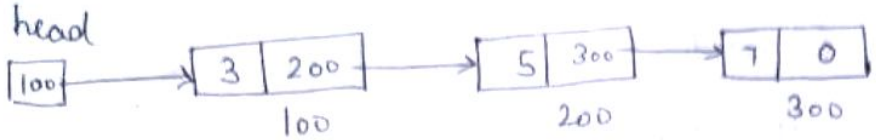
Single linked list :-

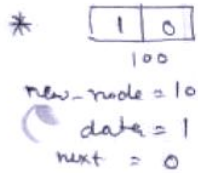


Creation of Single linked list

```
void main()
{
    Struct node                                  // Create a structure of node.
    {
        int data;
        Struct node * next;
    };

    Struct node * head, * new_node, * temp;
    head = 0;
    int choice = 1;
    while (Choice)
    {
        new_node = (Struct node *) malloc (Size of (Struct node));
        printf (" enter data of new_node");
        scanf (" %d", & new_node -> data);
        new_node -> next = 0;
        if (head == 0)                           // for first node
        {
            head = temp = new_node;
        }
        else
        {
            temp -> next = new_node;
            temp = new_node;
        }
        printf ("Do you want to continue, press 0 or 1");
        scanf ("%d", & choice);
    }
```

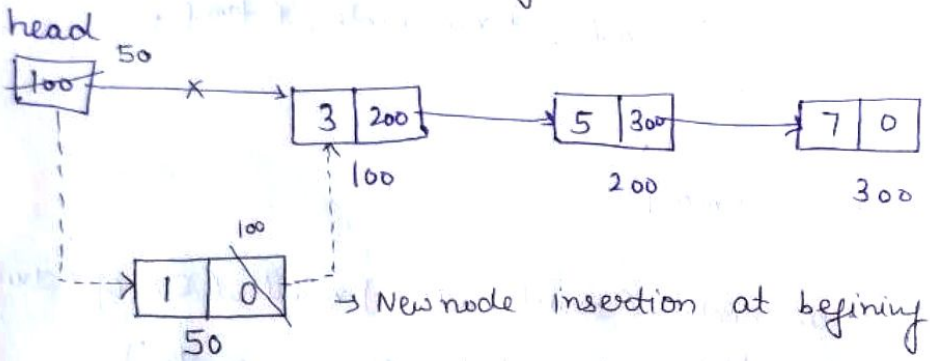
// memory allocation of new node.



```
temp = head;
// For display the data
while (temp != 0)
{
    printf("%d", temp->data);
    temp = temp->next;
}
getch();
}
```

Insertion in single linked list :-

a) Insertion at beginning :->



Note :- * [first write the code for creation (as defined previous)]

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

struct node *head, *new_node;
new_node = (struct node *) malloc (size of (struct node));
printf (" enter data that you want to insert");
scanf ("%d", &new_node->data);
```

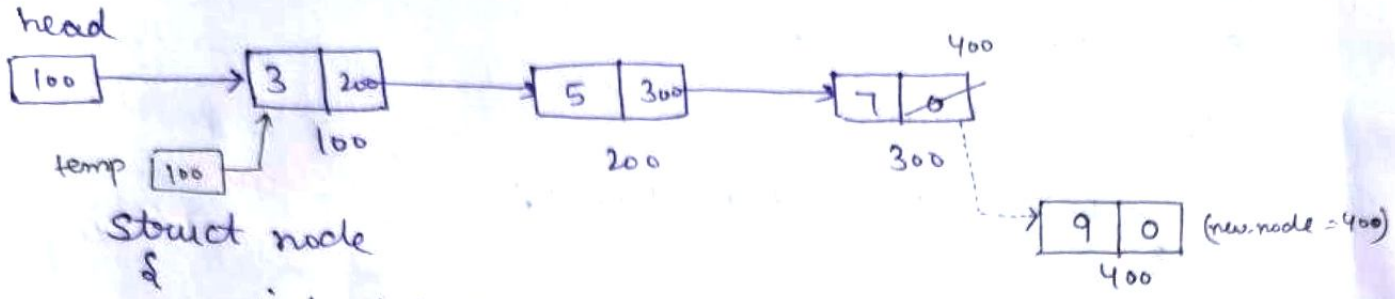
[* head = 100, new_node = 50]

