Sairam INSTITUTIONS

Sri SAI RAM ENGINEERING COLLEGE
INSTITUTE OF TECHNOLOGY
West Tambaram, Chennai - 44

YEAR	SEM
II	III

CS8391

DATA STRUCTURES
(Common to CSE & IT)

UNIT NO 1

LINEAR DATA STRUCTURES - LISTS

1.4 CIRCULARLY LINKED LISTS - SINGLE, DOUBLE

1.4.1 SINGLE CIRCULAR LINKED LIST

Version: 1.XX

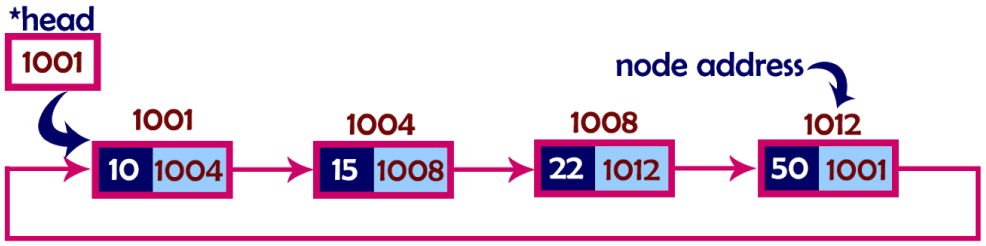
4 QUALITY EDUCATION

NBA NAAC (A+) nirf

CIRCULAR SINGLY LINKED LIST

A circular linked list is a sequence of elements in which every element has a link to its next element in the sequence and the last element has a link to the first element.

Example



STRUCTURE OF A NODE

```
struct Node
{
  int data;
  struct Node *next;
}*head = NULL;
```

OPERATIONS

In a circular linked list, we perform the following operations.

1. Insertion
2. Deletion
3. Display

INSERTION

In a circular linked list, the insertion operation can be performed in three ways. They are as follows.

1. Inserting At Beginning of the list
2. Inserting At End of the list
3. Inserting At Specific location in the list

1. INSERTION AT THE BEGINNING OF THE LIST

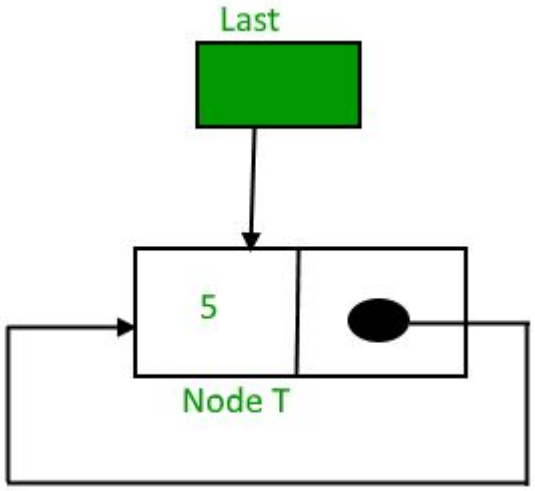
```
void insertAtBeginning(int value)
{
   struct Node *newNode;
   newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode -> data = value;
   if(head == NULL)
   {
     head = newNode;
     newNode -> next = head;
   }
   else
   {
     struct Node *temp = head;
     while(temp -> next != head)
        temp = temp -> next;
     newNode -> next = head;
     head = newNode;
     temp -> next = head;
   }
   printf("\nInsertion success!!!");
}
```

EXAMPLE

Step 1: Assume linked list is empty and hence Head=Null

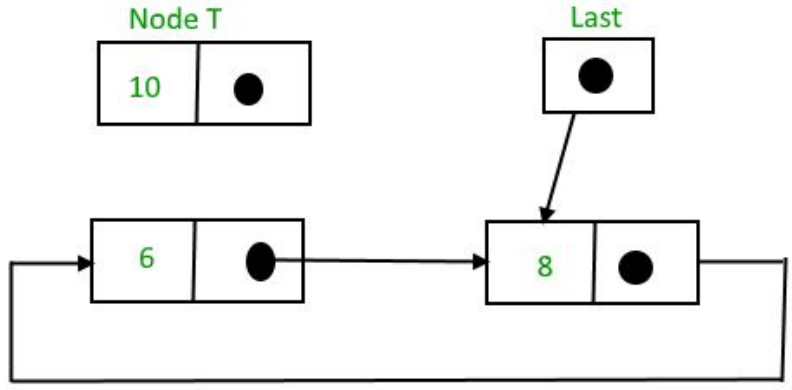
HEAD

Step 2: Insert an element at the beginning of the list when the head is null.

