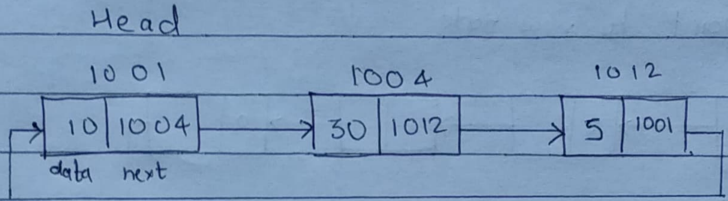
Shreyas Sawant
D7A 53

AIM: Implement Circular Linked list using ADT

THEORY:

1. Circular linked list is a variation of linked list in which the first elements points to the last element
2. 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.

For Example,



- Operations on Circular Linked List:

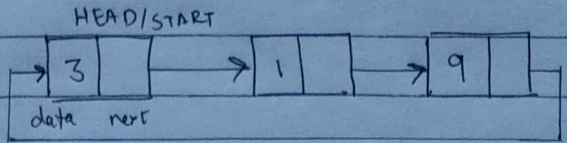
1) Traversing

a) Traversing a linked list means accessing the nodes of the list in order to perform some processing on them.

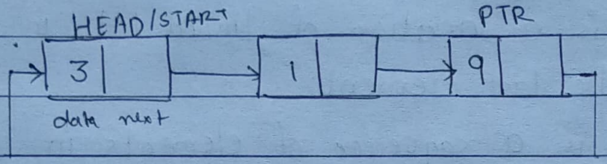
b) A circular linked list contains a pointer variable START which stores address of first node of list.

2) Insertion:

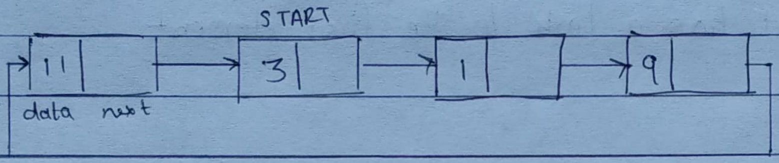
a) Insertion of node at beginning of circular linked list:
Consider the linked list shown below. Suppose we want to add a new node with data 11 as first node of list.



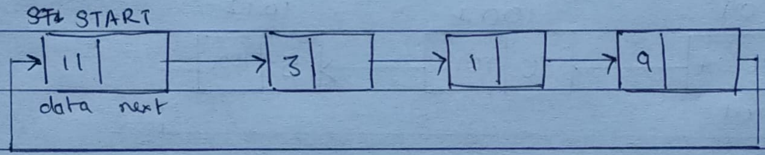
Allocate memory for new node and set its data part to 11.
Then take a pointer variable PTR that points to the start node of the list. Move PTR so that it now points to last node of list.



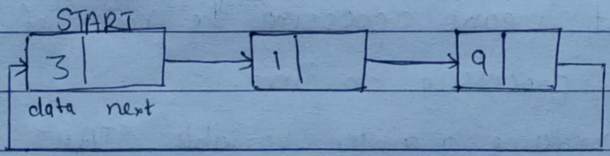
Add a new node between PTR and START



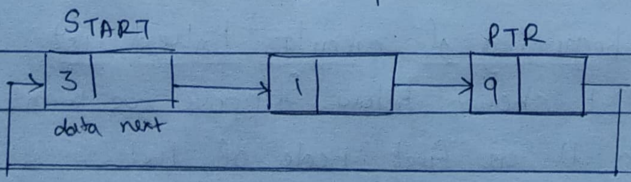
Make START point to new node



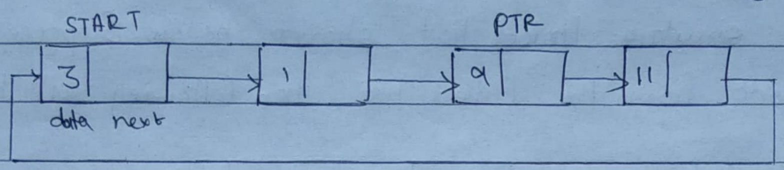
b) Insertion of Node at end of the circlar linked list.
Consider the linked list shown below. Suppose we want to add a new node with data 11 at the last node of the list



Allocate memory for new node and set its data part to 11.
Take a pointer variable PTR which will intially point to START. Move PTR so that it now point to last node of list



