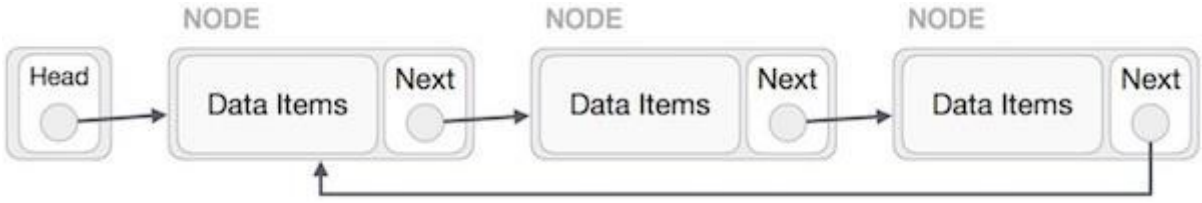
Circular Linked List Data Structure

What is Circular Linked List?

Circular Linked List is a variation of Linked list in which the first element points to the last element and the last element points to the first element. Both Singly Linked List and Doubly Linked List can be made into a circular linked list.

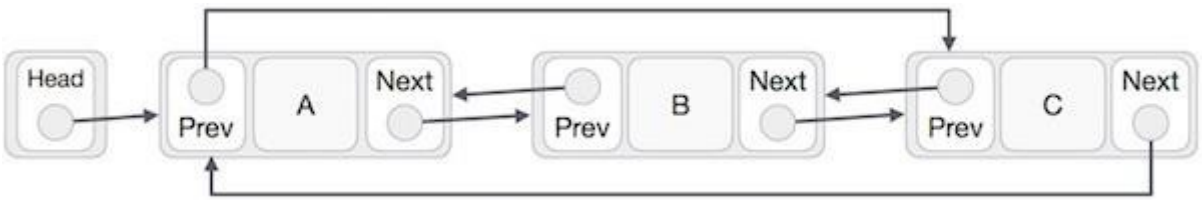
Singly Linked List as Circular

In singly linked list, the next pointer of the last node points to the first node.



Doubly Linked List as Circular

In doubly linked list, the next pointer of the last node points to the first node and the previous pointer of the first node points to the last node making the circular in both directions.



As per the above illustration, following are the important points to be considered.

- The last link's next points to the first link of the list in both cases of singly as well as doubly linked list.
- The first link's previous points to the last of the list in case of doubly linked list.

Basic Operations in Circular Linked List

Following are the important operations supported by a circular list.

- **insert** – Inserts an element at the start of the list.
- **delete** – Deletes an element from the start of the list.
- **display** – Displays the list.

Circular Linked List - Insertion Operation

The insertion operation of a circular linked list only inserts the element at the start of the list. This differs from the usual singly and doubly linked lists as there is no particular starting and ending points in this list. The insertion is done either at the start or after a particular node (or a given position) in the list.

Algorithm

```
1. START
2. Check if the list is empty
3. If the list is empty, add the node and point the head
   to this node
4. If the list is not empty, link the existing head as
   the next node to the new node.
5. Make the new node as the new head.
6. END
```

Example

Following are the implementations of this operation in various programming languages –

C | C++ | Java | Python

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <stdbool.h>
struct node {
   int data;
   int key;
   struct node *next;
};
struct node *head = NULL;
struct node *current = NULL;
```

```
bool isEmpty(){
   return head == NULL;
}

//insert link at the first location
void insertFirst(int key, int data){

   //create a link
   struct node *link = (struct node*) malloc(sizeof(struct node));
   link->key = key;
   link->data = data;
   if (isEmpty()) {
      head = link;
      head->next = head;
   } else {

      //point it to old first node
      link->next = head;

      //point first to new first node
      head = link;
   }
}

//display the list
void printList(){
   struct node *ptr = head;
   printf("\n[ ");

   //start from the beginning
   if(head != NULL) {
      while(ptr->next != ptr) {
         printf("(%d,%d) ",ptr->key,ptr->data);
         ptr = ptr->next;
      }
   }
   printf(" ]");
}
void main(){
   insertFirst(1,10);
   insertFirst(2,20);
   insertFirst(3,30);
```

```
    insertFirst(4,1);
    insertFirst(5,40);
    insertFirst(6,56);
    printf("Circular Linked List: ");

    //print list
    printList();
}
```

Output

```
Circular Linked List:
[ (6,56) (5,40) (4,1) (3,30) (2,20)  ]
```

Circular Linked List - Deletion Operation

The Deletion operation in a Circular linked list removes a certain node from the list. The deletion operation in this type of lists can be done at the beginning, or a given position, or at the ending.

Algorithm

