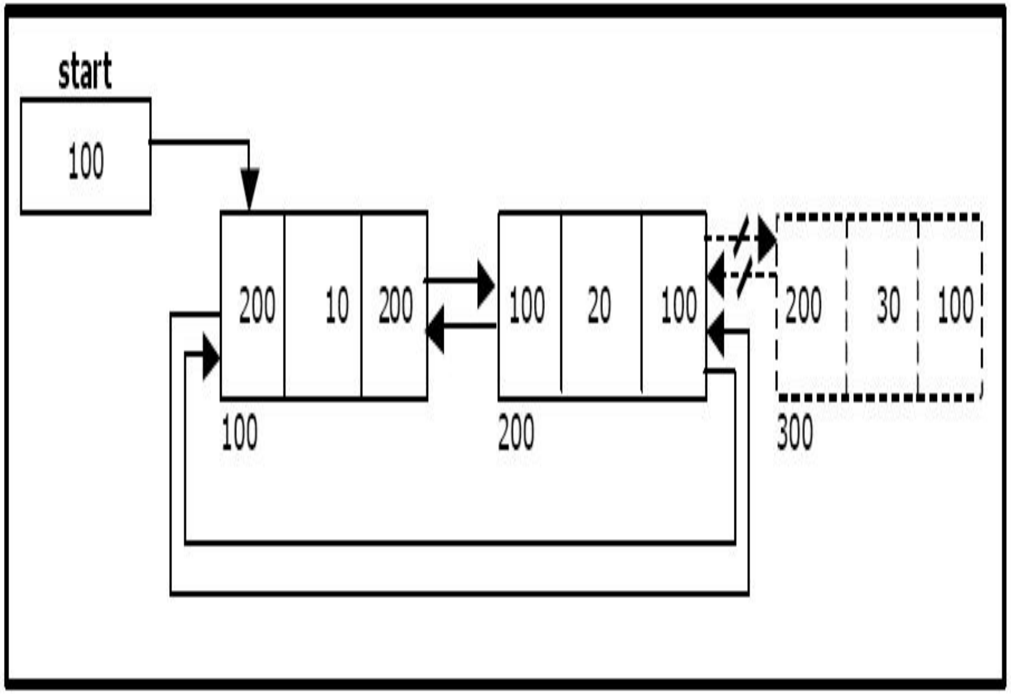
Delete first in CDLL

```
if(head->next==head) {
head=NULL;
return;
 }
Node *temp,*nhead;
 temp=head;
nhead=head->next;
 nhead->prev=temp->prev;
temp->prev->next=nhead;
 head=nhead;
delete(temp);
```



