TUTORIAL XI:

Circular Linked List Implementation

*U19CS012 [D-12]*

Implement the following operations in context to ***Circular linked list***:

*1) Creation*

*2) Insertion (at beginning, middle and end)*

*3) Deletion (from beginning, middle and end)*

*---------------------Additional Functions in Assignment------------------------*

4) Exchange First and Last Node of List

5) Delete Alternating Nodes of Circular Linked List

6.) Split Linked List into Two Halves

Code:

*// Implement the following operations in context to Circular linked list:*

*// 1) Creation*

*// 2) Insertion (at begining, middle and end)*

*// 3) Deletion (from begining, middle and end)*

*#include* <stdio.h>

*// For Exit Function*

*#include* <stdlib.h>

*// Structure for Each Node*

struct node

{

    int data;

    struct node \*next;

};

*//Helper Functions*

*// 1 -> Creation of Circular Linked List*

*// Creation of the Circular Linked List*

void CREATION\_CLL();

*// Display of the Whole Circular Linked List*

void DISPLAY\_CLL();

*// Returns the Length of Circular Linked List*

int LENGTH\_CLL();

*// 2 -> Insertion in Circular Linked List*

*// Insert at the Beginning of Circular Linked List*

void Insert\_Begin();

*// Insert at the End of Circular Linked List*

void Insert\_Last();

*// Insert in the Middle Of the Circular Linked List*

void Insert\_Middle();

*// 3 -> Deletion in the Circular Linked List*

*// Delete at the Beginning of Circular Linked List*

void Delete\_Begin();

*// Delete at the End of Circular Linked List*

void Delete\_Last();

*// Delete in the Middle Of the Circular Linked List*

void Delete\_Middle();

*// 1 -> Deletes Node at Particular Position*

void Delete\_Position();

*// 2 -> Deletes all Nodes with Particular Value*

void Delete\_Value();

*// Exchange the First and Last Node*

struct node \*Exchange\_First\_And\_Last(struct node \*head);

*//Delete Alternating Node*

void Delete\_Alternating\_Node(struct node \*head);

*// Split List into Two Halves*

void Split\_CLL(struct node \*head);

*// head Pointer -> head of Linked List*

struct node \*head = NULL;

*// For Splitted Head Pointers*

struct node \*head1 = NULL;

struct node \*head2 = NULL;

int main()

{

    int choice;

    printf("\nCIRCULAR LINKED LIST\n");

    printf(" 1 -> Create a Circular Linked List\n");

    printf(" 2 -> Display the Circular Linked List\n");

    printf(" 3 -> Insert at the Beginning of Circular Linked List\n");

    printf(" 4 -> Insert at the End of Circular Linked List\n");

    printf(" 5 -> Insert at Middle of Circular Linked List\n");

    printf(" 6 -> Delete from Beginning\n");

    printf(" 7 -> Delete from the End\n");

    printf(" 8 -> Delete at Middle of Circular Linked List\n");

    printf(" 9 -> Exchange First and Last Node of List\n");

    printf(" 10 -> Delete Alternating Nodes of Circular Linked List\n");

    printf(" 11 -> Split Linked List into Two Halves\n");

    printf(" 12 -> Exit\n");

*while* (1)

    {

        printf("Enter your choice : ");

        scanf("%d", &choice);

*switch* (choice)

        {

*case* 1:

            CREATION\_CLL();

*break*;

*case* 2:

            DISPLAY\_CLL(head);

*break*;

*case* 3:

            Insert\_Begin();

*break*;

*case* 4:

            Insert\_Last();

*break*;

*case* 5:

*// Insert at Middle of Circular lL*

            Insert\_Middle();

*break*;

*case* 6:

            Delete\_Begin();

*break*;

*case* 7:

            Delete\_Last();

*break*;

*case* 8:

*// Delete at Middle of Circular ll*

            Delete\_Middle();

*break*;

*case* 9:

            head = Exchange\_First\_And\_Last(head);

*break*;

*case* 10:

            Delete\_Alternating\_Node(head);

*break*;

*case* 11:

            Split\_CLL(head);

            printf("The Splitted Lists Are :\n");

            DISPLAY\_CLL(head1);

            DISPLAY\_CLL(head2);

*break*;

*case* 12:

            exit(0);

*break*;

*default*:

            printf("Enter a Valid Choice!");

*break*;

        }

    }

*return* 0;

}

*// Creation of the Circular Linked List*

void CREATION\_CLL()

{

    struct node \*ptr, \*temp;

    int item;

*// Allocate Memory for One Node*

    ptr = (struct node \*)malloc(sizeof(struct node));

*if* (ptr == NULL)

    {

        printf("No Memory Space on Device!\n");

*return*;

