Institute of Computer Technology
B. Tech Computer Science and Engineering
Subject: DS (2CSE302)

PRACTICAL-17

AIM: - Implement the scenario based on circular singly linked list.

Q. Nilima is working in production department of pharma company. She needs to update details for stock of medicine, sometimes afterwards she needs to update record at the end of day so she wants to prepare data which she can dynamically update as per her requirement. Refer the given scenario and provide appropriate solution using circular singly linked list:

1. Insert new item at the first position
2. Insert new item at the last position
3. Insert new item before the given value of node
4. Insert new item after the given value of node
5. Delete item from the first position
6. Delete item from the last position
7. Delete particular item
8. Delete item before the given item
9. Delete item after the given item
10. Search item from the list
11. Display all elements of the list
12. Count total number of items

SOLUTION

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
        int data;
        struct node *next;
};
struct node *head;

void insert_F()
{
        printf("\n+++++| INSERT FIRST |+++++\n");
        struct node *ptr, *temp;
        int yash = 0;
        ptr = (struct node *)malloc(sizeof(struct node));
        if (ptr == NULL)
        {
```

```
            printf("\nOVERFLOW");
        }
        else
        {
            printf("\nEnter value you want to insert: ");
            scanf("%d", &yash);
            ptr->data = yash;
            if (head == NULL)
            {
                head = ptr;
                ptr->next = head;
            }
            else
            {
                temp = head;
                while (temp->next != head)
                    temp = temp->next;
                ptr->next = head;
                temp->next = ptr;
                head = ptr;
            }
            printf("\nNode Inserted\n");
        }
}
void insert_L()
{
        printf("\n+++++| INSERT LAST |+++++\n");
        struct node *ptr, *temp;
        int yash = 0;
        ptr = (struct node *)malloc(sizeof(struct node));
        if (ptr == NULL)
        {
            printf("\nOVERFLOW\n");
        }
        else
        {
            printf("\nEnter value you want to insert: ");
            scanf("%d", &yash);
            ptr->data = yash;
            if (head == NULL)
            {
                head = ptr;
                ptr->next = head;
            }
```

```
                else
                {
                        temp = head;
                        while (temp->next != head)
                        {
                                temp = temp->next;
                        }
                        temp->next = ptr;
                        ptr->next = head;
                }

                printf("\nNode Inserted\n");
        }
}
void insert_B()
{
        printf("\n+++++| INSERT BEFORE |+++++\n");
        int yash, prajapati;
        struct node *new_node, *ptr, *previous;
        printf("Insert item which you want to insert: ");
        scanf("%d", &yash);
        printf("Insert item before where you want to insert: ");
        scanf("%d", &prajapati);
        new_node = (struct node *)malloc(sizeof(struct node));
        new_node->data = yash;
        ptr = head;
        while (ptr->data != prajapati)
        {
                previous = ptr;
                ptr = ptr->next;
        }
        previous->next = new_node;
        new_node->next = ptr;
}

void insert_A()
{
        printf("\n+++++| INSERT AFTER |+++++\n");
        int yash, prajapati;
        struct node *new_node, *ptr, *after;
        printf("Insert item which you want to insert: ");
        scanf("%d", &yash);
        printf("Insert item after where you want to insert: ");
        scanf("%d", &prajapati);
```

```
        new_node = (struct node *)malloc(sizeof(struct node));
        new_node->data = yash;
        ptr = head;

        while (ptr->data != prajapati)
        {
                ptr = ptr->next;
        }
        new_node->next = ptr->next;
        ptr->next = new_node;
}

void delete_F()
{
        struct node *ptr;
        if (head == NULL)
        {
                printf("\nUNDERFLOW");
        }
        else if (head->next == head)
        {
                head = NULL;
                free(head);
                printf("\nData Deleted.\n");
        }

        else
        {
                ptr = head;
                while (ptr->next != head)
                        ptr = ptr->next;
                ptr->next = head->next;
                free(head);
                head = ptr->next;
                printf("\nData Deleted.\n");
        }
}
void delete_L()
{
        struct node *ptr, *preptr;
        if (head == NULL)
        {
                printf("\nUNDERFLOW");
        }
```

```
        else if (head->next == head)
        {
                head = NULL;
                free(head);
                printf("\nData Deleted\n");
        }
        else
        {
                ptr = head;
                while (ptr->next != head)
                {
                        preptr = ptr;
                        ptr = ptr->next;
                }
                preptr->next = ptr->next;
                free(ptr);
                printf("\nNode Deleted\n");
        }
}

void delete_pr ()
{
   int yash;
        if (head != NULL)
        {
                printf("\nEnter element which you want to delete: ");
                scanf("%d", &yash);
                struct node *ptr, *previous;
                ptr = head;
                previous = ptr;
                while (ptr->data != yash)
                {
                        previous = ptr;
                        ptr = ptr->next;
                }
                previous->next=ptr->next;
                free(ptr);
                printf("\nData deleted: %d",yash);
        }
        else
        {
                printf("\nEmpty!!!\n");
        }
```