Delete last in CDLL

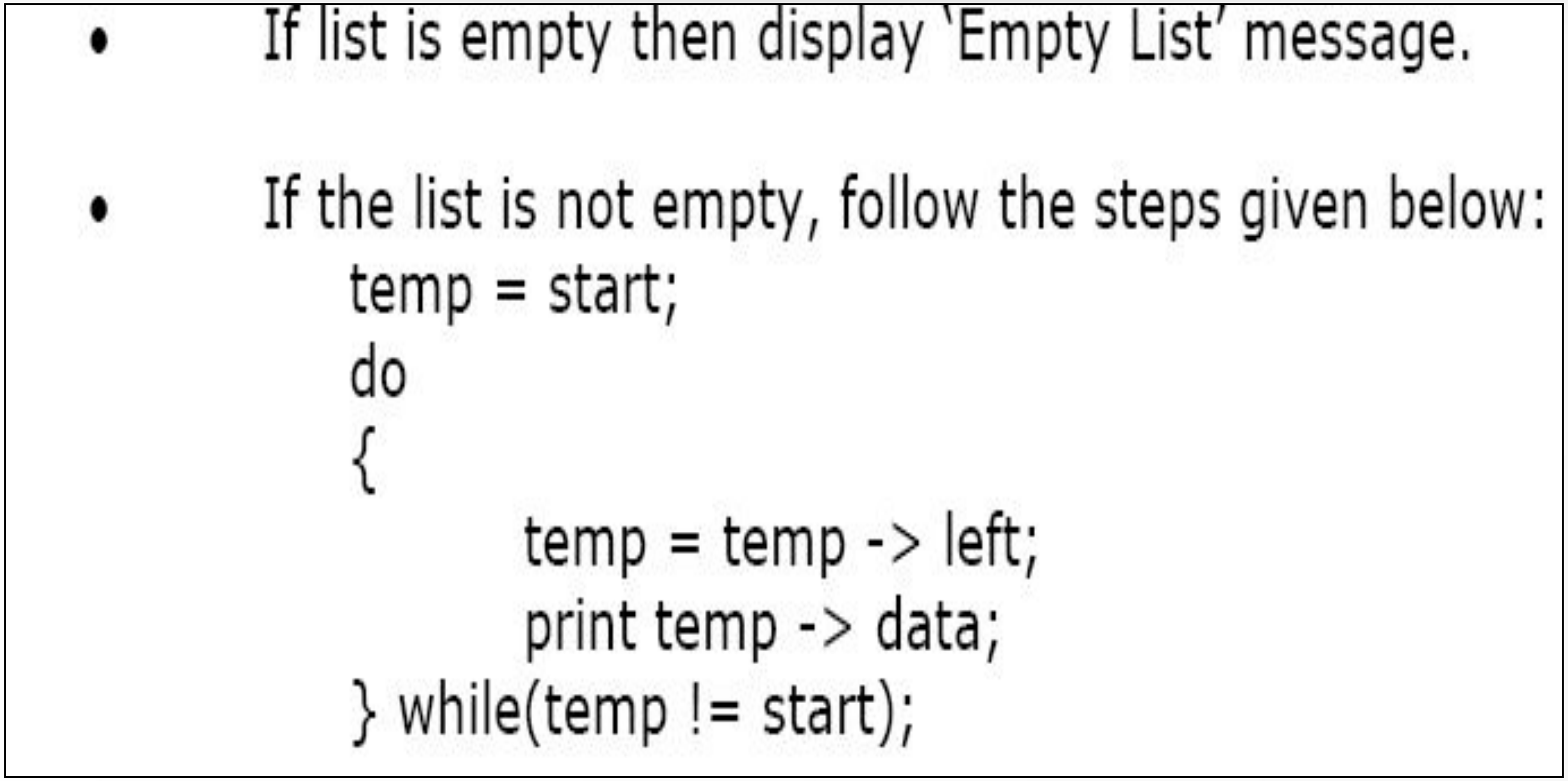
```
if(head->next==head) {
 head=NULL;
 return; }
Node *temp,*nlast;
 temp=head->prev;
nlast=temp->prev;
 nlast->next=head;
head->prev=nlast;
 delete(temp);
```



Search CDLL

```
int data,posn;
 cout<<"\nEnter the data you want to operate upon: "; cin>>data;
Node *temp;
temp=head;
posn=1;
while(temp->next!=head) {
 if(temp->data==data) return posn;
 posn++;
temp=temp->next;
} if(temp->data==data) return posn;
cout<<"reached end of search"; return 0;
```

Traversing CDLL



- If list is empty then display 'Empty List' message.
- If the list is not empty, follow the steps given below:

```
temp = start;
do
{
        temp = temp -> left;
        print temp -> data;
} while(temp != start);
```

