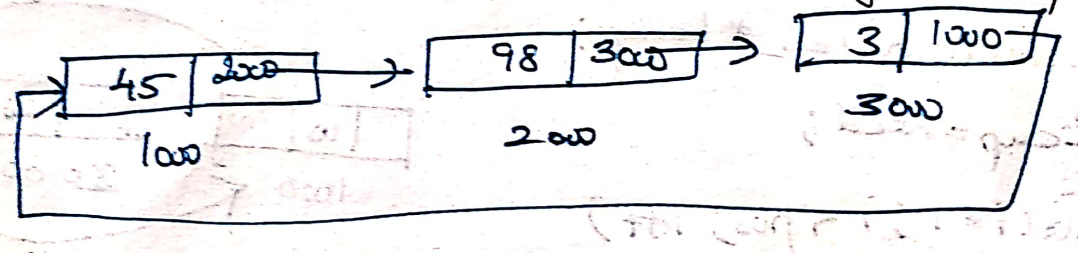
Circular Linked List.

There are two types of circular linked list.
- Circular singly linked list.
- Circular doubly linked list

Applications
* Scheduling, managing playlist, Table maintenance
* Browser surfing
  - Records of right visited by user so far are saved

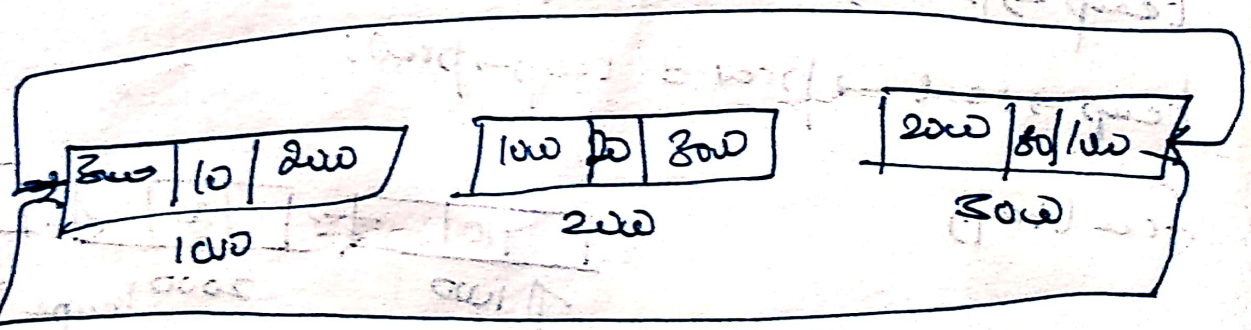
Circular singly list.

→ Similar to singly linked list. except that the last node of the circular singly linked list. points to the first node



Circular Doubly linked list.

→ Similar to doubly linked list. except that the last node of the circular doubly linked list. points to the first node & the first node circular doubly linked list. points to the last node

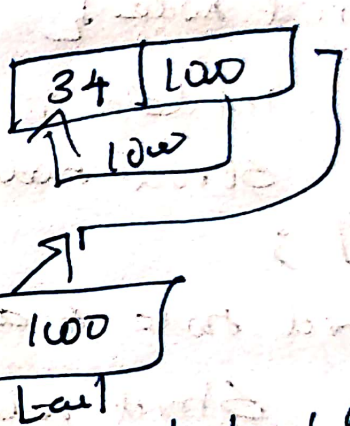


Operations
- Insertions at beginning Forward.
- Insertion at end
- Deletion at beginning
- Deletion at end.
- Searching

Creating a node of the circular singly list.