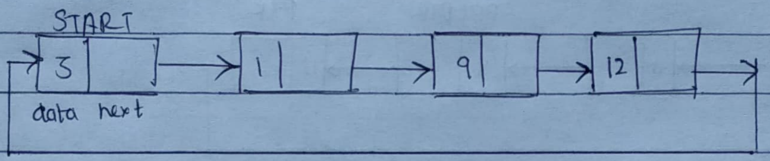
Add the new node after the node pointed by PTR



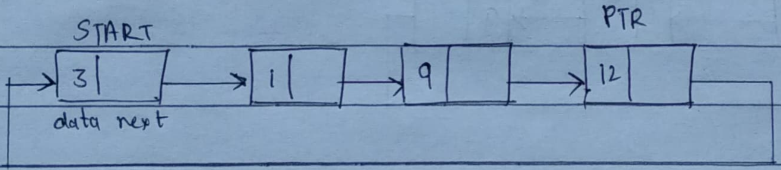
3) Deletion :

a) Deleting the first node from circular linked list :

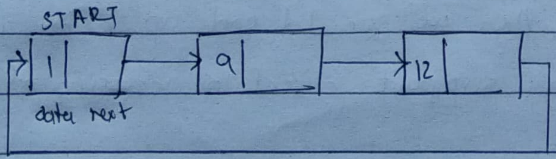
Consider the circular linked list shown below. Suppose we want to delete a node from beginning of the list.



Take a variable PTR and make it point to START. Move PTR further so that it points to last node of list

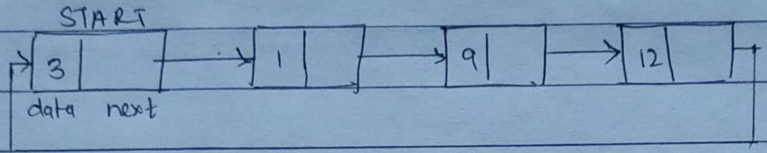


The next part of PTR is made to point to second node of list and memory of first node is freed. The ~~b~~ second node becomes the START of the list

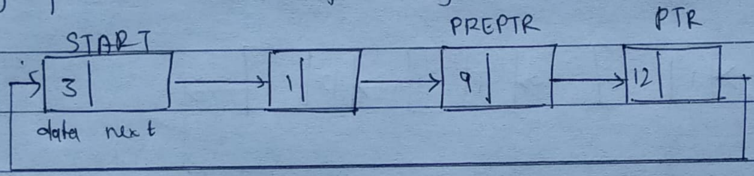


b) Deleting the last node from circular linkd list.

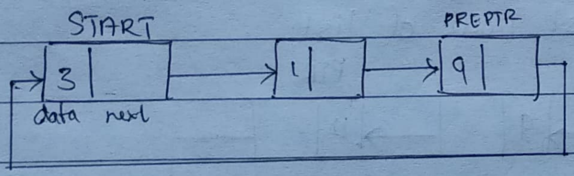
Consider the circulara linked list shown below. Suppose we want to delete last node from the linked list, then following will be done:



Take 2 pointer's PREPTR and PTR will will intially point to START. More PTR so that it points to the last node of list. PREPTR will always point to node preceding PTR.



Make PREPTR's next part START and free PTR. Now PREPTR is last node of list.



- Limitations of Circular Linked List:

i. They are complex as compared to linked list.

ii. Reversing of the list is complex as complared to single linked list.

iii. If not traversed carefully, then we would end up in an infinite loop.

iv. Circular linked list doesn't support direct ~~d~~ accessing of elements

CONCLUSION:

Errors encountered:

1) variable 'choice' declared inside switch, variable not defined.

