SNS COLLEGE OF ENGINEERING

Kurumbapalayam (Po), Coimbatore – 641 107

An Autonomous Institution

Accredited by NBA – AICTE and Accredited by NAAC – UGC with 'A' Grade
Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING-IOT Including Cyber Security &BCT

COURSE NAME : 19ITT201- DATA STRUCTURES

II YEAR / III SEMESTER

Unit 1- LINEAR STRUCTURES AND TREES

Topic 6 : Circular Linked List Based Implementation

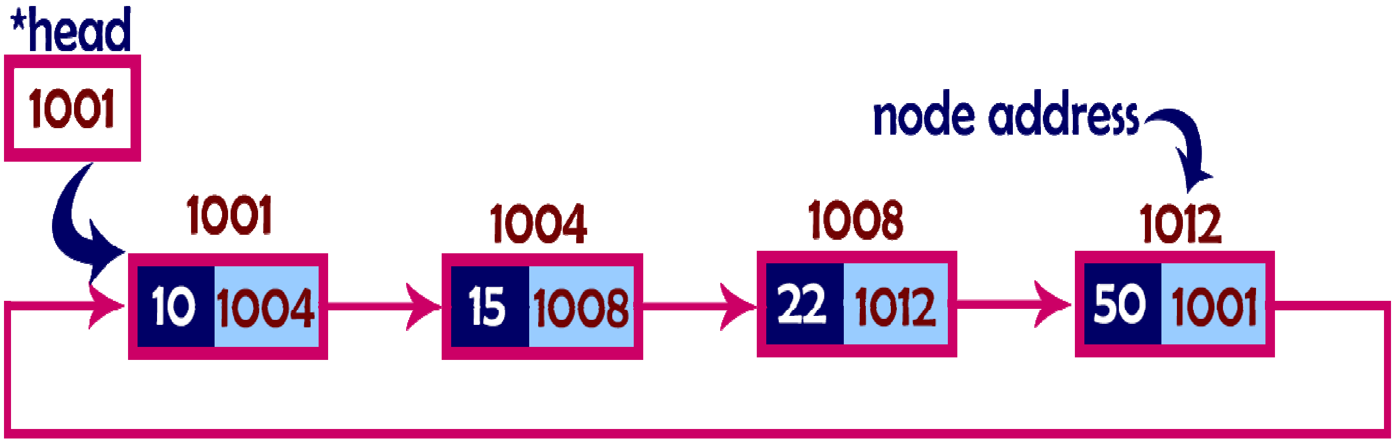
Problem

- Depending on implementation, inserting at start of list would require doing a search for the last node which could be expensive.
- Finding end of list and loop control is harder (no NULL's to mark beginning and end)

Circular Linked List Based Implementation

➢Define Circular 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.



Circular Linked List Based Implementation –Cont..

Basic Operations

Following are the basic operations supported by a list.

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

- ➢Inserting At Beginning of the list
- ➢Inserting At End of the list
- ➢Inserting At Specific location in the list

Circular Linked List Based Implementation–**Cont..**

1. Creation of list

A linked list node

```
 struct Node {
    int data;
    Node* next;
    Node(int x)
    {
        data = x;
        next = NULL;
    }
};
```

Circular   Linked List Based Implementation

Add a node at the front.

We can use the following steps to insert a new node at beginning of the circular linked list...

Step 1 - Create a **newNode** with given value.

Step 2 - Check whether list is **Empty** (**head == NULL**)

Step 3 - If it is **Empty** then,

set **head** = **newNode** and **newNode→next** = **head** .

Step 4 - If it is **Not Empty** then, define a Node pointer '**temp**' and initialize with '**head**'.

Step 5 - Keep moving the '**temp**' to its next node until it reaches to the last node (until '**temp** → **next == head**').

Step 6 - Set '**newNode** → **next =head**', '**head** = **newNode**' and '**temp** → **next** = **head**'.

Circular Linked List Based Implementation –Cont..

```
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!!!"); }
```

Circular **Linked List Based Implementation –Cont..**

2) We can use the following steps to insert a new node after a node in the circular linked list...

Step 1 - Create a **newNode** with given value.

Step 2 - Check whether list is **Empty** (**head** == **NULL**)

Step 3 - If it is **Empty** then, set **head** = **newNode** and **newNode** → **next** = **head**.

Step 4 - If it is **Not Empty** then, define a node pointer **temp** and initialize with **head**.

Step 5 - Keep moving the **temp** to its next node until it reaches to the node after which we want to insert the newNode (until **temp1** → **data** is equal to **location**, here location is the node value after which we want to insert the newNode).

Step 6 - Every time check whether **temp** is reached to the last node or not. If it is reached to last node then display **'Given node is not found in the list!!! Insertion not possible!!!'** and terminate the function. Otherwise move the **temp** to next node.

Step 7 - If **temp** is reached to the exact node after which we want to insert the newNode then check whether it is last node (temp → next == head).

Step 8 - If **temp** is last node then set **temp** → **next** = **newNode** and **newNode** → **next** = **head**.

Step 8 - If **temp** is not last node then set **newNode** → **next** = **temp** → **next** and **temp** → **next** = **newNode**.

Circular **Linked List Based Implementation –Cont..**

```
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:
 }
```

Circular **Linked List Based Implementation –Cont..**

3) Add a node at the end.

Inserting At End of the list

We can use the following steps to insert a new node at end of the circular linked list...

Step 1 - Create a **newNode** with given value.

Step 2 - Check whether list is **Empty** (**head == NULL**).

Step 3 - If it is **Empty** then, set **head** = **newNode** and **newNode** → **next** = **head**.

Step 4 - If it is **Not Empty** then, define a node pointer **temp** and initialize with **head**.

Step 5 - Keep moving the **temp** to its next node until it reaches to the last node in the list (until **temp** → **next** == **head**).

Step 6 - Set **temp** → **next** = **newNode** and **newNode** → **next** = **head**.

Circular Linked List Based Implementation –Cont..

```
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!!!");
 }
```

Activity

Advantages

1. Any node can be a starting point. We can traverse the whole list by starting from any point. We just need to stop when the first visited node is visited again.
2. Useful for implementation of queue. Unlike this implementation, we don’t need to maintain two pointers for front and rear if we use circular linked list
3. Circular Doubly Linked Lists are used for implementation of advanced data structures like Fibonacci Heap.

Disadvantages

1. It is not easy to reverse the linked list.
2. If proper care is not taken, then the problem of infinite loop can occur.
3. If we at a node and go back to the previous node, then we can not do it in single step.
4. Instead we have to complete the entire circle by going through the in between nodes and then we will reach the required node.

Assessment 1

1. List out the advantages of circular linked list based implementation

a)________________________________

b)________________________________

c)________________________________

d)________________________________

2. Identify the disadvantages of circular linked list based implementation

a)________________________________

b)________________________________

c)________________________________

d)________________________________

REFERENCES

1. M. A. Weiss, “Data Structures and Algorithm Analysis in C”, Pearson Education, 8th Edition, 2007. [Unit I, II, III, IV,V]

2. A. V. Aho, J. E. Hopcroft and J. D. Ullman, “Data Structures and Algorithms”, Pearson Education, 2nd Edition, 2007 [Unit IV].

3. A.M.Tenenbaum, Y. Langsam and M. J. Augenstein, “Data Structures using C”,PearsonEducation, 1st Edition, 2003.(UNIT I,II,V)

4.https://www.youtube.com/watch?v=0xoYNbVTiSE&t=256s

THANK YOU