    }

*else*

    {

*// Creation of New Node { [?]->NULL }*

        printf("Enter the Data to be stored in Node : ");

        scanf("%d", &item);

        ptr->data = item;

*if* (head == NULL)

        {

*// If the Linked List is Empty*

            head = ptr;

*// Since its Circular Linked List*

            ptr->next = head;

        }

*else*

        {

*// If the Linked List is Not Empty*

            temp = head;

*// Traverse and Point temp to Rear of Circular Link List*

*while* (temp->next != head)

                temp = temp->next;

*// Node should Point to head*

            ptr->next = head;

*// Rear Element next should Point to "New Node"*

            temp->next = ptr;

*// Since "New Node" is Our New Head [Start]*

            head = ptr;

        }

*// printf("Node Inserted!\n");*

    }

}

*// Display of the Whole Circular Linked List*

void DISPLAY\_CLL(struct node \*h1)

{

    struct node \*ptr;

*if* (h1 == NULL)

    {

        printf("List is Empty!!\n");

*return*;

    }

*else*

    {

*// Head Pointer*

        ptr = h1;

        printf("Elements of List : ");

*while* (ptr->next != h1)

        {

            printf("%d -> ", ptr->data);

            ptr = ptr->next;

        }

        printf("%d -> ", ptr->data);

        printf("HEAD\n");

    }

}

*// Returns the Length of Circular Linked List*

int LENGTH\_CLL()

{

    struct node \*ptr;

*if* (head == NULL)

    {

*return* 0;

    }

*else*

    {

*// Head Pointer*

        int cnt = 0;

        ptr = head;

*while* (ptr->next != head)

        {

            cnt++;

            ptr = ptr->next;

        }

*return* cnt + 1;

    }

}

*// Insert at the Beginning of Circular Linked List*

void Insert\_Begin()

{

    struct node \*ptr, \*temp;

    int item;

*// Allocate Memory for One Node*

    ptr = (struct node \*)malloc(sizeof(struct node));

*if* (ptr == NULL)

    {

        printf("No Memory Space on Device!\n");

*return*;

    }

*else*

    {

*// Creation of New Node { [?]->NULL }*

        printf("Enter the Data to be stored in Node : ");

        scanf("%d", &item);

        ptr->data = item;

*if* (head == NULL)

        {

*// If the Linked List is Empty*

            head = ptr;

*// Since its Circular Linked List*

            ptr->next = head;

        }

*else*

        {

*// If the Linked List is Not Empty*

            temp = head;

*// Traverse and Point temp to Rear of Circular Link List*

*while* (temp->next != head)

                temp = temp->next;

*// Node should Point to head*

            ptr->next = head;

*// Rear Element next should Point to "New Node"*

            temp->next = ptr;

*// Since "New Node" is Our New Head [Start]*

            head = ptr;

        }

*// printf("Node Inserted!\n");*

    }

}

*// Insert at the End of Circular Linked List*

void Insert\_Last()

{

    struct node \*ptr, \*temp;

    int item;

*// Allocate Memory for One Node*

    ptr = (struct node \*)malloc(sizeof(struct node));

*if* (ptr == NULL)

    {

        printf("No Memory Space on Device!\n");

*return*;

    }

*else*

    {

*// Creation of New Node { [?]->NULL }*

        printf("Enter the Data to be stored in Node : ");

        scanf("%d", &item);

        ptr->data = item;

*if* (head == NULL)

        {

*// If the Linked List is Empty*

            head = ptr;

*// Since its Circular Linked List*

            ptr->next = head;

        }

*else*

        {

*// If the Linked List is Not Empty*

            temp = head;

*// Traverse and Point temp to Rear of Circular Link List*

*while* (temp->next != head)

                temp = temp->next;

*// Rear Element next shoudl Point to "New Node"*

            temp->next = ptr;

*// Node should Point to head*

            ptr->next = head;

        }

*// printf("Node Inserted!\n");*

    }

}

*// Delete at the Beginning of Circular Linked List*

void Delete\_Begin()

{

*// temporary pointer to store old head*

    struct node \*ptr;

*if* (head == NULL)

    {

        printf("List is Empty! No Deletion Possible!!\n");

*return*;

    }

*else* *if* (head->next == head)

    {

*// Only One Element in Circular Linked List*

        head = NULL;

        free(head);

        printf("Node Deleted!\n");

    }

*else*

    {

        ptr = head;

*// Traverse and Point ptr to Rear of Circular Link List*

*while* (ptr->next != head)

            ptr = ptr->next;

*// Point the Rear Element of CLL to Second Element of CLL*

        ptr->next = head->next;

*// free(head);*

*// head is pointing to Second Element of CLL*

        head = ptr->next;

        printf("Node Deleted!\n");

    }

}

*// Delete at the End of Circular Linked List*

void Delete\_Last()