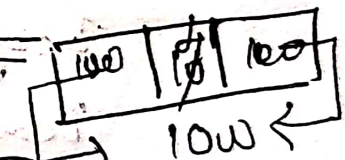
```
#include <stdio.h>
#include <stdlib.h>
struct node {
    int data;
    struct node *next;
};
int main()
{
  int data = 34;
  struct node *tail;
  tail = circularSingly(data);
  printf("%d", tail->data);
  return 0;
}
```

```
struct node * circular
Singly(int data)
{
struct node *temp =
malloc(sizeof(struct node));
temp->data = data;
temp->next = temp;
return temp;
}
```

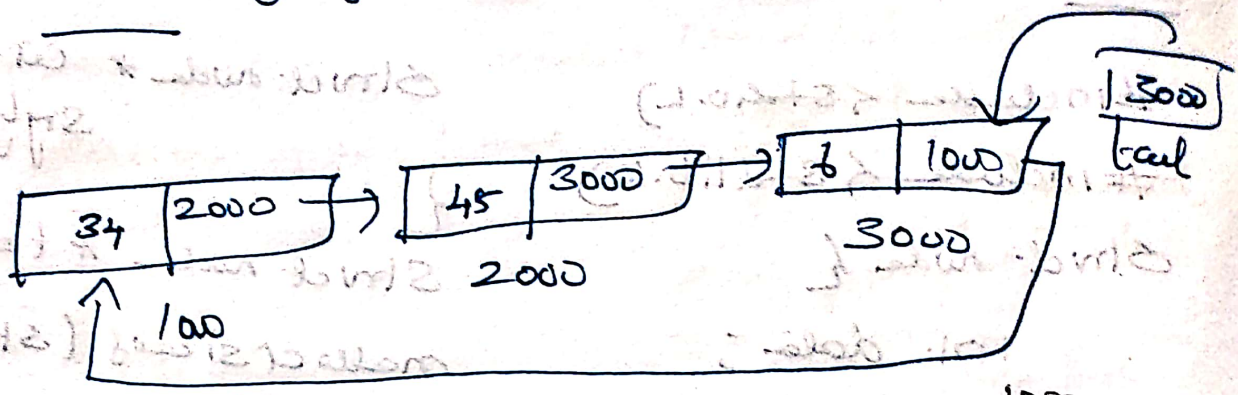


Creating a node in the circular doubly linked list.

```
#include <stdio.h>
#include <stdlib.h>
struct node {
  struct node *prev;
  int data;
  struct node *next;
};
int main()
{ int data = 45;
  struct node *tail;
  tail = circularDoubly(data);
  printf("%d", tail->data);
  return 0;
}
struct node * circularDoubly(int
                             data)
{ struct node *temp = malloc
    (sizeof(struct node));
  temp->data = data;
  temp->next = temp;
  temp->prev = temp; return temp;
}
```



Insertion at Beginning



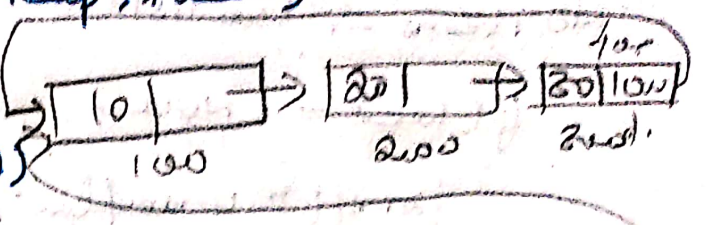

```
#include <stdio.h>
#include <stdlib.h>
struct node {
    int data;
    struct node *next;
};
struct node * addToEmpty(int data)
{
    struct node *temp = malloc(sizeof(struct node));
    temp->data = data;
    temp->next = temp;
    return temp;
}
struct node *addAtBeg(struct node *tail, int data)
{
    struct node *newp = malloc(sizeof(struct node));
    newp->data = data;
    newp->next = tail->next;
    tail->next = newp;
    return tail;
}
```

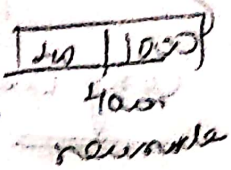
1000

```
newp->next = tail->next
tail->next = newp;
void print(struct node *tail)
{
    struct node *p = tail->next;
    do {
        printf("%d", p->data);
        p = p->next;
    } while(p != tail->next);
}
```

