1) Useful Concepts

A circular singly linked list is a singly linked list which has the last element linked to the first element in the list. Being circular it really has no ends; then we'll use only one pointer pNode to indicate one element in the list – the newest element. Figure 3.1 show a model of such a list. pNode

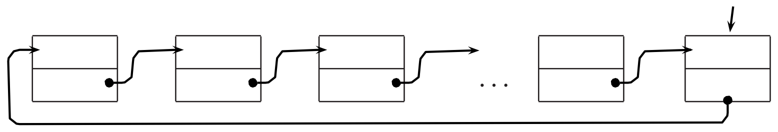

Figure 3.1.: A model of a circular singly-linked list.

```
The structure of a node may be:
struct nodetype
{
int key ; /* an optional f i e l d */
/* other useful data f i e l ds */
struct nodetype *next ;
/* l ink to next node */
};
```

2) Solved Lab Activites

Sr.No	*Allocated Time*	*Level of Complexity*	*CLO Mapping*
Activity 1	***10 mins***	***Medium***	***CLO-4***
Activity 2	***15 mins***	***Medium***	***CLO-4***
Activity 3	***15 mins***	***Medium***	***CLO-4***
Activity 4	***10 mins***	***Medium***	***CLO-4***
Activity 5	***10 mins***	***Medium***	***CLO-4***

Activity 1:

Creation of Circular linked list

Consider the following steps which apply to dynamic lists:

Initially, the list is empty. This can be coded by setting the pointers to the first and last cells to the special value

NULL, i.e. first = NULL, last = NULL.

Reserve space for a new node to be inserted in the list:

```
/*  reserve  space  */

p =  new nodetype ;
```

Then place the data into the node addressed by p. The implementation depends on the type of data the node holds. For primitive types, you can use an assignment such as ∗p = data.

Make the necessary connections:

```
p−>next = NULL;      /*  node  i s  appended  to  the  l i s t  */

i f  (  last  != NULL )  /*  l i s t  i s  not  empty  */

last−>next = p;

else

f i r s t  = p;  /*  f i r s t  node  */

last = p; }
```

Activity 2:

Accessing nodes of Circular linked list

```
NodeT *p;

p = pNode;

i f   (  p != NULL )

do

{

access current node and get data ;

p = p−>next ;

}

while  (  p != pNode ) ;
```

Another choice is to look for a key, say givenKey. Code for such list scan is given below:

```
NodeT *p;

p = pNode;
```

```
i f  (  p != NULL )
do
{
i f  (  p->key = givenKey )
{  /*  key  found  at  address  p  */
return p;
}
p = p->next ;
}
while  (  p != NULL ) ;
return NULL;  /*  not  found  */
}
```

Output

Return the index of Node if the required element is found.

Activity 3:

Insertion in circular linked list

```
Inserting Before a node with key givenKey
There are two steps to execute:
Find the node with key givenKey:
NodeT *p,  *q,  *q1;
q1 = NULL;  /*  i n i t i a l i z e  */
q = pNode;
do
{
q1 = q;
q = q->next ;
i f  (  q->key == givenKey )  break;
}
while (  q != pNode ) ;
Insert the node pointed to by p, and adjust links:
i f  (  q->key == givenKey )
{  /*  node  with  key  givenKey  has  address  q  */
q1->next = p;
p->next = q;
}
Insertion after a node with key
Again, there are two steps to execute:

Find the node with key givenKey:NodeT *p,  *q;
q = pNode;
do
{
i f  (  q->key == givenKey )  break;
q = q->next ;
}
while (  q != pNode ) ;
Insert the node pointed to by p, and adjust links:
i f  (  q->key == givenKey )
{  /*  node  with  key  givenKey  has  address  q  */
p->next = q->next ;
q->next = p;
}
```

Activity 4:

Deleting a node from circular linked list

Again there are two steps to take:

Find the node with key givenKey:

```
NodeT *p,  *q,  *q1;
q = pNode;
do
{
q1 = q;
q = q->next ;
i f  (  q->key == givenKey )  break;

}

while (  q != pNode ) ;

Delete the node pointed to by q. If that node is pNode then we adjust pNode to
point to its previous.

i f  (  q->key == givenKey )

{  /*  node  with  key  givenKey  has  address  q  */

i f  (  q == q->next  )

{

/*  l i s t  now empty  */

}

else

{

q1->next = q->next ;

i f  (  q == pNode )  pNode = q1;

}

free (  q  ) ;

}
```

Activity 5:

Complete deletion of Circular linked list

```
NodeT *p,  *p1;

p = pNode;

do

{

p1 = p;

p = p->next ;

free (  p1  ) ;

}

while  (  p != pNode ) ;

pNode = NULL;
```

Output

All the nodes of Circular Linked List will be deleted.

3) Graded Lab Tasks

Note: The instructor can design graded lab activities according to the level of difficulty and complexity of the solved lab activities. The lab tasks assigned by the instructor should be evaluated in the same lab

Lab Task 1

Write a function that deletes all those nodes from a linked list which have even/odd numbered value in their data part.

Lab Task 2

Write a function that implements Josephus problem.

Lab Task 3

Write a function that deletes all even positioned nodes from a linked list. Last node should also be deleted if its position is even.