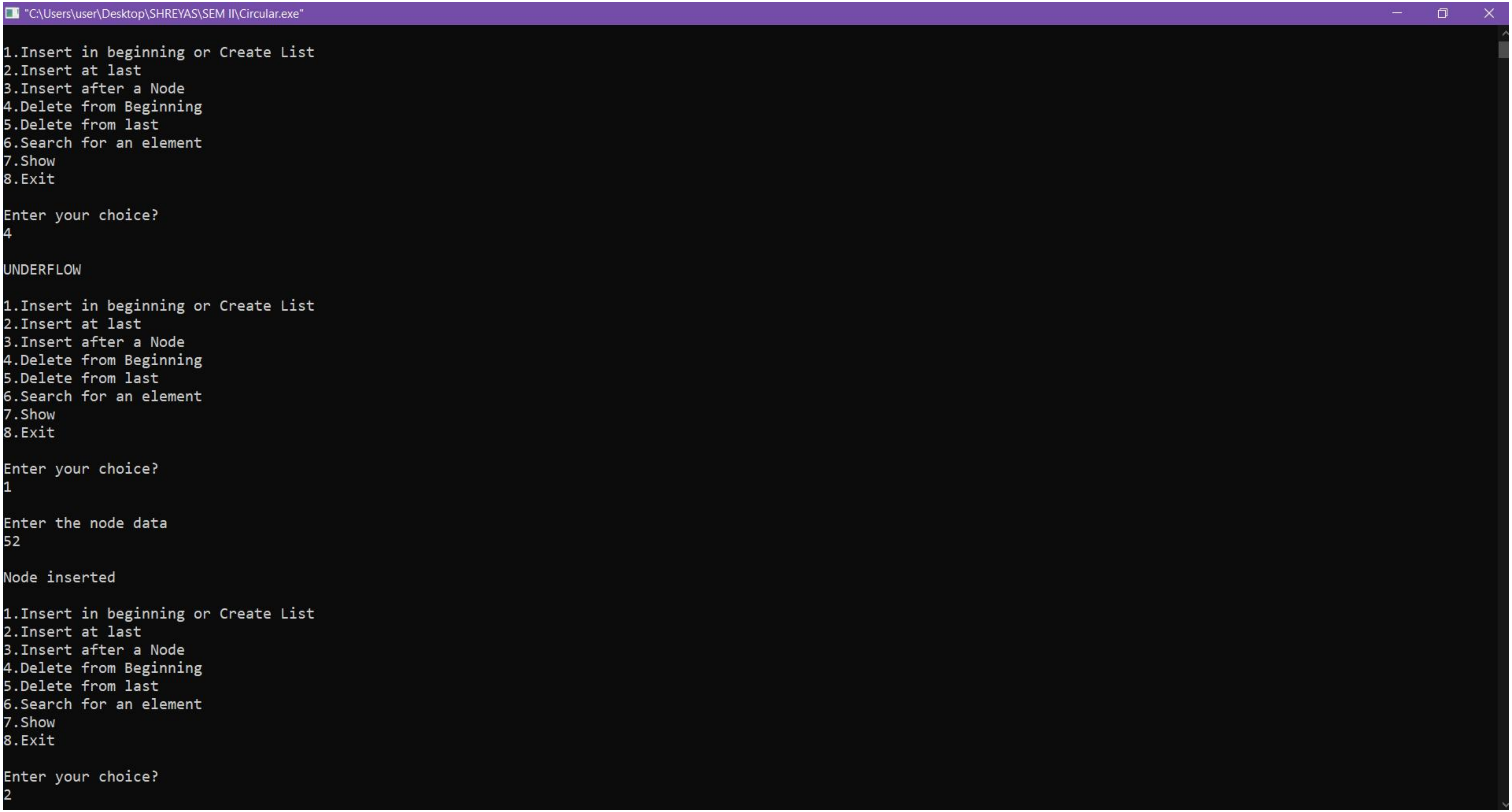
Solution Declare the variable outside the switch () and take input from user.

2) Using assignment operater '=' instead of '==' in if statement.