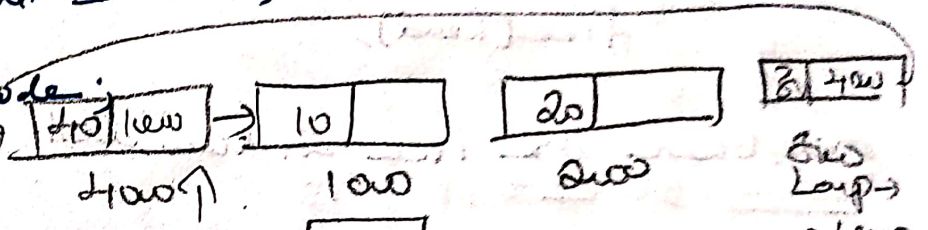
Insertion at beginning

```
Struct node * newnode, *temp, *head;
temp = head;
while(temp -> next != head)
    temp = temp -> next;
newnode = malloc(sizeof(struct node));
newnode -> data 10;
temp -> next = newnode;
newnode -> next = head;
head = newnode;
```





temp->next = newnode->next;



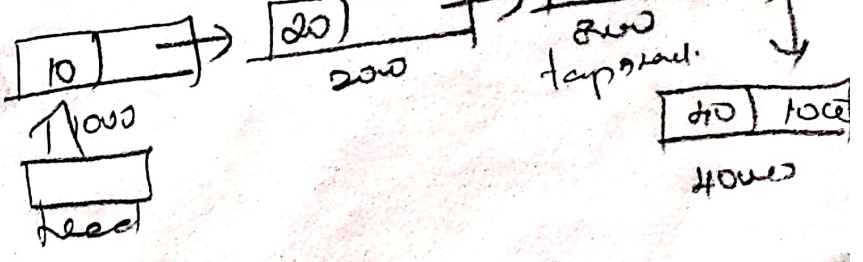
Insertion at end

```
Struct node *temp, *head, *newnode;
temp = head;
newnode = malloc(sizeof(struct node));
newnode -> data = 40;
while(temp -> next != head)
    temp = temp -> next;
newnode -> next = temp -> next;
temp -> next = newnode;
new -> next -> head;
head = new -> next;
```

temp -> next = tail -> next.

tail -> next = temp.

tail = tail -> next.



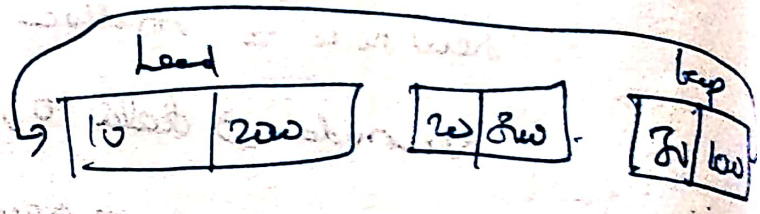
Deletion at beginning

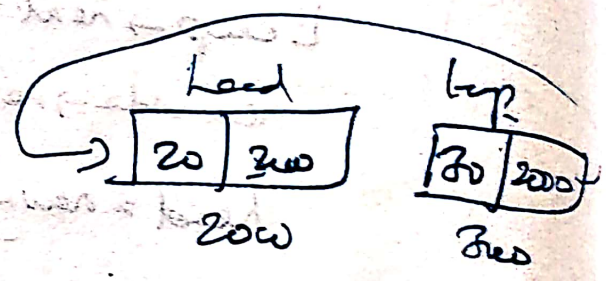
No items in the list

```
if(head == NULL)
    printf("Underflow");
```

Only one element in the list.