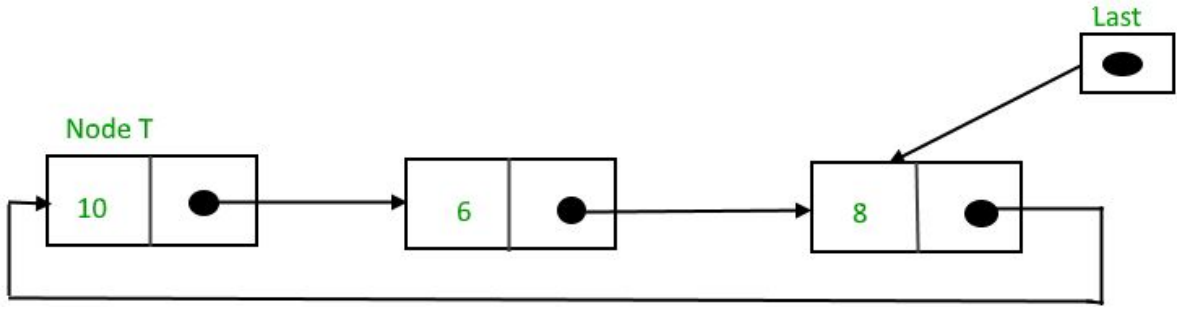


Step 3: Insert an element at the beginning of the list when the head is not null.