```
new_node->next = head;
head = new_node;
```

] insertion at beginning

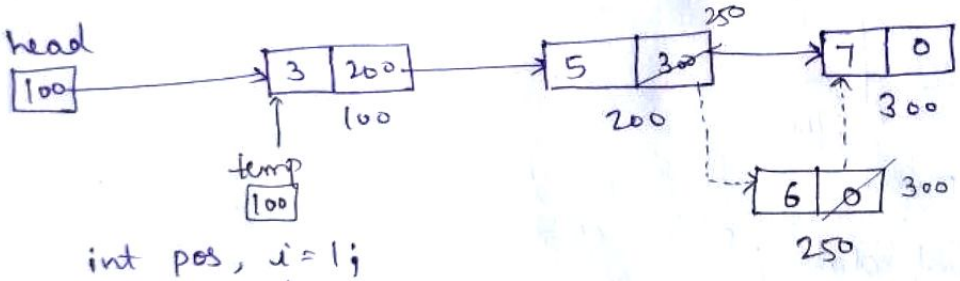
Note :- * Also write code for display. (explained previous)]

Insertion at end :→




```
struct node
{
    int data;
    struct node * next;
};
struct node * head, * new_node, * temp;
new_node = (struct node *) malloc (size of (struct node));
printf (" enter data that you want to insert");
scanf ("%d", & new_node -> data);
    new_node -> next = 0;
    temp = head;                    *(temp=100 = head)
while (temp->next != 0)             * Loop will continue till end.
{
    temp = temp->next;
    temp->next = new_node;
}
```

c) ## Insertion at specified position :→




```
int pos, i = 1;
struct node
{
    int data;
    struct node *next;
};
struct node * head, * new_node, * temp;
new_node = (struct node *) malloc (size of (struct node));
```

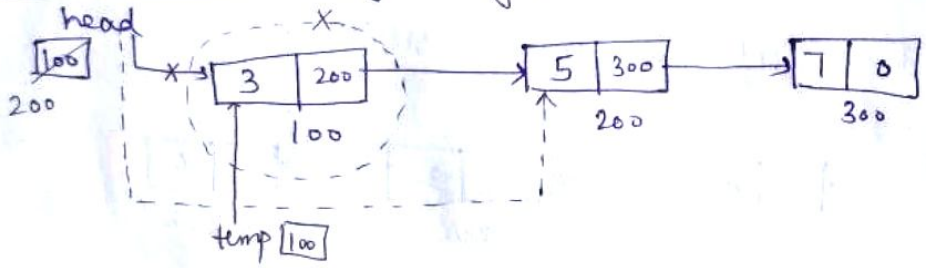
```
printf (" enter position");
scanf ("%d", &pos);
if (pos > count)
{
    printf ("Invalid position");
}
else
{
    temp = head;
    while (i < pos)
    {
        temp = temp->next;
        i++
    }
printf (" enter data that you want to insert");
scanf ("%d", &new_node->data);
new_node->next = temp->next;
temp->next = new_node;
}
```

* If position is less than 0 & greater than length.

* Reach till position

Deletion in single linked list :-

a) Deletion at beginning :-

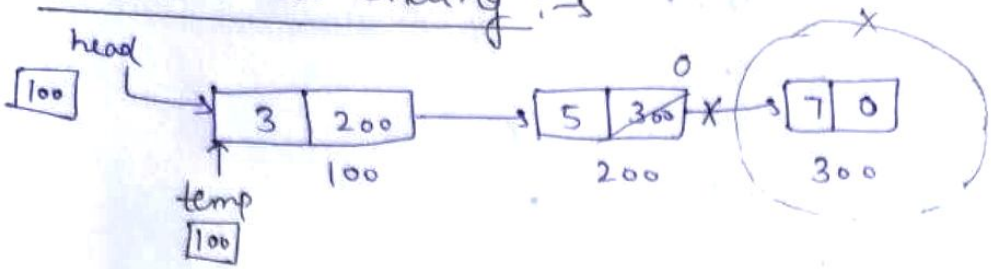



```
Struct node
{
    int data;
    Struct node * next;
};

Struct node * head, * temp;
temp = head;
head = head->next;
free (temp);
```

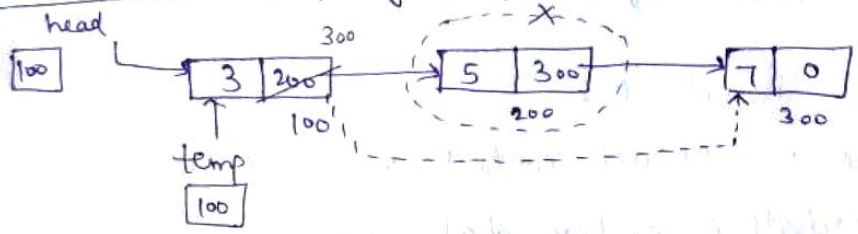
*deletion at beginning

Deletion at ending :→