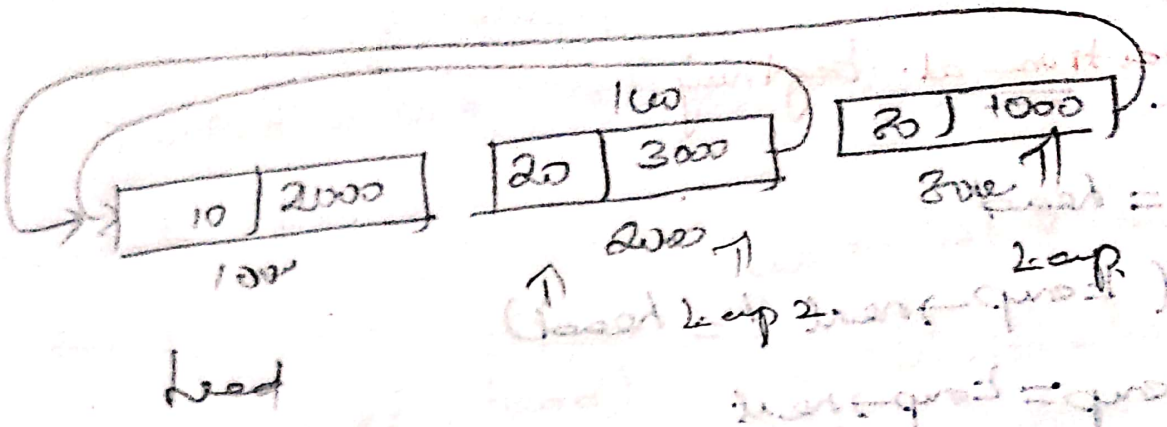
```
if(head->next == head)
{
    head = NULL;
    free(head);
}
```





More than one item in the list.

```
temp = head
while(temp->next != head)
    temp = temp->next
temp->next = head->next;
free(head)
head = temp->next;
```



Deletion at end

list is empty

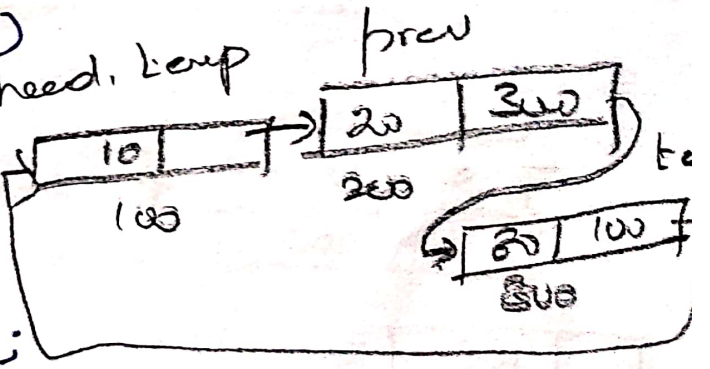
```
if (head == NULL)
   pf ("\n underflow");
```

list contains only one element.

```
if (head->next == head)
{
   head = NULL
   free(head);
}
```

list contains more than one element.

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



Circular Doubly Linked list

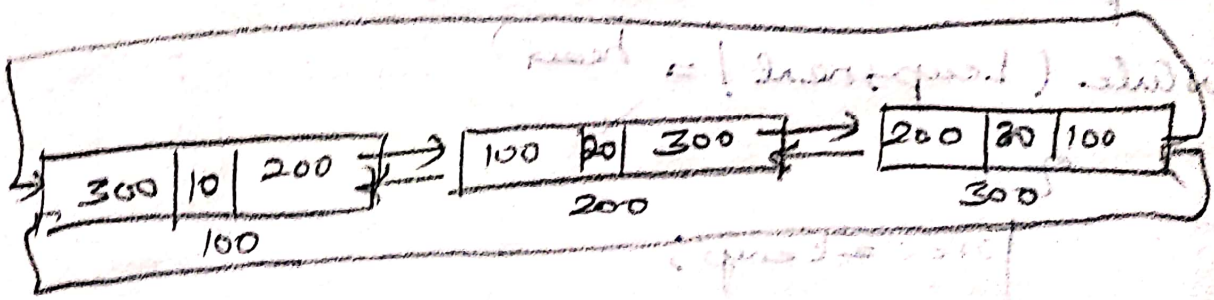
Insertion at beginning

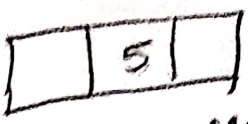
```
temp = head
while ( temp->next != head)
    temp = temp->next

newnode = malloc (sizeof (Structnode));

newnode -> data = 5
```