Before Insertion



After the insertion

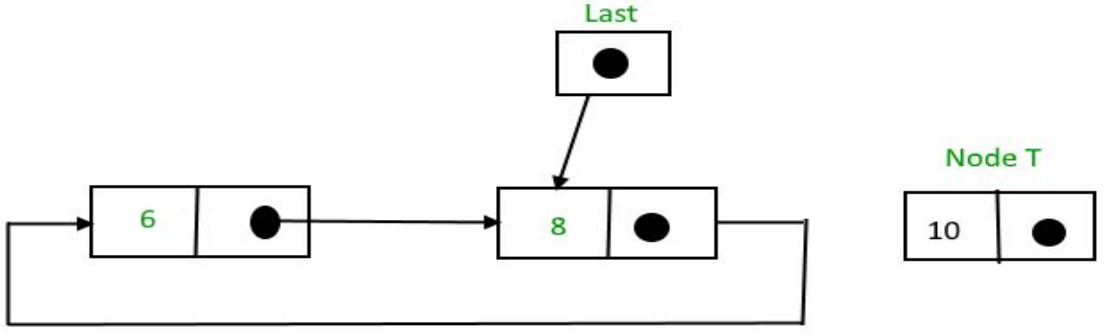


2. 1. INSERTION AT THE END OF THE LIST

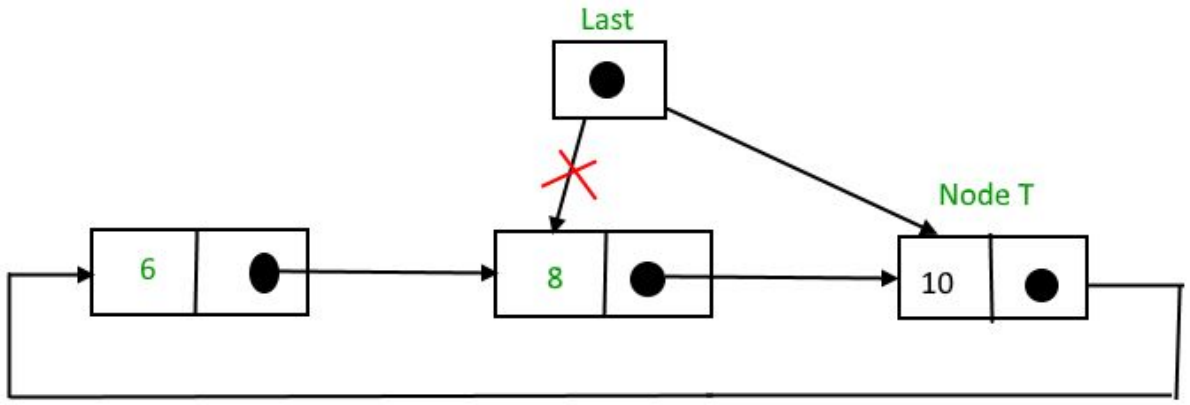
```
void insertAtEnd(int value)
{
  struct Node *newNode;
  newNode = (struct Node*)malloc(sizeof(struct Node));
  newNode -> data = value;
  if(head == NULL)
  {
    head = newNode;
    newNode -> next = head;
  }
  else
  {
    struct Node *temp = head;
    while(temp -> next != head)
      temp = temp -> next;
    temp -> next = newNode;
    newNode -> next = head;
  }
  printf("\nInsertion success!!!");
}
```

Example

Before Insertion



After Insertion



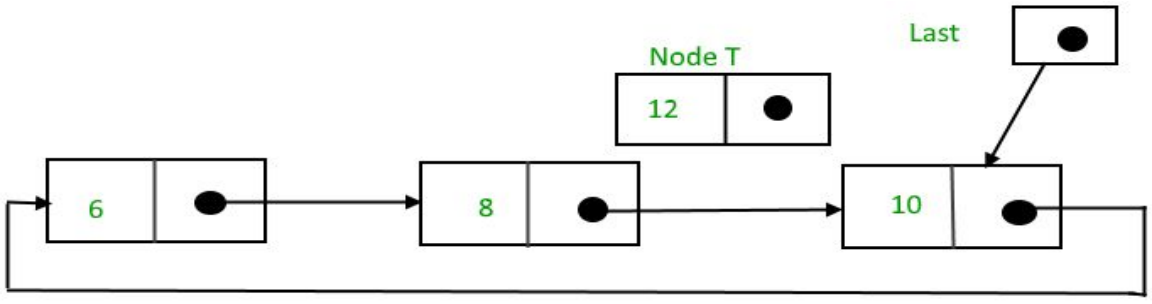
3. INSERTION AT THE SPECIFIC LOCATION OF THE LIST

```
void insertAfter(int value, int location)
{
  struct Node *newNode;
  newNode = (struct Node*)malloc(sizeof(struct Node));
  newNode -> data = value;
  if(head == NULL)
  {
    head = newNode;
    newNode -> next = head;
  }
  else
  {
    struct Node *temp = head;
    while(temp -> data != location)
    {
      if(temp -> next == head)
      {
        printf("Given node is not found in the list!!!");
```

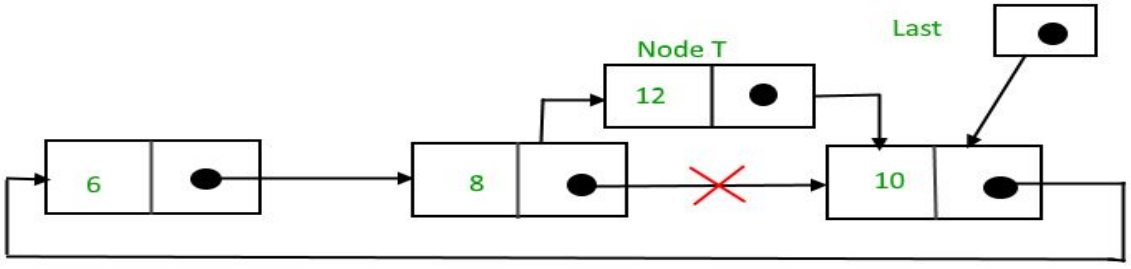
```
            goto EndFunction;
          }
          else
          {
            temp = temp -> next;
          }
        }
        newNode -> next = temp -> next;
        temp -> next = newNode;
        printf("\nInsertion success!!!");
      }
     EndFunction:
}
```