Polynomial Manipulation

- Representation
- Addition
- Multiplication

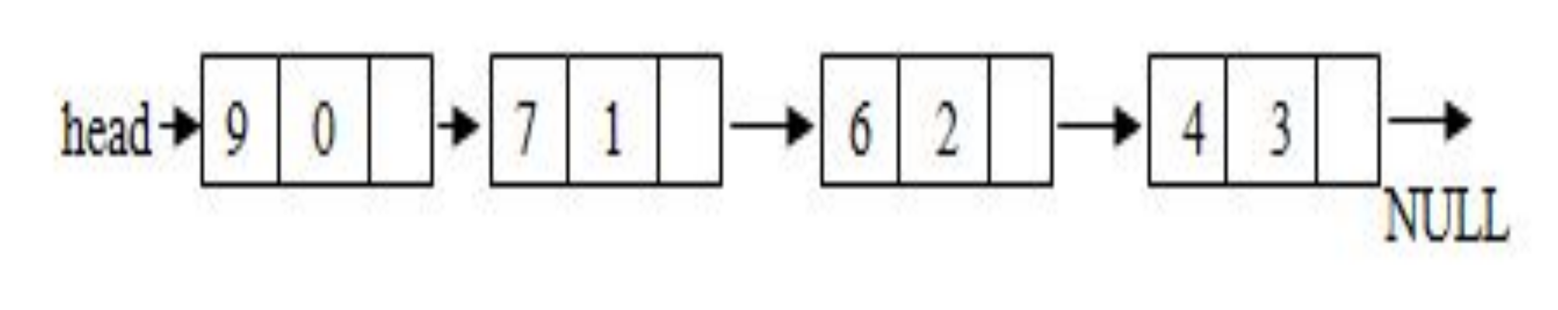
Representation

- A polynomial may also be represented using a linked list. A structure may be defined such that it contains two parts- one is the coefficient and second is the corresponding exponent. The structure definition may be given as shown below:

-
  ```
  struct polynomial
  {
  int coefficient;
  int exponent;
  struct polynomial *next;
  };
  ```

Example

- $P(x) = 4x^3+6x^2+7x+9$



Cont..

