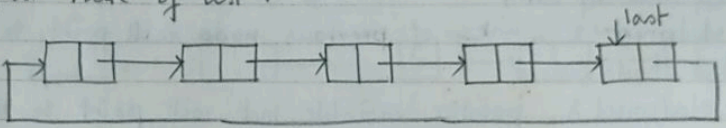
13/2/14

Circular Linked List : Circular linked list is a linked list where we can access any node of the linked list without going back and start traversal again from first node because it's direction is just like circle and we can traverse from last node to first node.

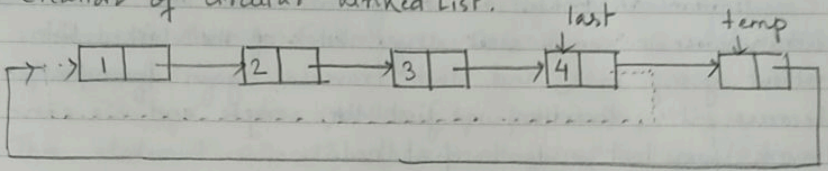
Creation of circular linked list is same as single linked list. In circular linked list, there is no first and last node but conventionally we maintain it. Here last node will always point to first node instead of NULL. So, we maintain one pointer last which points to last node of list and link part of this node points to the first node of list.



Advantage: ① Each node in a circular list can be accesse from any node or The circular list can be traversal from any node.

② While deleting a particular node, we need the address of The previous node to the n to be deleted. So, we have to first traver throughout the linked list to reach to th address and then delete the node next to this address. But in a circular list we ca reach directly to this address and hence reduce the effect of traversing from The start.

Creation of Circular Linked List.



```
temp -> link = last->link ;   /* added at the end
last -> link = temp ;            of list */
last = temp ;
```

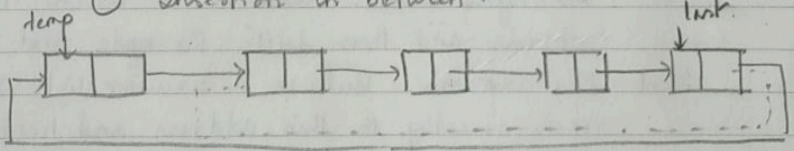
After statement 1, link of added node will point to the first node of list.
After statement 2, link of previous node will point to The added node, and
after statement 3, pointer variable last will point to the added node which is now the last node of list.

Insertion in The Circular Linked list :

It may be possible in two ways :

(i) Insertion at beginning
(ii) insertion in between.

i)



```
temp -> info = data
temp -> link = last ->link ,
last -> link = temp ;
```

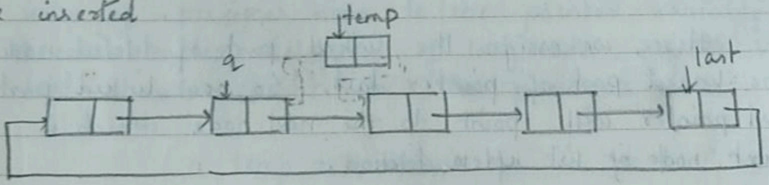
After statement ②, inserted node points to The previou
first node of list.

After statement ③, linked part of last node will point to the inserted node which is first node of circular linked list.

ii) Insertion in between is same as single linked list.
This can be as —

```
temp -> link  = q -> link
temp -> info  = data;
q -> link     = temp;
```

Here q points to the node after which new node will be inserted



Deletion in Circular Linked List :

Deletion from circular linked list is little bit different from linked list. Here, we have a need to take care of some more conditions because of its circular behavior. These are the causes for deletion.

① If list has only one element
② Node to be deleted is the first node of the list
③ Deletion in between
④ Node to be deleted in last node of list.

DATE 14/02/19

Case 1 : Here we check the condition for only one element of list, then assign NULL value to last pointer because after deletion no node will be in list.

```
if (last -> link == last && last -> info == data)
    {
        temp = last ;
        last = NULL ;
        free (temp) ;
    }
```

Case 2 :
Here, we assign the linked part of deleted node to the linked part of pointer last. So, now linked part of last pointer will point to the next node which is now first node of list after deletion.

```
q = last -> link ;
if (q -> info == data)
    { temp = q ;
      last -> link = q -> link ;
      free (temp) ;
    }
```

After statement 1, q is pointing to the first node of list.