Example

Before Insertion



After Insertion



Deletion

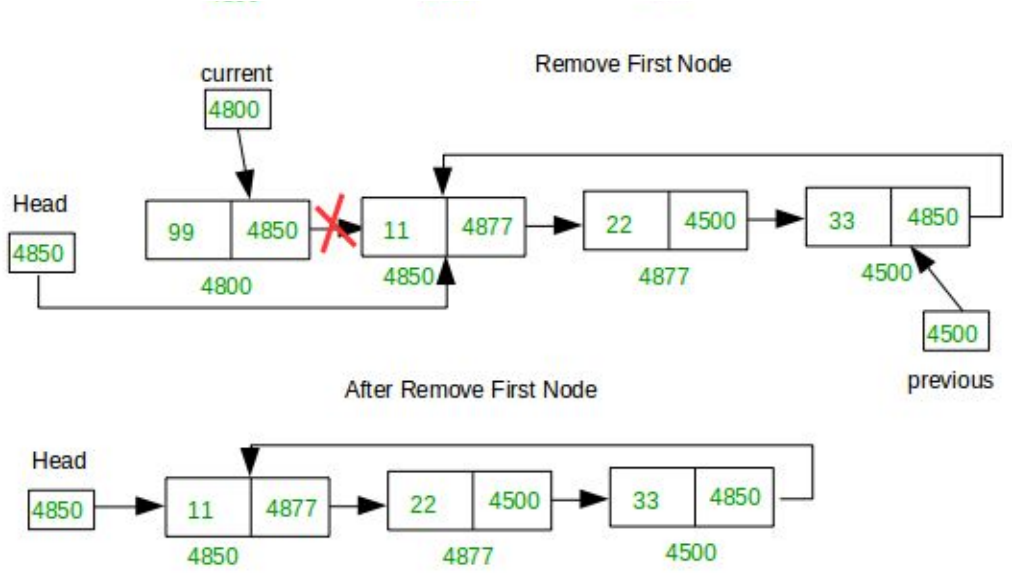
In a circular linked list, the deletion operation can be performed in three ways those are as follows.

1. Deleting from Beginning of the list
2. Deleting from End of the list
3. Deleting a Specific Node

1. Deleting from Beginning of the list

```
void deleteBeginning()
{
  if(head == NULL)
    printf("List is Empty!!! Deletion not possible!!!");
  else
  {
    struct Node *temp = head;
    if(temp -> next == head)
    {
      head = NULL;
      free(temp);
    }
    else{
      head = head -> next;
      free(temp);
    }
    printf("\nDeletion success!!!");
  }
}
```