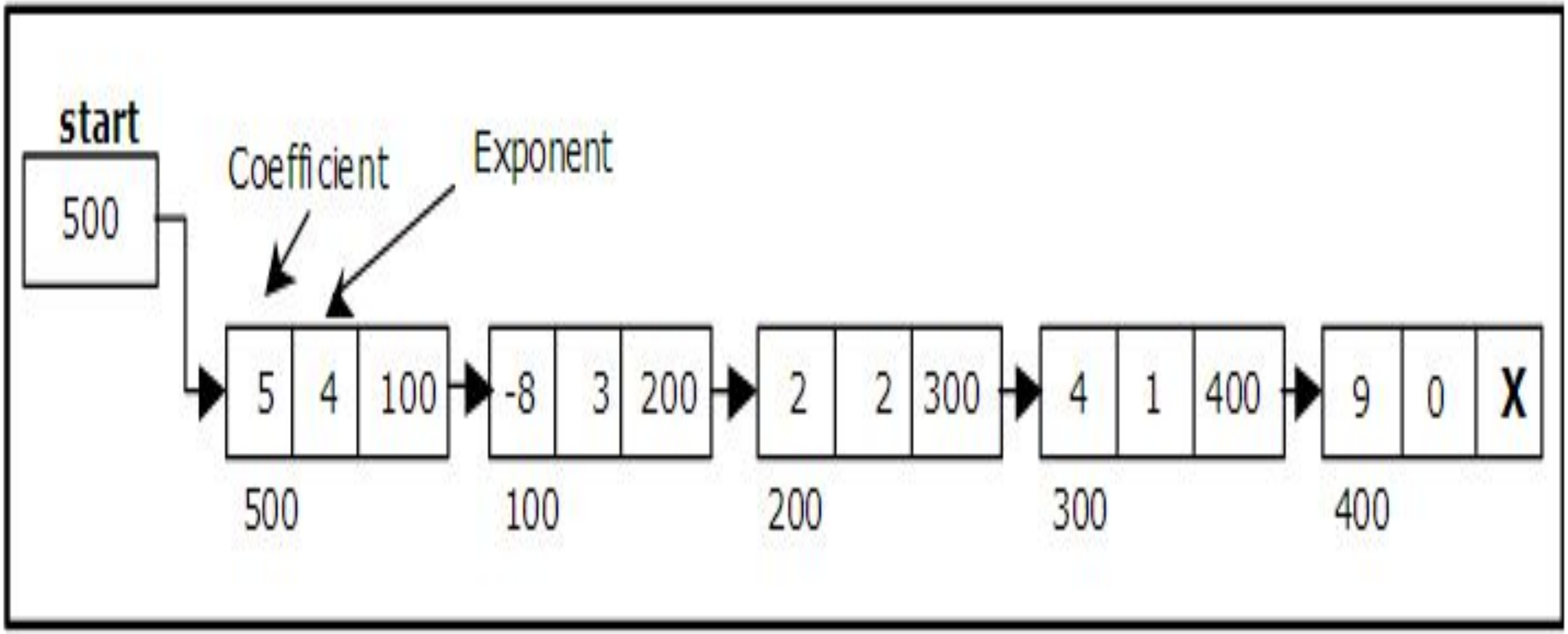


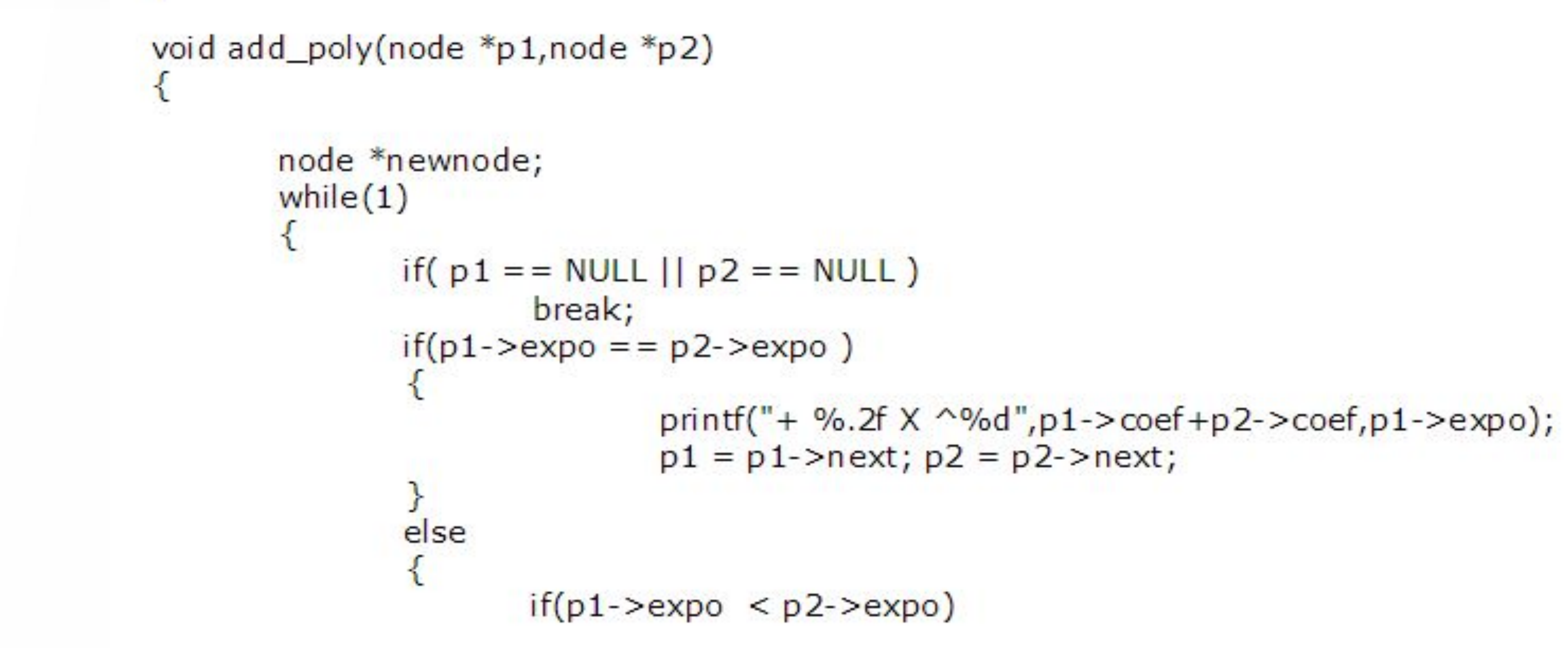
Figure 3.10.1. Single Linked List for the polynomial $F(x) = 5x^4 - 8x^3 + 2x^2 + 4x^1 + 9x^0$

Addition of polynomial

- Adding polynomials :