```
void delete_from_end()
{
    struct node * prev_node;
      temp = head;
    while ( temp -> next != 0 )
    {
            prev_node = temp;
              temp = temp -> next;
    }
    if ( temp == head )
    {
            head = 0;
    }
    else
    {
            prev_node -> next = 0;
    }
    free (temp);
}
```

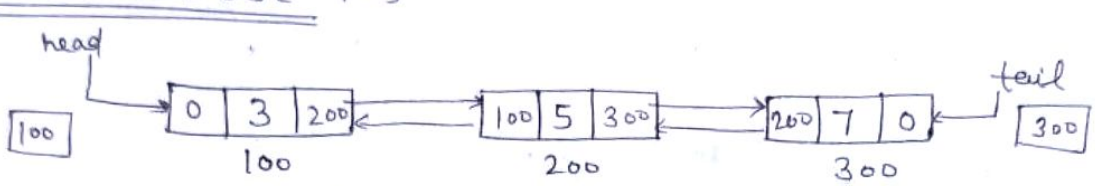
c) ## Deletion at Specified position :→




```
struct node * head, * temp, next_node;
void deletefrom_pos()
{
    int pos, i=1;
      temp = head;
    prin(" enter position");
```

```
scanf("%d", &pos);
while (i < pos-1)
{
    temp = temp->next;
    i++;
}
next_node = temp->next;
temp->next = next_node->next;
free(next_node);
}
```

Double Linked list :→



Creation of double Linked list :→

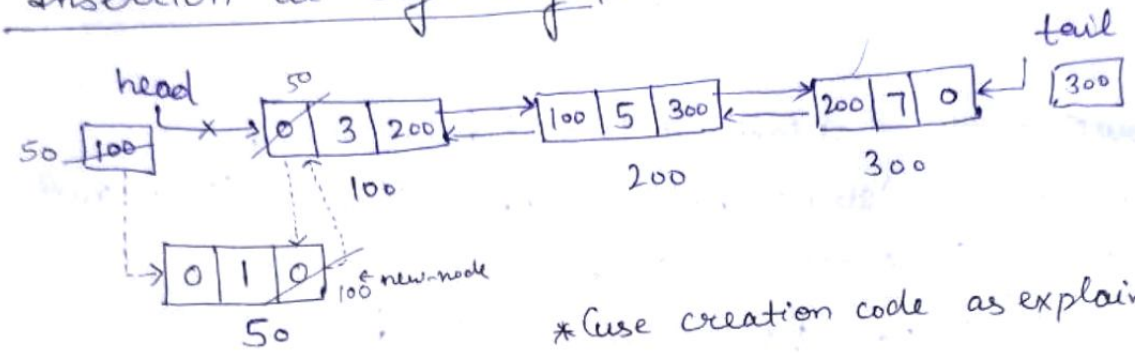
```
Struct node
{
    int data;
    Struct node * next;
    Struct node * prev;
};
Struct node *head, *new_node, *temp;
void create()
{
    head = 0;
new_node = (struct node *) malloc (size of (Struct node));
    printf("enter data");
    scanf("%d", &new_node->data);
    new_node->prev = 0;
    new_node->next = 0;
if (head == 0)
{
    head = temp = new_node;
}
```

```
else
{
        temp->next = new_node;
        new_node->prev = temp;
        temp = new_node;
}
```

(* For display, same code as single linked list.)

Insertion in double linked list :-

a) Insertion at beginning :-



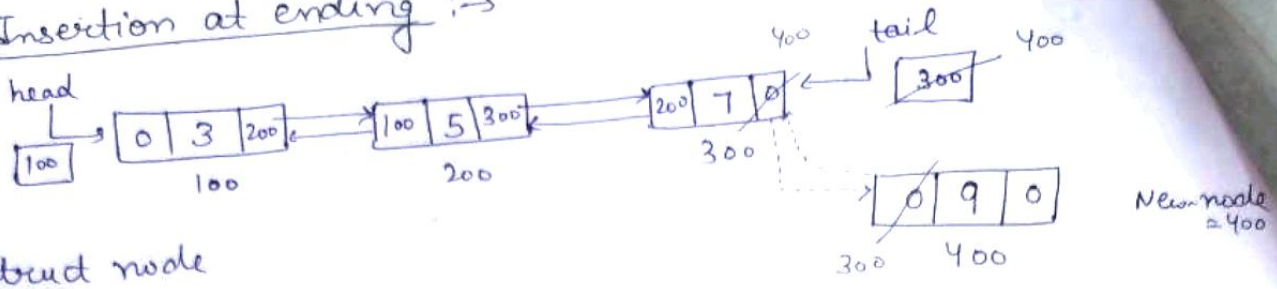
*Use creation code as explained before