{

    struct node \*ptr, \*preptr;

*if* (head == NULL)

    {

        printf("List is Empty! No Deletion Possible!!\n");

*return*;

    }

*else* *if* (head->next == head)

    {

*// Only One Element in Circular Linked List*

        head = NULL;

        free(head);

        printf("Node Deleted!\n");

    }

*else*

    {

        ptr = head;

*// Traverse and Point ptr to Rear of Circular Link List*

*// Traverse and Point preptr to One Element Before Rear of Circular Link List*

*while* (ptr->next != head)

        {

            preptr = ptr;

            ptr = ptr->next;

        }

*// Second Rear Element Should Point to Next of Rear*

*// Therby Deleting Rear Element of CLL*

        preptr->next = ptr->next;

*// free(ptr);*

        printf("Node Deleted!\n");

    }

}

void display()

{

    struct node \*ptr;

    ptr = head;

*if* (head == NULL)

    {

        printf("\nnothing to print");

    }

*else*

    {

        printf("\n printing values ... \n");

*while* (ptr->next != head)

        {

            printf("%d\n", ptr->data);

            ptr = ptr->next;

        }

        printf("%d\n", ptr->data);

    }

}

*// Insertion at Middle of Linked List*

void Insert\_Middle()

{

    struct node \*ptr, \*temp;

    int i, pos;

    temp = (struct node \*)malloc(sizeof(struct node));

*if* (temp == NULL)

    {

        printf("No Memory Space on Device!\n");

*return*;

    }

    printf("Enter the Position for the New Node to be Inserted : ");

    scanf("%d", &pos);

*// pos = 1 -> Insertion at Beginning of LL*

*// pos = len + 1 -> Insertion at Ending of LL*

*// Length of the Linked Lst*

    int len = LENGTH\_CLL();

*if* (pos <= 0 || pos > len + 1)

    {

        printf("Enter Valid Postion for Insertion!\n");

*return*;

    }

*// Creation of New Node { [?]->NULL }*

    printf("Enter the Data to be stored in Node : ");

    scanf("%d", &temp->data);

    temp->next = NULL;

*if* (pos == 1)

    {

*// At the Beginning of Linked List*

        temp->next = head;

        head = temp;

    }

*else*

    {

*for* (i = 1, ptr = head; i < pos - 1; i++)

        {

            ptr = ptr->next;

        }

*// temp is also pointing to next of "pos" to be inserted*

        temp->next = ptr->next;

*// Make ptr Point to Temp*

        ptr->next = temp;

    }

}

*// Delete in the Middle Of the Circular Linked List*

void Delete\_Middle()

{

*if* (head == NULL)

    {

        printf("List is Empty! No Deletion Possible!!\n");

        exit(0);

    }

*else*

    {

        int ch = 0;

        printf("Delete A Node By : \n");

        printf(" 1 -> Position\n");

        printf(" 2 -> Value\n");

        printf("Enter Your Choice : ");

        scanf("%d", &ch);

*switch* (ch)

        {

*case* 1:

            Delete\_Position();

*break*;

*case* 2:

            Delete\_Value();

*break*;

*default*:

            printf("Enter a Valid Choice!\n");

*break*;

        }

    }

}

*// 1 -> Deletes Node at Particular Position*

void Delete\_Position()

{

    int i, pos;

    struct node \*temp, \*ptr;

    printf("Enter the Position of the Node to be Deleted : ");

    scanf("%d", &pos);

*// pos = 1 -> Deletion at Beginning of LL*

*// pos = len -> Deletion at Ending of LL*

*// Length of the Circular Linked Lst*

    int len = LENGTH\_CLL();

*// printf("LENGTH : %d\n", len);*

*if* (pos <= 0 || pos > len)

    {

        printf("Enter Valid Postion for Deletion!\n");

*return*;

    }

*if* (pos == 1)

    {

        Delete\_Begin();

*return*;

    }

*else*

    {

*if* (pos == len)

        {

            Delete\_Last();

*return*;

        }

        ptr = head;

*for* (i = 1; i < pos; i++)

        {

            temp = ptr;

            ptr = ptr->next;

        }

*// point the prev of {element to be deleted} to "next of deleted"*

*// []     []     []*

*//temp    ptr*

        temp->next = ptr->next;

        printf("The Deleted Element is : %d\n", ptr->data);

        free(ptr);

    }

}

*// 2 -> Deletes all Nodes with Particular Value*

void Delete\_Value()

{

    int value;

    struct node \*temp, \*ptr;

    printf("Enter the Value of the Node to be Deleted : ");

    scanf("%d", &value);

    int flag = 0;

*if* (head == NULL)

    {

        printf("List is Empty!No Deletions Possible\n");

*return*;

    }

*else*

    {

*// Head Pointer*

        ptr = head;