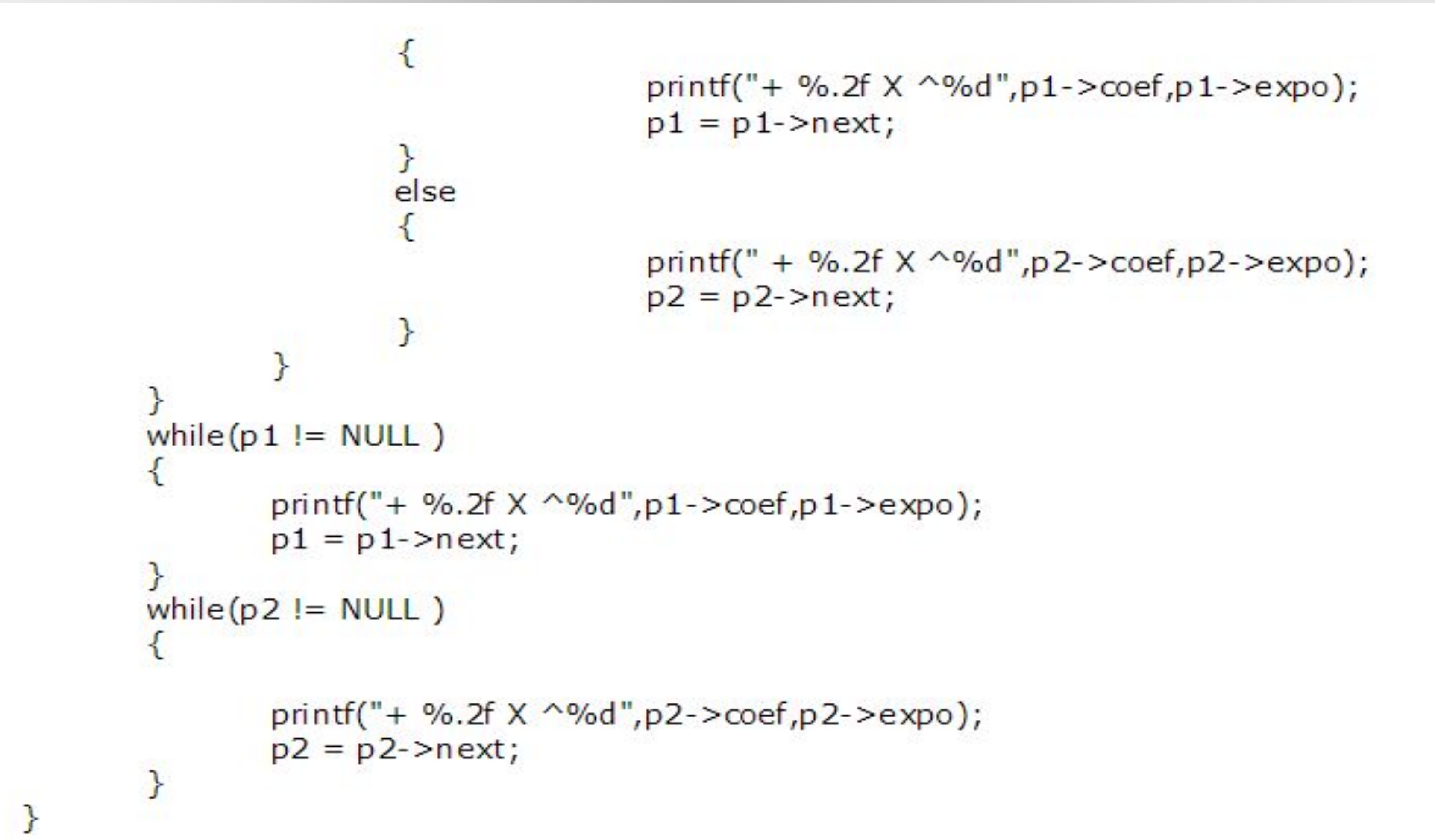
$$(3x^5 - 9x^3 + 4x^2) + (-8x^5 + 8x^3 + 2)$$
$$= 3x^5 - 8x^5 - 9x^3 + 8x^3 + 4x^2 + 2$$
$$= -5x^5 - x^3 + 4x^2 + 2$$

Function for addition



```
void add_poly(node *p1,node *p2)
{

        node *newnode;
        while(1)
        {
                if( p1 == NULL || p2 == NULL )
                        break;
                if(p1->expo == p2->expo )
                {
                                printf("+ %.2f X ^%d",p1->coef+p2->coef,p1->expo);
                                p1 = p1->next; p2 = p2->next;
                }
                else
                {
                        if(p1->expo  < p2->expo)
```