```
Struct node
{
    int data
    Struct node * next;
    Struct node * prev;
};
Struct node * head, * tail, * new_node;
    head = 0;
new_node = (Struct node *) malloc (size of (Struct node));
   printf ("enter data");
   Scanf ("%d", & new_node->data);
    new_node->prev = 0;
    new_node->next = 0;
    if (head == 0)
    {
        head = tail = new_node;
    }
    else
    {
        tail->next = new_node;
        new_node->prev = tail;
        tail = new_node;
    }
```

insertion

(* use display code as explained *)

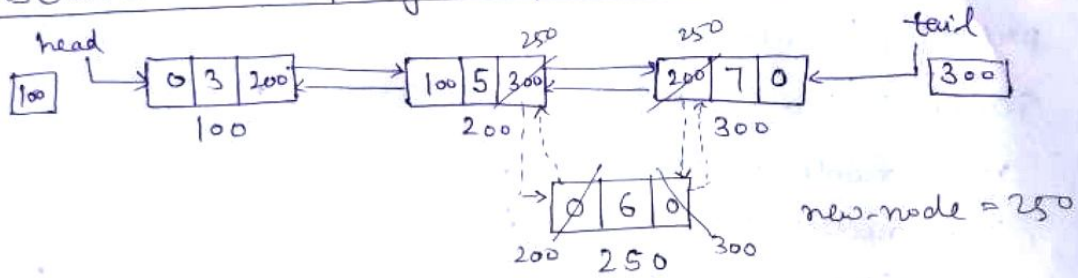
b) Insertion at ending :→




```
struct node
{
    int data;
    struct node * next;
    struct node * prev;
};
struct node * head, * tail, * new_node;
new_node = (struct node *) malloc (size of (struct node));
  printf (" enter data");
  scanf ("%d", &new_node → data);
   new_node → prev = 0;
   new_node → next = 0;
    tail → next = new_node;
    new_node → prev = tail;
      tail = new_node;
```

} insertion at end

c) Insertion at specified position :→



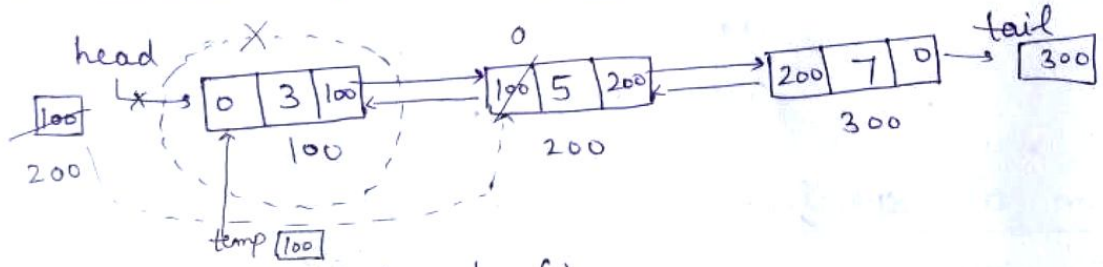

```
struct node
{
    int data;
    struct node * next;
    struct node * prev;
};
struct node * head, * tail, * new_node;
new_node = (struct node *) malloc (size of (struct node));
```

