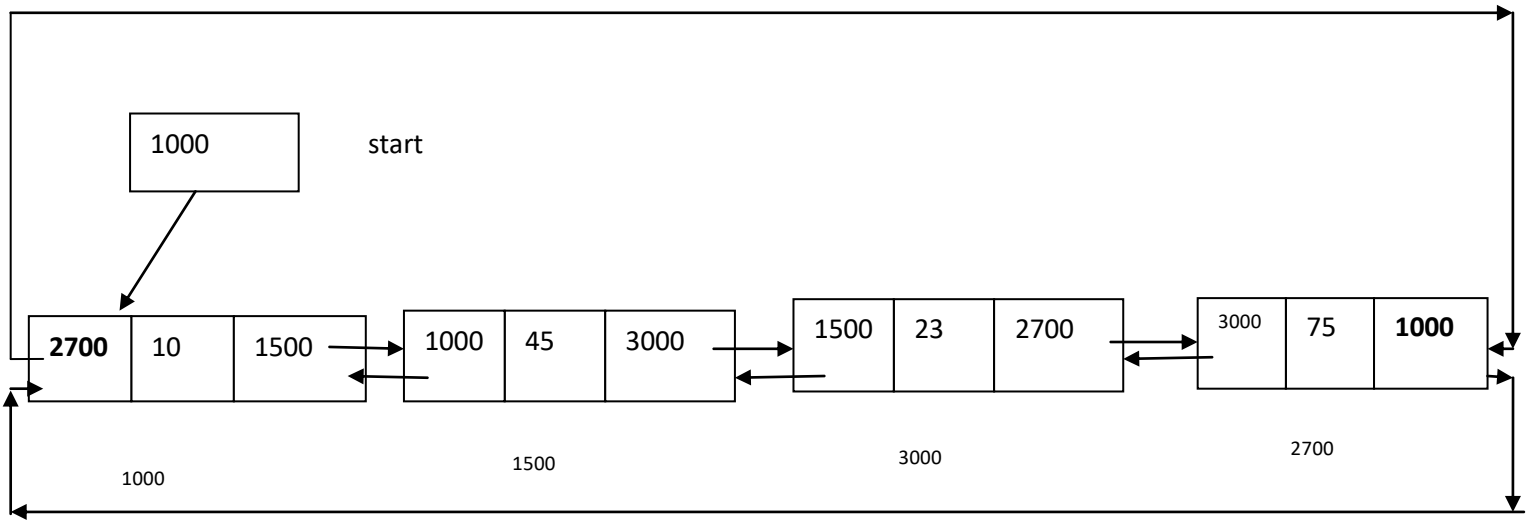
Circular Doubly Linked list in Data structures

The difference between a doubly linked and a circular doubly linked list is same as that exists between a singly linked list and a circular linked list. **The circular doubly linked list does not contain NULL in the previous field of the first node and the next field of the last node. Rather, the next field of the last node stores the address of the first node of the list, i.e., start. Similarly, the previous field of the first field stores the address of the last node.** A circular doubly linked list is one which has both the successor pointer and predecessor pointer in circular manner.



Data type or Structure of a node in the circular double linked list defined as (Same as Double linked list):

```
 struct node
{
    int info;
    struct node *next; // Pointer to next node
    struct node *prev; // Pointer to previous node
};

struct node * start = NULL ;
void display_forward()
{
        struct node *t;
        if(start == NULL)
        {
                printf("List is empty");
                return;
        }
        t = start;
```

```
        do
        {
                printf("%d\t",t->info);
                t=t->next;
        } while(t!= start);
        return;
}
void display_backward()
{
        struct node *t;
        if(start == NULL)
        {
                printf("List is empty");
                return;
        }
        t = start->prev; // t points to last node
        do
        {
                printf("%d\t",t->info);
                t=t->prev;
        }while(t!= start->prev);
        return;
}
```

Inserting a New Node in a Circular Doubly Linked List

Insert first and insert last are different from double linked list. Rest of the cases are similar to that given for doubly linked lists.

```
void insert_first(int ele)
{
  struct node *new = (struct node *)malloc(sizeof(struct node));
  struct node *last;
  if(new == NULL)
  {
     printf("\nOVERFLOW");
     return;
  }
  else
  {
        new->info=ele;
        if(start == NULL)
        {
                new -> next = new;
                new -> prev = new;
```

```
        }
        else
        {
                last = start->prev;  // It points to last node
                last -> next = new;
                new -> next = start;
                new -> prev = last;
                start -> prev = new;
        }
        start = new;
  }
}
void insert_last(int ele)
{
         struct node *new,*last;
         new = (struct node *) malloc (sizeof(struct node));
         if(new == NULL)
         {
                printf("\nOVERFLOW");
         }
        else
        {
                new->info = ele;
                last = start->prev;   // It points to the last node
                new -> next = start;
                new->prev=last;
                 last->next = new;
                start -> prev = new;
        }
}
```

Deleting a Node from a Circular Doubly Linked List

delete first and delete last are different from double linked list. Rest of the cases are same as that given for doubly linked lists.

```
void delete_first()
{
        struct node *temp,*last;
         temp = start;
        if(start = = NULL)
        {
                printf("\n List is empty\n");
                exit(0);
        }

      // only one node present in the list
```

```
        else if(start->next == start)
        {
                start = NULL;
      }
        else
        {
                last = start->prev;
                last -> next = temp -> next;
                start = start->next;
                start -> prev = last;
        }
         free(temp);
 }

void delete_last()
{
         struct node *temp,*last,*t;
        if(start = = NULL)
        {
                 printf("\n List is empty\n");
                exit(0);
         }
         temp = start->prev;
        if(start->prev == start)
        {
                start = NULL;
         }
         else
        {
                t = last->prev; // last but one node
                t -> next = start;
                start->prev = t;
        }
         free(temp);
 }
```