Cont..



```
            {
                    printf("+ %.2f X ^%d",p1->coef,p1->expo);
                    p1 = p1->next;
            }
            else
            {
                    printf(" + %.2f X ^%d",p2->coef,p2->expo);
                    p2 = p2->next;
            }
        }
    }
    while(p1 != NULL )
    {
        printf("+ %.2f X ^%d",p1->coef,p1->expo);
        p1 = p1->next;
    }
    while(p2 != NULL )
    {

        printf("+ %.2f X ^%d",p2->coef,p2->expo);
        p2 = p2->next;
    }
}
```

Multiplication of polynomials

- Multiplying polynomials:

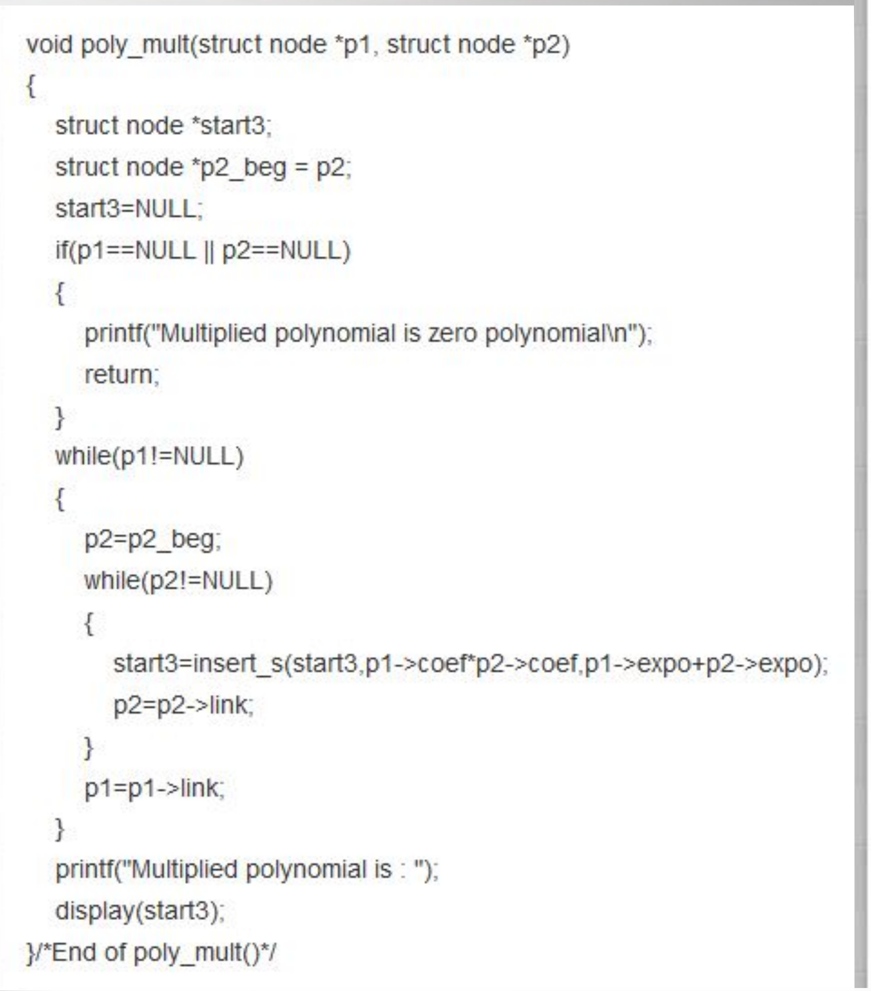
$$(2x - 3)(2x^2 + 3x - 2)$$

$$= 2x(2x^2 + 3x - 2) - 3(2x^2 + 3x - 2)$$

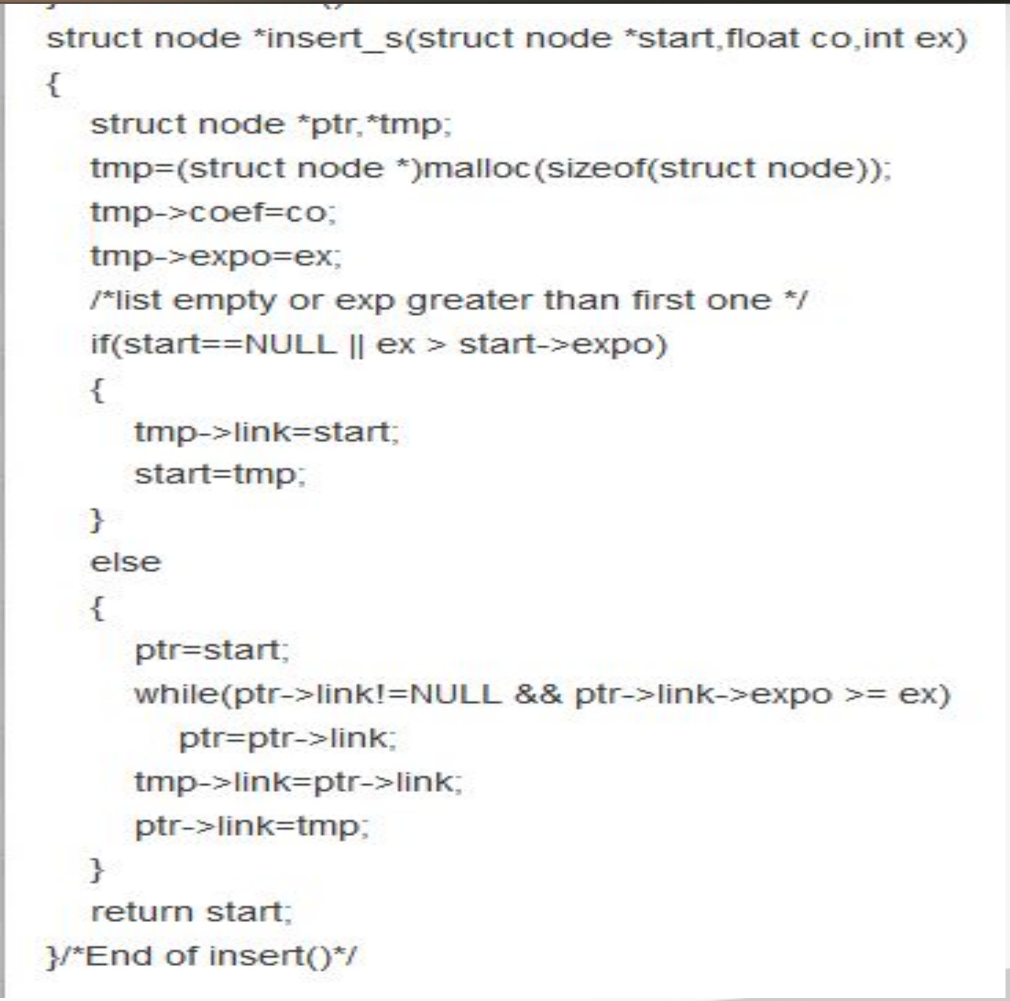
$$= 4x^3 + 6x^2 - 4x - 6x^2 - 9x + 6$$

$$= 4x^3 - 13x + 6$$

Function for multiplication



```
void poly_mult(struct node *p1, struct node *p2)
{
   struct node *start3;
   struct node *p2_beg = p2;
   start3=NULL;
   if(p1==NULL || p2==NULL)
   {
      printf("Multiplied polynomial is zero polynomial\n");
      return;
   }
   while(p1!=NULL)
   {
      p2=p2_beg;
      while(p2!=NULL)
      {
         start3=insert_s(start3,p1->coef*p2->coef,p1->expo+p2->expo);
         p2=p2->link;
      }
      p1=p1->link;
   }
   printf("Multiplied polynomial is : ");
   display(start3);
}/*End of poly_mult()*/
```



```
struct node *insert_s(struct node *start,float co,int ex)
{
    struct node *ptr,*tmp;
    tmp=(struct node *)malloc(sizeof(struct node));
    tmp->coef=co;
    tmp->expo=ex;
    /*list empty or exp greater than first one */
    if(start==NULL || ex > start->expo)
    {
        tmp->link=start;
        start=tmp;
    }
    else
    {
        ptr=start;
        while(ptr->link!=NULL && ptr->link->expo >= ex)
            ptr=ptr->link;
        tmp->link=ptr->link;
        ptr->link=tmp;
    }
    return start;
}/*End of insert()*/
```