```
printf("enter data");
scanf("%d", &new_node->data);
new_node->prev = 0;
new_node->next = 0;
int pos, i=1;
printf("enter position");
scanf("%d", &pos);
while(i < pos-1)
{
    temp = temp->next;
    i++;
}
new_node->prev = temp;
new_node->next = temp->next;
temp->next = new_node;
```

insertion at position

Deletion in double linked list :→

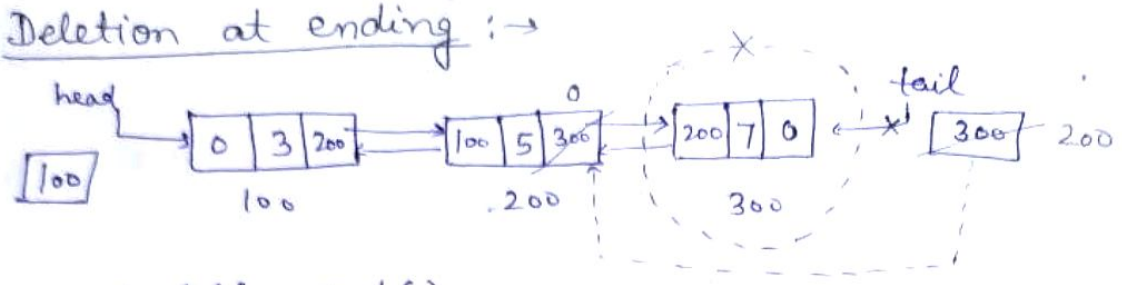
a) Deletion at beginning :→




```
void deletefrombeg()
{
    struct node * temp;
    if(head==0)
    {
        printf("list is empty");
    }
    else
    {
        temp = head;
```

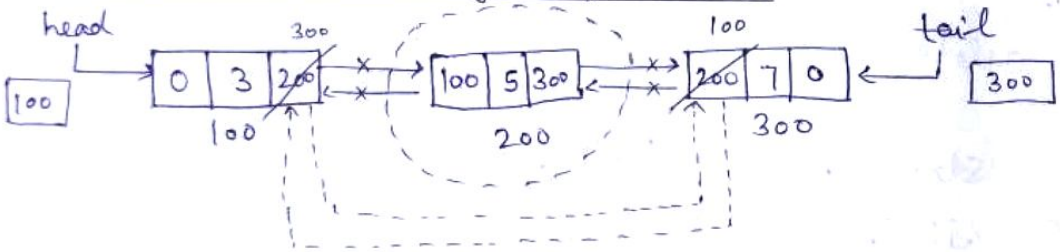
```
    head = head -> next ;
    head -> prev = 0 ;
    free (temp) ;
}
```

b) Deletion at ending :→




```
Void delfromend()
{
    Struct node * temp ;
    if (tail = = 0)
    {
        printf (" List is empty") ;
    }
    else
    {
        temp = tail ;
        tail -> prev -> next = 0 ;
        tail = tail -> prev ;
        tail -> next = 0 ;
        free ( temp) ;
    }
```

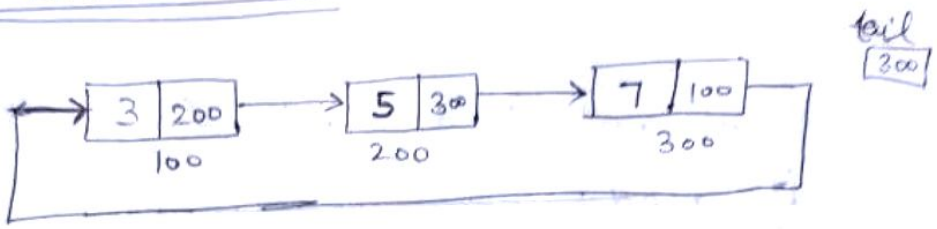
c) Deletion at Specified position :→




```
Void deffrompos()
{
    int pos , i = 1 ;
    Struct node * temp , * head ;
    temp = head ;
```

```
printf (" enter position ");
scanf (" %d", &pos);
while (i < pos)
{
    temp = temp->next;
    i++;
}
temp->prev->next = temp->next;
temp->next->prev = temp->prev;
free (temp);
```

Circular Linked list :→



Creation of Circular Linked list :→

```
Struct node
{
    int data;
    Struct node * next;
};
Struct node * head, * new_node, * temp;
int choice = 1;
head = 0;
while (choice)
{
new_node = (struct node*) malloc (Size of (Struct node));
printf (" enter data");
scanf ("%d", & new_node -> data);
new_node -> next = 0;
```

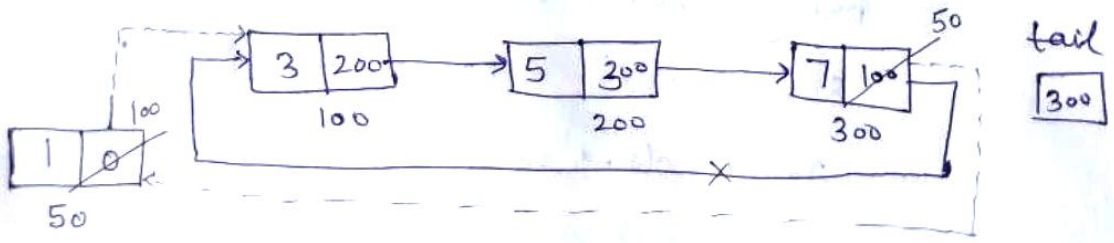
```
if (head == 0)
{
        head = temp = new_node;
}
else
{
        temp->next = new_node;
        temp = new_node;
}
temp -> next = head;
printf (" Do you want to continue ?");
scanf ("%d", &choice);
```

// For display