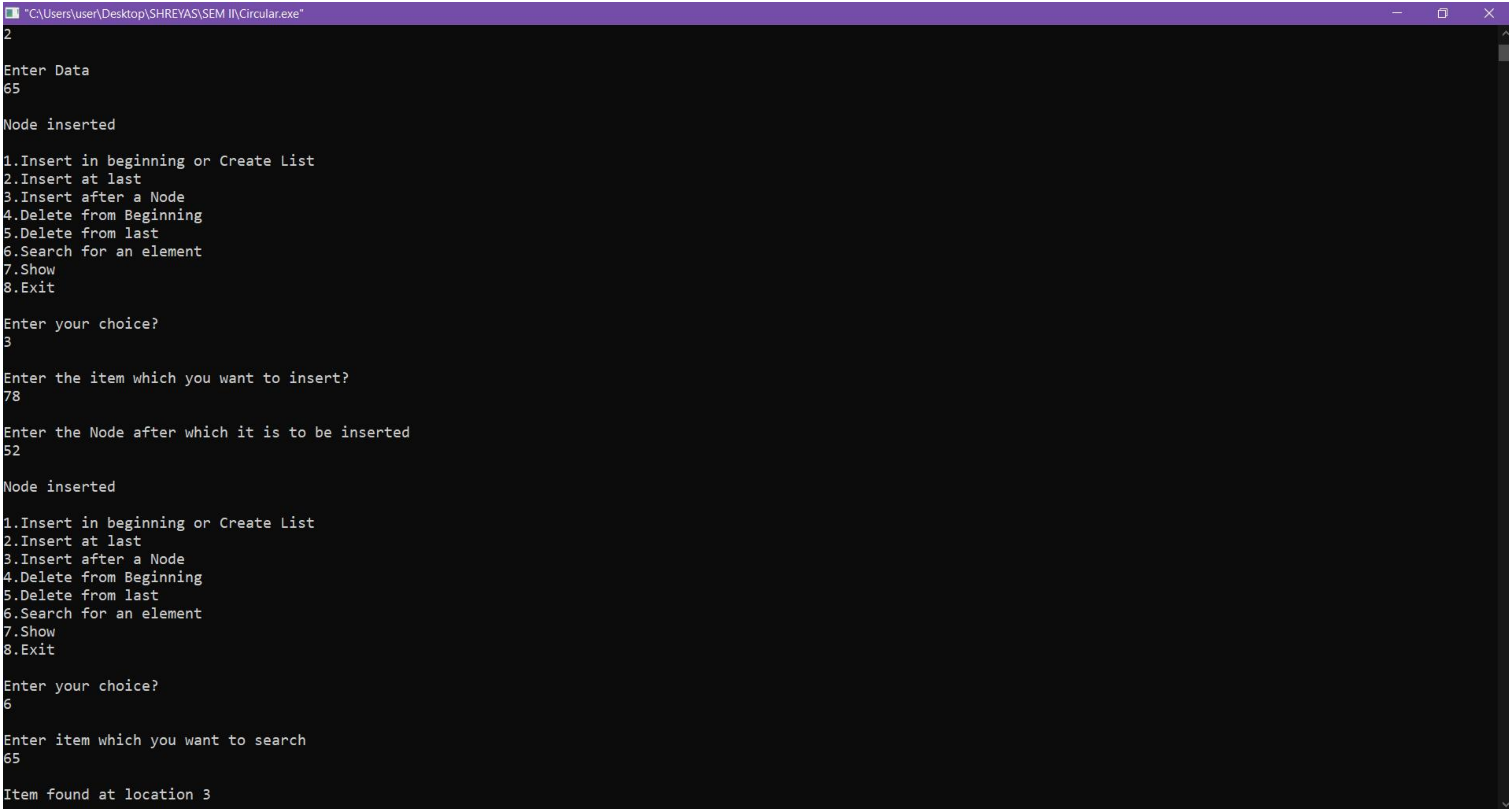
Solution Using the relation operator '==' solves the error.

```c
//SHREYAS SAWANT D7A 55
//Implement Circular Linked List using ADT

#include<stdio.h>
#include<stdlib.h>
struct node
{
    int data;
    struct Node *next;
};
struct node *head,*ptr,*temp;

void beginsert()
{
    struct node *ptr,*temp;
    int item;
    ptr = (struct node *)malloc(sizeof(struct node));
    if(ptr == NULL)
    {
        printf("\nOVERFLOW\n");
    }
    else
    {
        printf("\nEnter the node data\n");
        scanf("%d",&item);
        ptr -> data = item;
        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 lastinsert()
{
    struct node *ptr,*temp;
    int item;
    ptr = (struct node *)malloc(sizeof(struct node));
    if(ptr == NULL)
    {
        printf("\nOVERFLOW\n");
    }
    else
    {
        printf("\nEnter Data\n");
        scanf("%d",&item);
        ptr->data = item;
        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 insAfter(int a,int b)
{   int k=0;
    ptr =(struct Node *)malloc(sizeof(struct Node *));
    temp =head;
    if(temp==NULL)
    {
        printf("\nEMPTY\n");return;
    }
    while(temp->next!=NULL)
    {
        if(temp->data==b)
        {
            k=1;break;
        }
        temp=temp->next;
    }

    if(k)
    {
        ptr->next=temp->next;
        ptr->data=a;
        temp->next=ptr;
        printf("\nNode inserted\n");
    }
    else
        printf("\nNOT FOUND\n");
}

void begin_delete()
{
    struct node *ptr;
    if(head == NULL)
    {
        printf("\nUNDERFLOW\n");
    }
    else if(head->next == head)
    {
        head = NULL;
        free(head);
        printf("\nNode deleted\n");
    }

    else
    {   ptr = head;
        while(ptr -> next != head)
            ptr = ptr -> next;
        ptr->next = head->next;
        free(head);
        head = ptr->next;
        printf("\nNode deleted\n");

    }
}
void last_delete()
{
    struct node *ptr, *preptr;
    if(head==NULL)
    {
        printf("\nUNDERFLOW\n");
    }
    else if (head ->next == head)
    {
        head = NULL;
        free(head);
        printf("\nNode 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 search()
{
    struct node *ptr;
    int item,i=0,flag=1;
    ptr = head;temp=ptr->next;
    if(ptr == NULL)
    {
        printf("\nEmpty List\n");
    }
    else
    {
        printf("\nEnter item which you want to search\n");
        scanf("%d",&item);
        if(head ->data == item)
        {
        printf("\nItem found at location %d\n",i+1);
        flag=0;
        }
        else
        {
        while (ptr->next != head)
        {
            if(ptr->data == item)
            {
                printf("\nItem found at location %d\n ",i+1);
                flag=0;
                break;
            }
            else
            {
                flag=1;
            }
            i++;
            ptr = ptr -> next;

        }
        }
        if(ptr->data==item)
        {
             printf("\nItem found at location %d\n ",i+1);flag=0;
        }
        if(flag != 0)
        {
            printf("Item not found\n");
        }
    }

}

void display()
{
    struct node *ptr;
    ptr=head;
    if(head == NULL)
    {
        printf("\nEMPTY");return;
    }
    else
    {
        printf("\nElements of list \n");

        while(ptr->next!= head)
        {

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

}


void main ()
{
    int choice =0,item;
    while(choice != 7)
    {

        printf("\n1.Insert in beginning or Create List\n2.Insert at last\n3.Insert after a Node\n4.Delete from Beginning\n5.Delete from last\n6.Search for an element\n7.Show\n8.Exit\n");
        printf("\nEnter your choice?\n");
        scanf("\n%d",&choice);
        switch(choice)
        {
            case 1:
            beginsert();
            break;
            case 2:
            lastinsert();
            break;
            case 3:
```