Generalized linked list

- A *generalized list*, *A,* is a finite sequence of *n* > 0 elements, a$_{1}$, ..., $_{n}$where the $_{i}$ are either atoms or lists. The elements $_{i}$, 1 *i n* which are not atoms are said to be the *sublists* of *A*.
- The list *A* itself is written as *A* = (a$_1$, ..., a$_n$). *A* is the *name* of the list (a$_1$, ..., a$_n$) and *n* its *length*. By convention, all list names will be represented by capital letters. Lower case letters will be used to represent atoms. If *n*>= 1, then a$_1$ is the *head* of A while (a$_2$, ..., a$_2$) is the *tail* of *A*.
-

Examples

i)D = ( ) the null or empty list, its length is zero.

(ii) A = (a, (b,c)) a list of length two; its first element is the atom 'a' and its second element is the linear list (b,c).

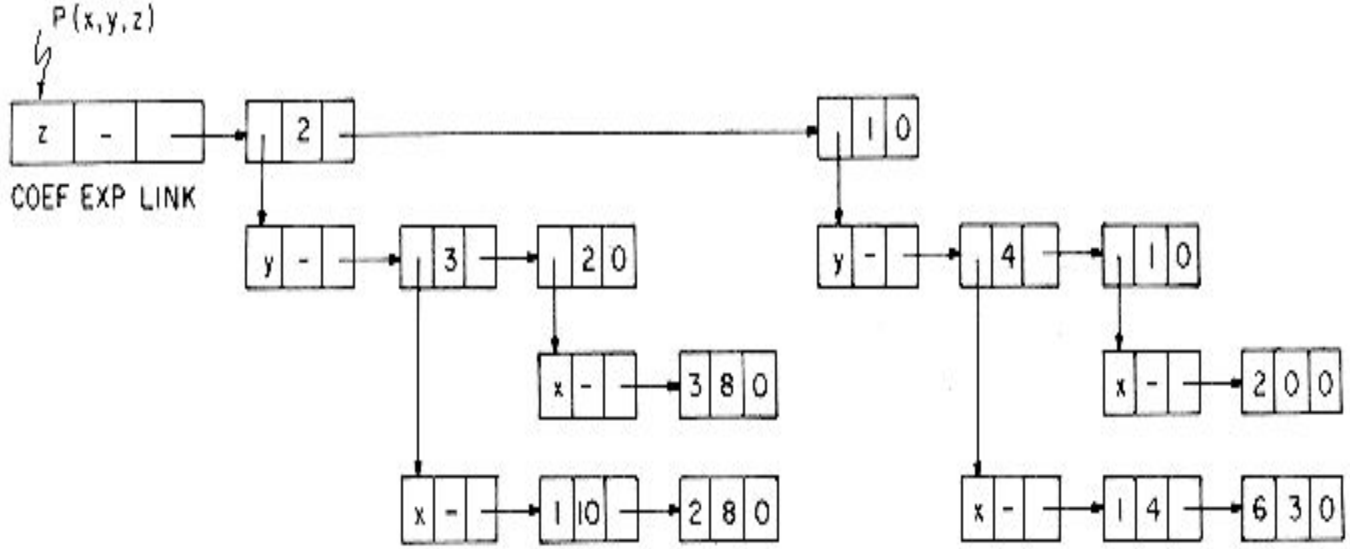
(iii) B = (A,A,( )) a list of length three whose first two elements are the lists A, the third element the null list.

(iv) C = (a, C) a recursive list of length two. C corresponds to the infinite list C = (a,(a,(a, ...).

Multivariable polynomial using GLL

- $x^{10}\, y^3\, z^2 + 2x^8\, y^3\, z^2 + 3x^8\, y^2\, z^2 + x^4\, y^4\, z + 6x^3\, y$ $z + 2yz$
- re-writing $P(x,y,z)$ as
- $((x^{10} + 2x^8)y^3 + 3x^8y^2)z^2 + ((x^4 + 6x^3)y^4 + 2y)z$
- $Cz^2 + Dz$,
- Looking closer at $C(x,y)$, we see that it is of the form $Ey^3 + Fy^2$

Cont..



THANK
YOU