Case 3 : Deletion in between is same as in single linked list. Deletion of node in between will be as

```
while (q -> link != last)
    if ( q -> link -> info == data)
        { temp = q -> link ;
          q -> link = temp -> link ;
        } free (temp) ;
                                  q = q -> link ;
```

First we are traversing the list, when we find the element to be deleted, then q points to the previous node. We assign the linked part of node to be deleted to the linked part of previous node. Then we free the address of node to be deleted from memory.

Case 4 : Deletion of last node requires to assign the linked part of last node to the linked part of previous node, so the linked part of previous node will point to the first node of list. Then assign the value of previous node to the pointer variable last because after deletion of last node, pointer variable last should point to the previous node.

```
temp = q -> link ;
q -> link = last -> link ;
free (temp) ;
last = q ;
```

Here q is pointing to the previous node of last node. After statement 1, temp will point to last node. After statement 2, linked part of previous node will point to the first node of the list.
After statement 4, pointer variable last will point to the last node of list.

DATE

Sorted Linked List :

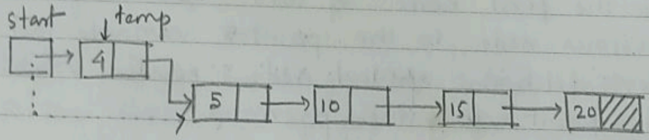
If some element inserted at the proper place in a linked list then this type of linked list is known as sorted order linked list

There will be two cases for adding the element :

i) List is empty or element value is less than the value of first node.

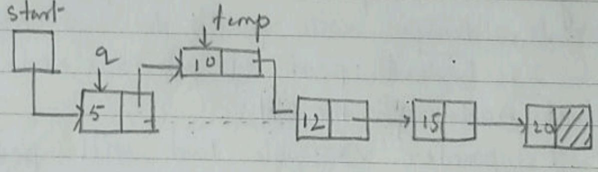
ii) The element value lies in between or at the end of list.

i)



```
if ( start == NULL || num < start -> info)
        {
              temp -> link = start ;
              start = temp ;
        }
```

ii)



```
q = start ;
while (q -> link != NULL && q -> link -> info < num)
          q = q -> link ;
temp -> link = q -> link ;
q -> link = temp ;
```

DATE 12/2/14

Dynamic Memory Allocation : The process of allocating memory at the time of execution is called as dynamic memory allocation. The C language has the facility to allocate memory at the time of execution. The allocation and releasing of this memory space can be done with the use of some built-in-functions which are found in alloc.h header file.

Let us take some functions –

① sizeof ()

② malloc()

③ calloc ()

④ free ()

⑤ realloc ()

① size of () : It is an unary operator. It gives the size of its argument interms of bytes. The argument can be a variable, array or any data type (int, float, char etc). This operator gives the size of any operator.

eg. size of (int); This gives the bytes occupied by the int data type i.e. 2.

② malloc () : This function is used to allocate memory space. The malloc () function reserves a memory space of specified size and gives the starting address to pointer variable. This can be written as – ptr = (datatype *) malloc (specifiedsize);
Here, datatype is the type of pointer and specified size is the size which is required to reserve in memory.

eg. ptr = (int *) malloc (10);

This allocates 10 bytes of memory space to the pointer ptr of type int and the base address is stored in the pointer variable ptr

ptr = (int *) malloc (10 * size of (int));

This allocates the memory space 10 times. ie. hold an int data type. The base address is stored in the pointer variable ptr.

Calloc (): The calloc () function is used to allocate multiple blocks of memory. This has 2 arguments.
eg. ptr = (int *) calloc (5,2);

This allocates 5 blocks of memory, each block contains 2 bytes of memory and the starting address is stored in the pointer variable ptr. which is of type int. The calloc() function is generally used for allocating the memory space for array and structure.

eg
```
struct record
    {   char name [10];
        int age;
        float sal;
    };
int tot_record = 100;
ptr = (record *) calloc (total_rec , sizeof(record));
```

This allocates the memory space for 100 blocks and the blocks contain the memory space. ie occupied by the structure variable record.

free () : For efficient use of memory space we can also release the memory space that is not required. We can use the free() function for releasing the memory space.
eg. free (ptr);

Here, ptr is a pointer variable that contains the base address of memory block, ie. created by malloc() ~~or~~ calloc().

(5) realloc(): For changing the size of the memory block. we can use the function realloc(). This is known as reallocation of memory. eg

ptr= malloc (specified size);

This statement allocate the memory of the specified size and the base address of this memory block are stored in the pointer variable ptr. If we want to change the size of the memory block, then

ptr = realloc (ptr, newsize);

From this statement we can allocate memory space of this newsize and the base address of this memory block is stored in the pointer variable ptr.