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





Insertion at end

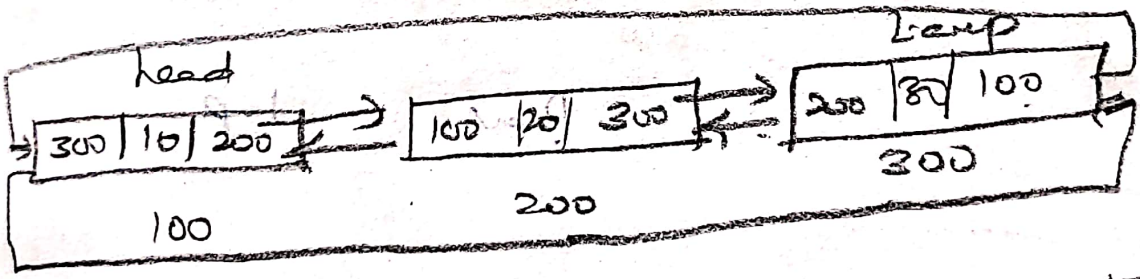
```
struct node *head, *temp, *newnode;
newnode = malloc(sizeof(struct node));
newnode->data = 50;
temp = head;
while(temp->next != head)
    temp = temp->next;
```

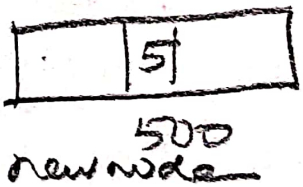
When list is empty

```
if (head == NULL)
    head = newnode;
```

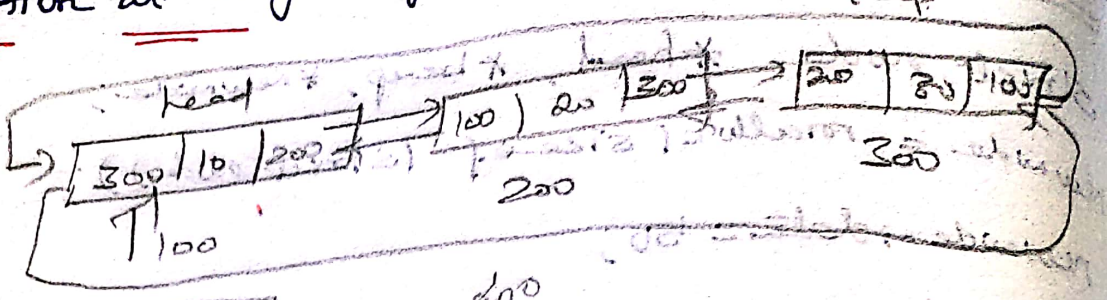
```
newnode->next = head;
newnode->prev = head;
```

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





Deletion at beginning



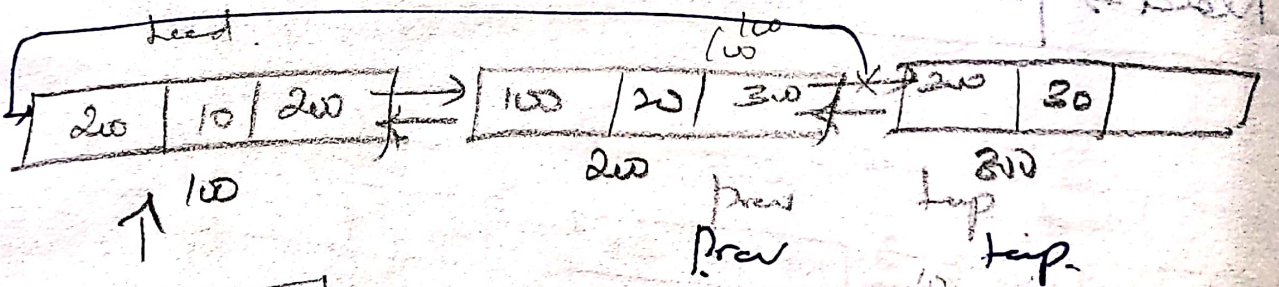
temp → next = head → next;

head → next → prev = temp;

free(head);

head = temp → next;

Deletion at end



Struct Node *prev, *temp;

while(temp → next != NULL)

{

Traversing → prev = temp;

temp = temp → next;

}

prev → next = head;

head → prev = prev

temp → prev = NULL;

free(temp);