```
// Struct node * temp; // (if not initialized earlier)
if ( head == 0)
{
    printf (" list is empty ");
}
else
{
    temp = head;
    while (temp->next != head)
    {
        printf ("%d", temp->data);
        temp = temp->next;
    }
    printf ("%d", temp->data);
}
```

Insertion in circular linked list :-

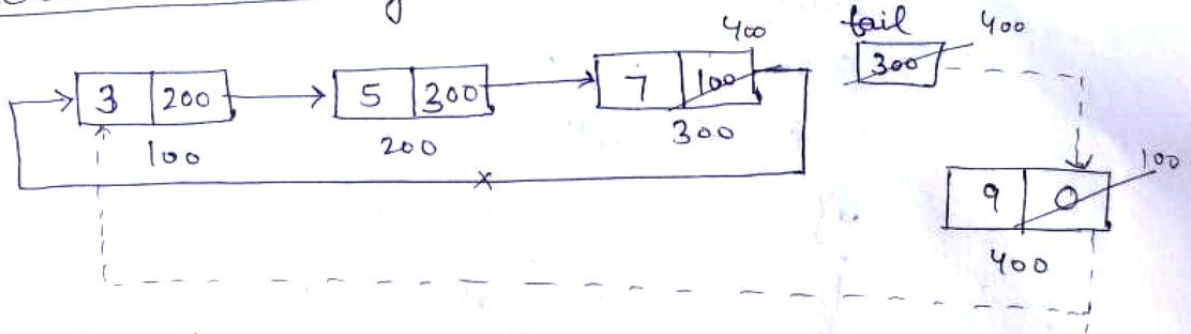
a) Insertion at beginning :-



```
Struct node
{
    int data;
    Struct node *next;
};
 Struct node * tail, new_node;
new_node = (Struct node *) malloc (size of (Struct node));
 printf ("enter data");
 scanf ("%d", & new_node -> data);
     new_node -> next = 0;
 if (tail == 0)
 {
        tail = new_node;
        tail -> next = new_node;
 }
 else
  {
    new_node -> next = tail -> next;
      tail -> next = new_node;
  }
```

* use display code for displaying item.

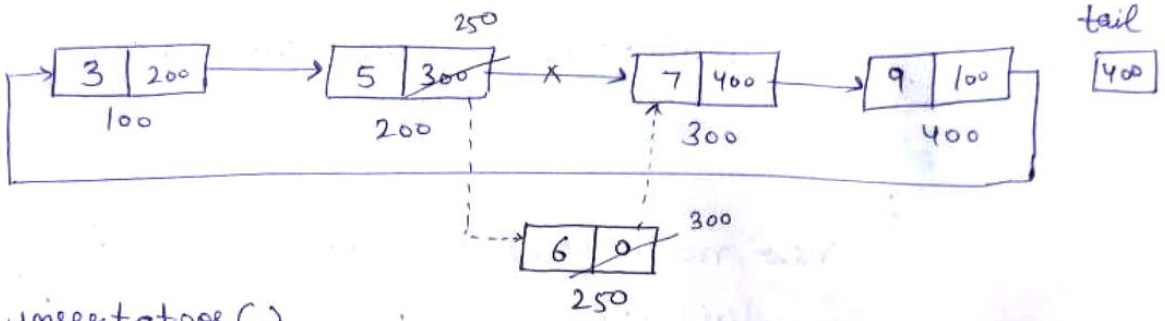
b) Insertion at ending :→




```
Struct node
 {
      int data;
       Struct node * next;
 };
   Struct node  *new_node, * tail;
new_node = (Struct node *) malloc (size of (Struct node));
    printf (" enter data");
```