Example

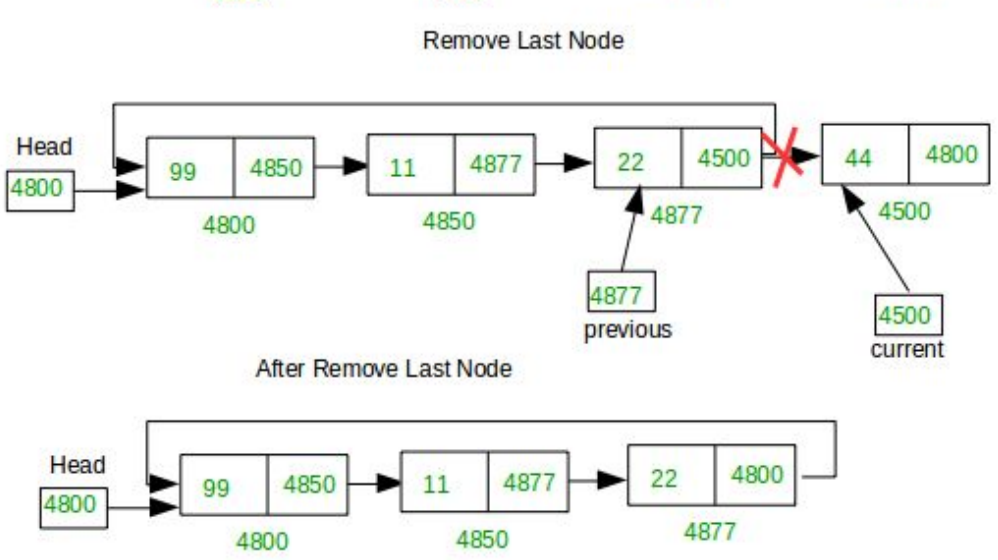


2. Deleting from end of the list

```
void deleteEnd()
{
  if(head == NULL)
    printf("List is Empty!!! Deletion not possible!!!");
  else
  {
    struct Node *temp1 = head, temp2;
    if(temp1 -> next == head)
    {
      head = NULL;
      free(temp1);
    }
    else{
      while(temp1 -> next != head){
        temp2 = temp1;
        temp1 = temp1 -> next;
```

```
        }
        temp2 -> next = head;
        free(temp1);
      }
      printf("\nDeletion success!!!");
    }
}
```

Example



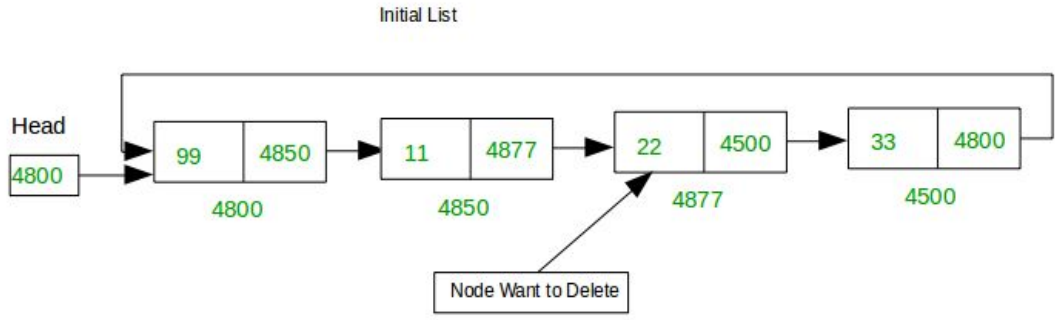
3. Deleting from specific location of the list

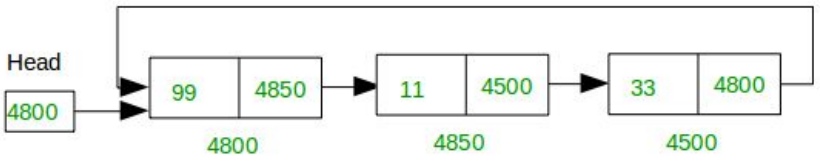
```
void deleteSpecific(int delValue)
{
  if(head == NULL)
    printf("List is Empty!!! Deletion not possible!!!");
  else
  {
    struct Node *temp1 = head, temp2;
```

```
while(temp1 -> data != delValue)
{
  if(temp1 -> next == head)
  {
    printf("\nGiven node is not found in the list!!!");
    goto FuctionEnd;
  }
  else
  {
    temp2 = temp1;
    temp1 = temp1 -> next;
  }
}
if(temp1 -> next == head){
  head = NULL;
  free(temp1);
}
else{
  if(temp1 == head)
  {
    temp2 = head;
    while(temp2 -> next != head)
      temp2 = temp2 -> next;
    head = head -> next;
    temp2 -> next = head;
    free(temp1);
  }
  else
  {
```

```
        if(temp1 -> next == head)
        {           temp2 -> next = head;          }
        else
        {           temp2 -> next = temp1 -> next;          }
        free(temp1);
      }    }
   printf("\nDeletion success!!!");
 }  FuctionEnd:}
```

Example





After Delete index 2 element