```
}

void delete_B()
{
	if (head != NULL)
	{
		struct node *new_node, *ptr, *temp1, *temp2;
		new_node = (struct node *)malloc(sizeof(struct node));
		new_node = head;
		int yash = 0;
		printf("\nEnter value before which you have delete node: ");
		scanf("%d", &yash);
		while (new_node->data != yash)
		{
			ptr = new_node;
			new_node = new_node->next;
		}
		temp1 = head;
		while (temp1 != ptr)
		{
			temp2 = temp1;
			temp1 = temp1->next;
		}
		temp2->next = new_node;
		printf("\nDeleted data: %d", ptr->data);
		free(ptr);
	}
	else
	{
		printf("\nEmpty!!!\n");
	}
}
void delete_A()
{
	if (head != NULL)
	{
		struct node *new_node, *ptr, *temp1;
		new_node = (struct node *)malloc(sizeof(struct node));
		new_node = head;
		int yash = 0;
		printf("\nEnter value after which you have delete node: ");
		scanf("%d", &yash);
		while (new_node->data != yash)
		{
```

```
                new_node = new_node->next;
            }
            ptr = new_node->next;
            temp1 = ptr->next;
            new_node->next = temp1;
            printf("\nDeleted data: %d", ptr->data);
            free(ptr);
        }
        else
        {
            printf("\nEmpty!!!\n");
        }
}

void search()
{
        struct node *ptr;
        int yash, i = 0, flag = 1;
        ptr = head;
        if (ptr == NULL)
        {
            printf("\nList is empty!!\n");
        }
        else
        {
            printf("\nEnter item which you want to search: ");
            scanf("%d", &yash);
            if (head->data == yash)
            {
                printf("\nItem found at location %d", i + 1);
                flag = 0;
            }
            else
            {
                while (ptr->next != head)
                {
                    if (ptr->data == yash)
                    {
                        printf("\nItem found at location %d ", i + 1);
                        flag = 0;
                        break;
                    }
                    else
                    {
```

```
                                flag = 1;
                        }
                        i++;
                        ptr = ptr->next;
                }
        }
        if (flag != 0)
        {
                printf("\nItem not found\n");
        }
    }
}

void display()
{
    struct node *ptr;
    ptr = head;
    if (head == NULL)
    {
        printf("\nEmpty!!!");
    }
    else
    {
        printf("\n+++++| D I S P L A Y |+++++\n\n");

        while (ptr->next != head)
        {

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

void count()
{
    int count = 0;
    struct node *ptr = head;
    ptr = head;

    do
    {
        ptr=ptr->next;
```

```
            count++;
        } while (ptr != head);


        printf("\nTotal number of items are:\t%d\n", count);
}

void main()
{
        int ch = 0;
        while (1)
        {
        RETRY:
            printf("\n+++++| M E N U |+++++\n");
            printf("\n1. Insert new item at the first position");
            printf("\n2. Insert new item at the last position");
            printf("\n3. Insert new item before the given value of node");
            printf("\n4. Insert new item after the given value of node");
            printf("\n5. Delete item from the first position");
            printf("\n6. Delete item from the last position");
            printf("\n7. Delete particular item");
            printf("\n8. Delete item before the given item");
            printf("\n9. Delete item after the given item");
            printf("\n10. Search item from the list");
            printf("\n11. Display all elements of the list");
            printf("\n12. Count total number of items");
            printf("\n13. EXIT");
            printf("\n\nEnter your choice: ");
            scanf("%d", &ch);

            switch (ch)
            {
            case 1:
                insert_F();
                break;
            case 2:
                insert_L();
                break;
            case 3:
                insert_B();
                break;
            case 4:
                insert_A();
                break;
```

```
            case 5:
                    delete_F();
                    break;
            case 6:
                    delete_L();
                    break;
            case 7:
                    delete_pr();
                    break;
            case 8:
                    delete_B();
                    break;
            case 9:
                    delete_A();
                    break;
            case 10:
                    search();
                    break;
            case 11:
                    display();
                    break;
            case 12:
                    count();
                    break;
            case 13:
                    printf("+++++| EXITING SYSTEM |+++++\n");
                    exit(0);
                    break;

            default:
                    printf("\nINVALID CHOICE !! Please try again.....\n");
                    goto RETRY;
                    break;
            }
    }
}
```

OUTPUT