```
scanf("%d", &new_node->data);
    new_node->next = 0;
if (tail == 0)
{
        tail = new_node;
        tail->next = new_node;
}
else
{
    new_node->next = tail->next;
    tail->next = new_node;
}
```

c) Insertion at Specified position :->

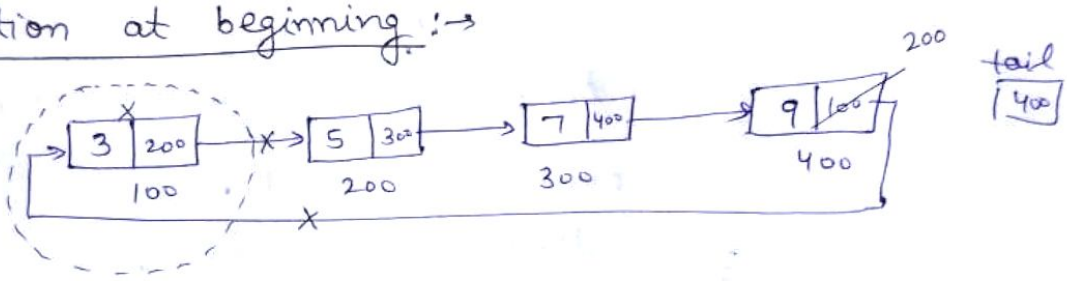



```
void insertatpos()
{
    struct node * new_node, * temp;
    int pos, i=1, l;
    printf("enter position");
    scanf("%d", &pos);
      l = getlength();
    if (pos < 0 || pos > l)
    {
        printf("invalid position");
    }
    else
    {
        new_node = (struct node *) malloc (size of (struct node));
        printf("enter data that you want to insert");
        scanf("%d", &new_node->data);
        new_node->next = 0;
```

```
temp = tail -> next ;
while ( i < pos-1)
{
        temp = temp -> next ;
          i++;
 }
new_node -> next = temp -> next ;
  temp -> next = new_node ;
}
}
```

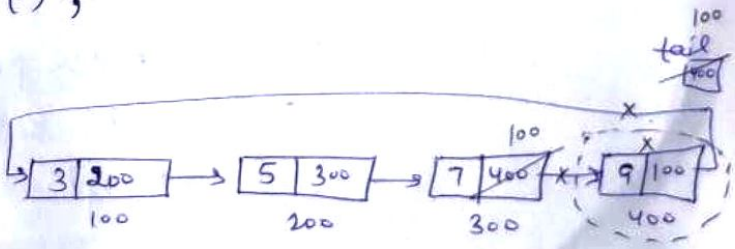
Deletion in Circular linked list :→

a) Deletion at beginning :→



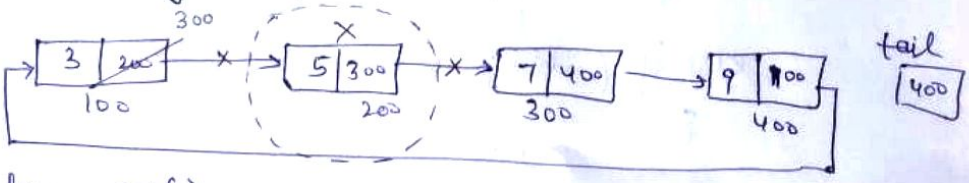

```
void del_from_beg()
{
    struct node * temp ;
     temp = tail -> next ;
    if (tail == 0)
     {   printf(" list is empty");
     }
   else if ( temp -> next == temp )
     {
          tail = 0;
          free (temp);
     }
   else
     {
        tail -> next = temp -> next ;
          free(temp) ;
     }
}
```

b) Deletion at ending :→



```
void del-from-end()
{
    struct node * temp, * prev-node;
    temp = tail->next;
    if (tail == 0)
    {
        printf("list is empty");
    }
    else if (temp->next == temp)
    {
        tail = 0;
        free (temp);
    }
    else
    {
        while (temp->next != tail->next)
        {
            prev-node = temp;
            temp = temp->next;
        }
        prev-node->next = tail->next;
        tail = prev-node;
        free (temp);
    }
}
```

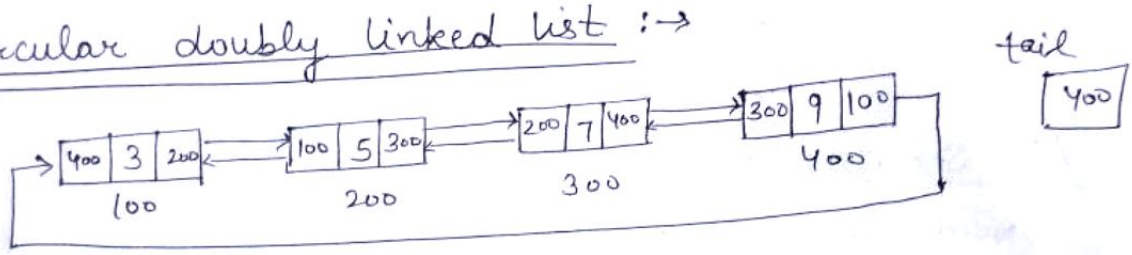
c) Deletion at Specified position :→