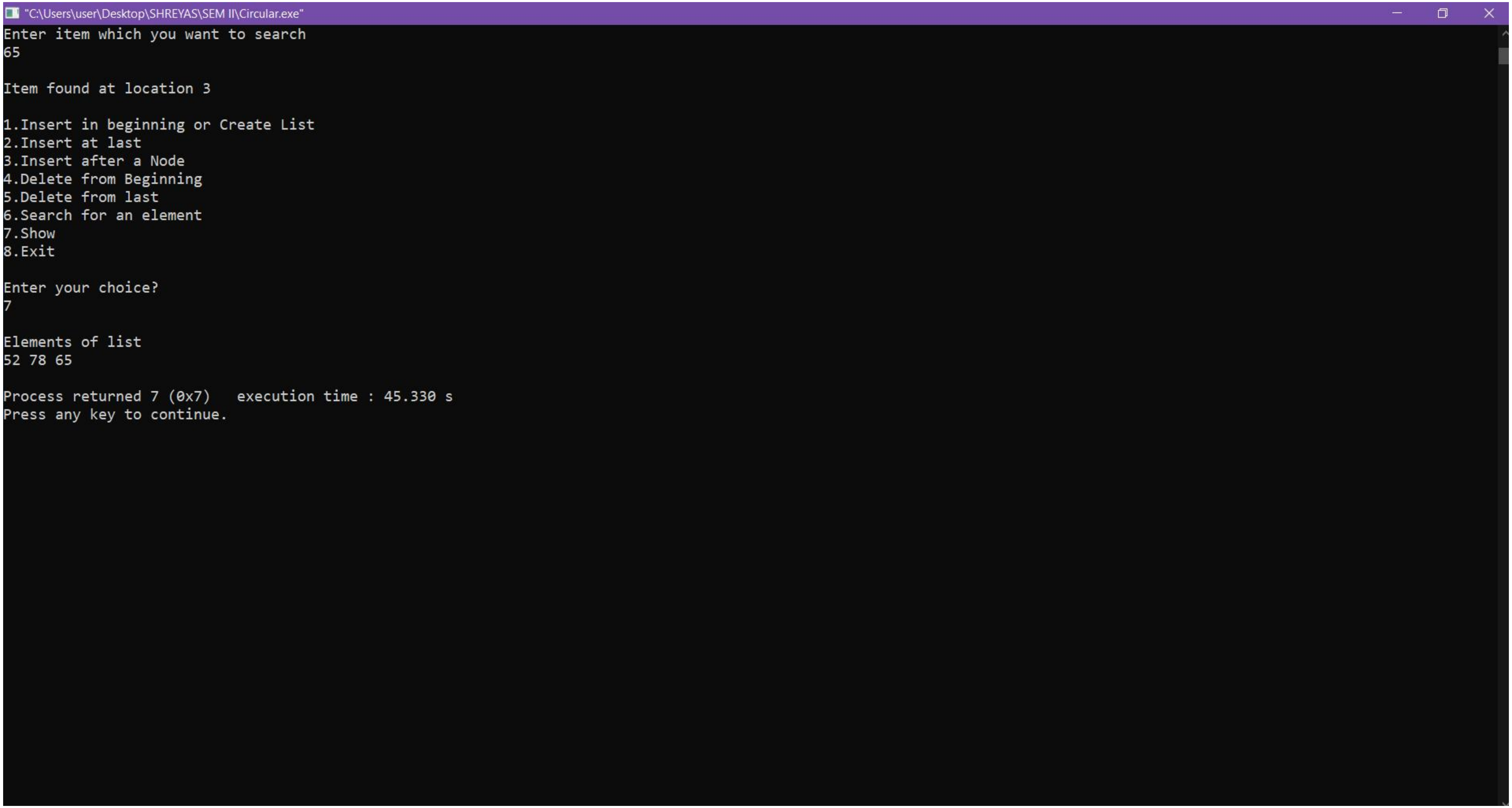
```
            {

            int n;
            printf("\nEnter the item which you want to insert?\n");
            scanf("%d",&item);
            printf("\nEnter the Node after which it is to be inserted\n");
            scanf("%d",&n);
            insAfter(item,n);break;
            }
            case 4:
            begin_delete();
            break;
            case 5:
            last_delete();
            break;
            case 6:
            search();
            break;
            case 7:
            {display();
            break;}
            case 8:
            exit(0);
            break;
            default:
            printf("Please enter valid choice..");
        }
    }
}