*while* (ptr->next != head)

        {

*// If the Value of Node = Value of Node to be Deleted*

*if* (ptr->data == value)

            {

*if* (ptr == head)

                {

                    Delete\_Begin();

                    flag = 1;

                }

*else*

                {

*if* (ptr->next == head)

                    {

                        Delete\_Last();

                        flag = 1;

                    }

*else*

                    {

                        temp->next = ptr->next;

                        flag = 1;

                    }

*// printf("The Deleted Element is : %d\n", ptr->data);*

                }

            }

*// temp stored old node's address*

            temp = ptr;

*// ptr now points to next node*

            ptr = ptr->next;

        }

*if* (flag == 0)

        {

            printf("Node with Given Value Does Not Exist! OR Deleted Earlier!\n");

        }

*else*

        {

            printf("Node with Given Value Found and Deleted Succesfully!\n");

        }

    }

}

*// Exchange the First and Last Node*

struct node \*Exchange\_First\_And\_Last(struct node \*head)

{

*// TASK 1 : Find pointer to previous of last node*

*// Declare a tmp Pointer*

    struct node \*tmp = head;

*// Iterate till it Points to the Previous of the Last Node*

*while* (tmp->next->next != head)

        tmp = tmp->next;

*//  Exchange first and last nodes using head and tmp*

*// Link Allocation*

    tmp->next->next = head->next;

    head->next = tmp->next;

    tmp->next = head;

    head = head->next;

*return* head;

}

*//Delete Alternating Node*

void Delete\_Alternating\_Node(struct node \*head)

{

*if* (head == NULL)

*return*;

    int length = LENGTH\_CLL();

*// Initialize prev and node to be deleted*

    struct node \*prev = head;

    struct node \*tmpnode = head->next;

*while* (prev->next != head && tmpnode->next != head)

    {

*// Change next link of previous node by Skipping its Next Node*

        prev->next = tmpnode->next;

*// Free memory*

        free(tmpnode);

*// Update prev and node*

        prev = prev->next;

        tmpnode = prev->next;

    }

*if* (length % 2 == 0)

    {

*// Last Node Needs to deleted in Even Length CLL*

        Delete\_Last();

    }

*else*

    {

        printf("Nodes Deleted!\n");

    }

}

*// Split List into Two Halves*

void Split\_CLL(struct node \*head)

{

    struct node \*tortoise = head;

    struct node \*hare = head;

*if* (head == NULL)

*return*;

*while* (hare->next != head && hare->next->next != head)

    {

        hare = hare->next->next;

        tortoise = tortoise->next;

    }

*// Even Elements in the Linked List*

*if* (hare->next->next == head)

        hare = hare->next;

*// Set the head pointer of first half*

    head1 = head;

*// Set the head pointer of second half*

*if* (head->next != head)

        head2 = tortoise->next;

*// Make second half circular*

    hare->next = tortoise->next;

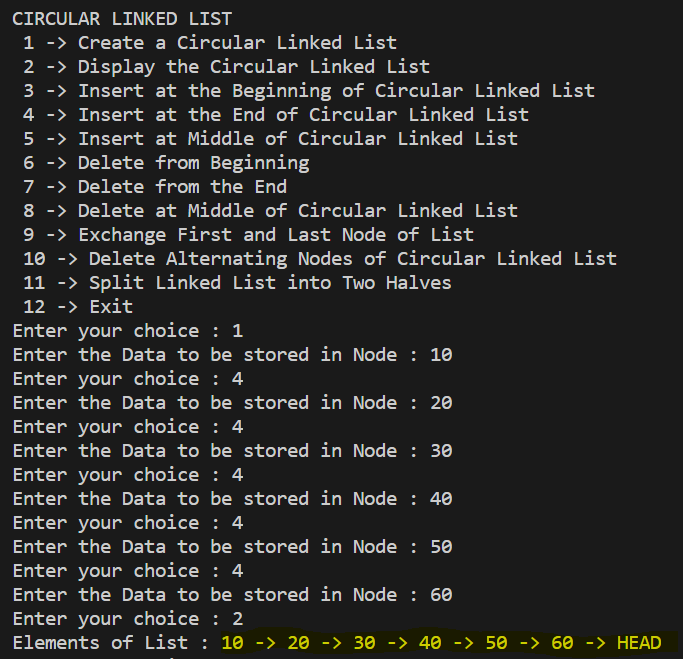
*// Make first half circular*

    tortoise->next = head;

}

Test Cases:

A.) Creation of Circular Linked List



B.) Exchange First and Last Node of Circular Linked List

C.) Delete Alternating Nodes of Circular Linked List

D.) Exchange First and Last Node of Circular Linked List

[Added 3 More Elements in Circular Linked List]

Splitted Circular Linked List into Two Halves:

