Department of Computer Science and Engineering

PES UNIVERSITY

UE19CS202: Data Structures and its Applications (4-0-0-4-4)

CIRCULAR DOUBLY LINKED LIST

Abstract

Circular Double Linked List – overview and pseudo code of various operations

Vandana M Ladwani

vandanamd@pes.edu

Contents

Overview

A circular doubly linked list has both successor pointer and predecessor pointer in circular manner that is the last and first node are connected. The objective behind considering circular double linked list is to simplify the insertion and deletion operations performed on double linked list. In circular double linked list the rlink link of the right most node points back to the head node and llink link of the first node points to the last node.
The basic operations in a circular double linked list are:

- Creation
- Insertion
- Deletion
- Traversing

Operations

Inserting a node at the beginning

The following steps are to be followed to insert a new node at the beginning of the list:

- Get the new node using createnode().
  new_node=createnode();
- If the list is empty, then
  head = new_node;
  new_node -> llink = head;
  new_node -> rlink = head;
- If the list is not empty, follow the steps given below:
  new_node -> llink = head -> llink;
  new_node -> rlink = head;
  head -> llink -> rlink = new_node;
  head -> llink = new_node;
  head = new_node;

Inserting a node at the end

The following steps are followed to insert a new node at the end of the list:

- Get the new node using createnode()
  new_node=createnode();
- If the list is empty, then
  head = new_node;

```
new_node -> llink = head;
new_node -> rlink = head;
```

- If the list is not empty follow the steps given below:

```
new_node -> llink = head -> llink;
new_node -> rlink = head;
head -> llink -> rlink = new_node;
head -> llink = new_node;
```

Inserting a node at an intermediate position

The following steps are followed, to insert a new node in an intermediate position in the list:

- Get the new node using createnode().
  new_node=createnode();
- Ensure that the specified position is in between first node and last node. If not, specified position is invalid. This is done by countnode() function.
- Store the heading address (which is in head pointer) in temp. Then traverse the temp pointer upto the specified position.
- After reaching the specified position, follow the steps given below:

```
new_node -> llink = temp;
new_node -> rlink = temp -> rlink;
temp -> rlink -> llink = new_node;
 temp -> rlink = new_node;
 nodectr++;
```

Deleting a node at the beginning

The following steps are followed, to delete a node at the beginning of the list:

- If list is empty then display ‘Empty List’ message.
- If the list is not empty, follow the steps given below:

```
temp = head;
head = head -> rlink; // second node
temp -> llink -> rlink = head;// temp->llink is last node
head -> llink = temp -> llink;
free(temp)
```

Deleting a node at the end

The following steps are followed to delete a node at the end of the list:

- If list is empty then display 'Empty List' message
- If the list is not empty, follow the steps given below:

```
cur=head->llink            // last node address
cur->llink->rlink=head // cur->llink is second last node
head->llink=cur->llink
free(cur)
```

Deleting a node at Intermediate position:

The following steps are followed, to delete a node from an intermediate position in the list (List must contains more than two node).

- If list is empty then display 'Empty List' message.
- If the list is not empty, follow the steps given below:
- Get the position of the node to delete.
- Ensure that the specified position is in between first node and last node. If not, specified position is invalid.
- Then perform the following steps:

```
if(pos > 1 && pos < nodectr)
    temp = head;
    i = 1;
    while(i < pos)
        temp = temp -> rlink ;
        i++;
    temp -> rlink -> llink = temp -> llink;
    temp -> llink -> rlink = temp -> rlink;
    free(temp);
    nodectr--;
```