```
void del-from-pos()
{
    struct node *temp, * next-node;
    int pos, i=1, l;
    temp = tail->next;
    printf("enter position");
    scanf("%d", &pos);
    l = getlength();
```

```
if ( pos<1 || pos>c)
{
        printf ("Invalid position");
}
else if (pos == 1)
{
        del-from-beg();
}
else
{
        while (i < pos-1)
        {
                temp = temp->next;
                i++;
        }
        next_node = temp->next;
        temp->next = next_node->next;
        free (next_node);
}
}
```

Circular doubly linked list :→



Insertion :→

a) **At begin :-**

```
if (head == 0)
{
        head = tail = new_node;
        new_node -> prev = tail;
        new_node -> next = head;
}
else
{
        new_node -> next = head;
        head -> prev = new_node;
        new_node -> prev = tail;
```

```
        tail->next = new_node;
        head = new_node;
    }
```

b) At end :->

```
void insert_at_end()
{
    if (head == 0)
    {
        head = tail = new_node;
        new_node->prev = tail;
        new_node->next = head;
    }
    else
    {
        new_node->prev = tail;
        tail->next = new_node;
        new_node->next = head;
        head->prev = new_node;
        tail = new_node;
    }
}
```

c) At Specified position :->

```
void insert_at_pos()
{
    Struct node *new_node, *temp
    int pos, i = 1, l;
    printf("enter position");
    scanf("%d", &pos);
    l = getlength();
    if (pos<1 || pos>l)
    {
        printf("Invalid position");
    }
    else
    {
        new_node = (Struct node *) malloc (sizeof (Struct node));
```

```
printf("enter data you want to insert");
scanf("%d", &new_node->data);
while (i < pos-1)
{
    temp = temp->next;
    i++;
}
new_node->prev = temp;
new_node->next = temp->next;
temp->next->prev = new_node;
temp->next = new_node;
}
```

Deletion in Doubly Circular linked list :-

a) **At begin :-**

```
void del_from_begin()
{
    struct node *temp;
    temp = tail;
    if (head == 0)
    {
        printf("list is empty");
    }
    else if (head->next == head)
    {
        head = tail = 0;
        free(temp);
    }
    else
    {
        head = head->next;
        head->prev = tail;
        tail->next = head;
        free(temp);
    }
```

b) **At end :-**

* Same as at begin, only else part changed.

```
else
{
        tail = tail -> prev;
        tail -> next = head;
        head -> prev = tail;
         free (temp);
}
```

c) At Specified position :->

```
* same code for 'position' before else *

else
{
        while (i < pos)
        {
                temp = temp -> next;
        }       i++;

        temp -> prev -> next = temp -> next;
        temp -> next -> prev = temp -> prev;
        if ( temp -> next == head)
        {
                tail = temp -> prev;
        }       free (temp);

        else
                {
                        free (temp);
                }
}
```

Creation :->

```
Struct node
{
        int data;
        Struct node *next;
        Struct node * prev;
};
  Struct node * head, * tail *new_node;
```

```
head = 0; int choice = 1;
while (choice)
{ new_node = (struct node *) malloc (size of (struct node));
printf ("enter data");
scanf ("%d", &new_node->data);
if (head == 0)
{
        head = tail = new_node;
        head->next = head;
        head->prev = head;
}
else
{
        tail->next = new_node;
        new_node->prev = tail;
        new_node->next = head;
        head->prev = new_node;
        tail = new_node;
}
printf ("Do you want to continue: 0 or 1");
scanf ("%d", &choice);
}
```

For display

```
if (head == 0)
{
    printf ("list is empty");
}
else
{
    while (temp != tail)
    {
        printf ("%d", temp->data);
        temp = temp->next;
    }
    printf ("%d", temp->data);
}
```