```



```
1.Insert in beginning or Create List
2.Insert at last
3.Insert after a Node
4.Delete from Beginning
5.Delete from last
6.Search for an element
7.Show
8.Exit

Enter your choice?
4

UNDERFLOW

1.Insert in beginning or Create List
2.Insert at last
3.Insert after a Node
4.Delete from Beginning
5.Delete from last
6.Search for an element
7.Show
8.Exit

Enter your choice?
1

Enter the node data
52

Node inserted

1.Insert in beginning or Create List
2.Insert at last
3.Insert after a Node
4.Delete from Beginning
5.Delete from last
6.Search for an element
7.Show
8.Exit

Enter your choice?
2
```



```
2

Enter Data
65

Node inserted

1.Insert in beginning or Create List
2.Insert at last
3.Insert after a Node
4.Delete from Beginning
5.Delete from last
6.Search for an element
7.Show
8.Exit

Enter your choice?
3

Enter the item which you want to insert?
78

Enter the Node after which it is to be inserted
52

Node inserted

1.Insert in beginning or Create List
2.Insert at last
3.Insert after a Node
4.Delete from Beginning
5.Delete from last
6.Search for an element
7.Show
8.Exit

Enter your choice?
6

Enter item which you want to search
65

Item found at location 3
```



```
Enter item which you want to search
65

Item found at location 3

1.Insert in beginning or Create List
2.Insert at last
3.Insert after a Node
4.Delete from Beginning
5.Delete from last
6.Search for an element
7.Show
8.Exit

Enter your choice?
7

Elements of list
52 78 65

Process returned 7 (0x7)   execution time : 45.330 s
Press any key to continue.
```