```
1. START
2. If the list is empty, then the program is returned.
3. If the list is not empty, we traverse the list using a
   current pointer that is set to the head pointer and create
   another pointer previous that points to the last node.
4. Suppose the list has only one node, the node is deleted
   by setting the head pointer to NULL.
5. If the list has more than one node and the first node is to
   be deleted, the head is set to the next node and the previous
   is linked to the new head.
6. If the node to be deleted is the last node, link the preceding
   node of the last node to head node.
7. If the node is neither first nor last, remove the node by
   linking its preceding node to its succeeding node.
8. END
```

Example

Following are the implementations of this operation in various programming languages –

C | C++ | Java | Python

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <stdbool.h>
struct node {
   int data;
   int key;
   struct node *next;
};
struct node *head = NULL;
struct node *current = NULL;
bool isEmpty(){
   return head == NULL;
}

//insert link at the first location
void insertFirst(int key, int data){

   //create a link
   struct node *link = (struct node*) malloc(sizeof(struct node));
   link->key = key;
   link->data = data;
   if (isEmpty()) {
      head = link;
      head->next = head;
   } else {

      //point it to old first node
      link->next = head;

      //point first to new first node
      head = link;
   }
}

//delete first item
struct node * deleteFirst(){
```

```
   //save reference to first link
   struct node *tempLink = head;
   if(head->next == head) {
      head = NULL;
      return tempLink;
   }

   //mark next to first link as first
   head = head->next;

   //return the deleted link
   return tempLink;
}

//display the list
void printList(){
   struct node *ptr = head;

   //start from the beginning
   if(head != NULL) {
      while(ptr->next != ptr) {
         printf("(%d,%d) ",ptr->key,ptr->data);
         ptr = ptr->next;
      }
   }
}
void main(){
   insertFirst(1,10);
   insertFirst(2,20);
   insertFirst(3,30);
   insertFirst(4,1);
   insertFirst(5,40);
   insertFirst(6,56);
   printf("Circular Linked List: ");

   //print list
   printList();
   deleteFirst();
   printf("\nList after deleting the first item: ");
   printList();
}
```

Output

```
Circular Linked List: (6,56) (5,40) (4,1) (3,30) (2,20)
List after deleting the first item: (5,40) (4,1) (3,30) (2,20)
```

Circular Linked List - Displaying the List

The Display List operation visits every node in the list and prints them all in the output.

Algorithm

```
1. START
2. Walk through all the nodes of the list and print them
3. END
```

Example

Following are the implementations of this operation in various programming languages –

C | C++ | Java | Python

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <stdbool.h>
struct node {
   int data;
   int key;
   struct node *next;
};
struct node *head = NULL;
struct node *current = NULL;
bool isEmpty(){
   return head == NULL;
}

//insert link at the first location
void insertFirst(int key, int data){
```

```
   //create a link
   struct node *link = (struct node*) malloc(sizeof(struct node));
   link->key = key;
   link->data = data;
   if (isEmpty()) {
      head = link;
      head->next = head;
   } else {

      //point it to old first node
      link->next = head;

      //point first to new first node
      head = link;
   }
}

//display the list
void printList(){
   struct node *ptr = head;
   printf("\n[ ");

   //start from the beginning
   if(head != NULL) {
      while(ptr->next != ptr) {
         printf("(%d,%d) ",ptr->key,ptr->data);
         ptr = ptr->next;
      }
   }
   printf(" ]");
}
void main(){
   insertFirst(1,10);
   insertFirst(2,20);
   insertFirst(3,30);
   insertFirst(4,1);
   insertFirst(5,40);
   insertFirst(6,56);
   printf("Circular Linked List: ");

   //print list
```

```
   printList();
}
```

Output

```
Circular Linked List:
[ (6,56) (5,40) (4,1) (3,30) (2,20)  ]
```

Circular Linked List - Complete Implementation

Following are the complete implementations of Circular Linked List in various programming languages –

C | C++ | Java | Python

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <stdbool.h>
struct node {
   int data;
   int key;
   struct node *next;
};
struct node *head = NULL;
struct node *current = NULL;
bool isEmpty(){
   return head == NULL;
}
int length(){
   int length = 0;
   //if list is empty
   if(head == NULL) {
      return 0;
   }
   current = head->next;
   while(current != head) {
      length++;
      current = current->next;
   }
   return length;
```

```
}
//insert link at the first location
void insertFirst(int key, int data){
   //create a link
   struct node *link = (struct node*) malloc(sizeof(struct node));
   link->key = key;
   link->data = data;
   if (isEmpty()) {
      head = link;
      head->next = head;
   } else {
      //point it to old first node
      link->next = head;

      //point first to new first node
      head = link;
   }
}
//delete first item
struct node * deleteFirst(){

   //save reference to first link
   struct node *tempLink = head;
   if(head->next == head) {
      head = NULL;
      return tempLink;
   }
   //mark next to first link as first
   head = head->next;

   //return the deleted link
   return tempLink;
}
//display the list
void printList(){
   struct node *ptr = head;
   printf("\n[ ");
   //start from the beginning
   if(head != NULL) {
      while(ptr->next != ptr) {
         printf("(%d,%d) ",ptr->key,ptr->data);
         ptr = ptr->next;
```

```
      }
   }
   printf(" ]");
}
int main(){
   insertFirst(1,10);
   insertFirst(2,20);
   insertFirst(3,30);
   insertFirst(4,1);
   insertFirst(5,40);
   insertFirst(6,56);
   printf("Original List: ");
   //print list
   printList();
   while(!isEmpty()) {
      struct node *temp = deleteFirst();
      printf("\nDeleted value:");
      printf("(%d,%d) ",temp->key,temp->data);
   }
   printf("\nList after deleting all items: ");
   printList();
}
```

Output

```
Original List:
[ (6,56) (5,40) (4,1) (3,30) (2,20)  ]
Deleted value:(6,56)
Deleted value:(5,40)
Deleted value:(4,1)
Deleted value:(3,30)
Deleted value:(2,20)
Deleted value:(1,10)
List after deleting all items:
[ ]
```