**Problem Statement 16** :Write a program to perform the following operations on a Single Linked List:

a) Insert at the beginning  
b) Insert at a specified position  
c) Insert at the end  
d) Delete an element from a specified position  
e) Search an element  
f) Count the number of nodes  
g) Sort the entire list  
h) Reverse the entire list  
i) Display the list

**Code:**

#include <stdio.h>

#include <stdlib.h>

struct node

{

    int num;

    struct node \*nextptr;

} \*stnode;

void createNodeList(int);

int countList();

void displayList();

void insertAtBeg();

void insertAtEnd();

void insertAtSpecificPos(int);

void deleteAtBeg();

void deleteAtEnd();

void deleteAtSpecificPos(int);

void reverseList();

int main()

{

    int n, choice, c, pos;

    printf("\n\n Linked List : To create and display Singly Linked List :\n");

    printf("-------------------------------------------------------------\n");

    printf(" Input the number of nodes : ");

    scanf("%d", &n);

    createNodeList(n);

    while (1)

    {

        printf("\nMenu: \n");

        printf("1.insertAtBeg\n");

        printf("2.insertAtEnd\n");

        printf("3.insertAtSpecificPos\n");

        printf("4.deleteAtBeg\n");

        printf("5.deleteAtEnd\n");

        printf("6.deleteAtSpecificPos\n");

        printf("7.displayList\n");

        printf("8.countList\n");

        printf("9.reverseList\n");

        printf("10.Exit\n");

        printf("Enter your choice: ");

        scanf("%d", &choice);

        switch (choice)

        {

        case 1:

            insertAtBeg();

            break;

        case 2:

            insertAtEnd();

            break;

        case 3:

            printf("Enter the position of insertion of the node:");

            scanf("%d", &pos);

            insertAtSpecificPos(pos);

        case 4:

            deleteAtBeg();

            break;

        case 5:

            deleteAtEnd();

            break;

        case 6:

            printf("Enter the position of deletion of the node:");

            scanf("%d", &pos);

            deleteAtSpecificPos(pos);

        case 7:

            printf("\nData entered in the list : \n");

            displayList();

            break;

        case 8:

            c = countList();

            printf("\nNumber of nodes : %d\n", c);

            break;

        case 9:

            reverseList();

            break;

        case 10:

            printf("Exiting!");

            return 0;

        default:

            printf("Invalid choice!");

            break;

        }

    }

}

void createNodeList(int n)

{

    struct node \*fnNode, \*tmp;

    int num, i;

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

    if (stnode == NULL)

    {

        printf(" Memory can not be allocated.");

    }

    else

    {

        printf(" Input data for node 1 : ");

        scanf("%d", &num);

        stnode->num = num;

        stnode->nextptr = NULL;

        tmp = stnode;

        for (i = 2; i <= n; i++)

        {

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

            if (fnNode == NULL)

            {

                printf(" Memory can not be allocated.");

                break;

            }

            else

            {

                printf(" Input data for node %d : ", i);

                scanf(" %d", &num);

                fnNode->num = num;

                fnNode->nextptr = NULL;

                tmp->nextptr = fnNode;

                tmp = tmp->nextptr;

            }

        }

    }

}

void insertAtBeg()

{

    struct node \*newNode;

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

    printf("Enter the value to insert:");

    scanf("%d", &newNode->num);

    if (stnode == NULL)

    {

        stnode = newNode;

        newNode->nextptr = NULL;

    }

    else

    {

        newNode->nextptr = stnode;

        stnode = newNode;

    }

}

void insertAtEnd()

{

    struct node \*newNode, \*temp;

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

    printf("Enter the value to insert:");

    scanf("%d", &newNode->num);

    newNode->nextptr = NULL;

    if (stnode == NULL)

    { // stnode = head

        stnode = newNode;

        newNode->nextptr = NULL;

    }

    else

    {

        temp = stnode;

        while (temp->nextptr != 0)

        {

            temp = temp->nextptr;

        }

        temp->nextptr = newNode;

    }

}

void insertAtSpecificPos(int pos)

{

    struct node \*newNode, \*temp;

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

    printf("Enter the value to insert:");

    scanf("%d", &newNode->num);

    newNode->nextptr = NULL;

    if (stnode == NULL)

    { // stnode = head

        stnode = newNode;

        newNode->nextptr = NULL;

    }

    else

    {

        temp = stnode;

        int cnt = 1;

        while (cnt < pos - 1)

        {

            temp = temp->nextptr;

            cnt++;

        }

        newNode->nextptr = temp->nextptr;

        temp->nextptr = newNode;

    }

}

void deleteAtBeg()

{

    struct node \*temp;

    if (stnode == NULL)

    {

        printf(" List is empty.");

    }

    else

    {

        temp = stnode;

        stnode = stnode->nextptr;

        free(temp);

    }

}

void deleteAtEnd()

{

    struct node \*temp, \*prevNode;

    if (stnode == NULL)

    {

        printf(" List is empty.");

    }

    else

    {

        while (temp->nextptr != NULL)

        {

            prevNode = temp;

            temp = temp->nextptr;

        }

        prevNode->nextptr = NULL;

        free(temp);

    }

}

void deleteAtSpecificPos(int pos)

{

    struct node \*newNode, \*temp, \*nextNode;

    if (stnode == NULL)

    {

        printf(" List is empty.");

    }

    else

    {

        temp = stnode;

        int cnt = 1;

        while (cnt < pos - 1)

        {

            temp = temp->nextptr;

            cnt++;

        }

        nextNode = temp->nextptr;

        temp->nextptr = nextNode->nextptr;

        free(nextNode);

    }

}

int countList()

{

    int ctr = 0;

    struct node \*tmp;

    tmp = stnode;

    // Counting the nodes by traversing the linked list

    while (tmp != NULL)

    {

        ctr++;

        tmp = tmp->nextptr;

    }

    return ctr;

}

void displayList()

{

    struct node \*tmp;

    if (stnode == NULL)

    {

        printf(" List is empty.");

    }

    else

    {

        tmp = stnode;

        printf("Linked List :\n");

        while (tmp != NULL)

        {

            printf("%d-->", tmp->num);

            tmp = tmp->nextptr;

        }

    }

    printf("NULL");

}

void reverseList()

{

    struct node \*prevNode, \*currNode, \*nextNode;

    prevNode = NULL;

    currNode = nextNode = stnode;

    while (currNode != NULL)

    {

        nextNode = currNode->nextptr;

        currNode->nextptr = prevNode;

        prevNode = currNode;

        currNode = nextNode;

    }

    stnode = prevNode;

}

**Output:**

A screenshot of a computer

Description automatically generated

**Problem Statement 17** :Write a program to do the following operations on Circular Linked List

a) Insert at beginning

b) Insert at a specified position

c) Insert at end

d) Delete an element from a specified position

e) Search an element

f) Count number of nodes

g) Sort the entire list

h) Reverse the entire list

i) Display the list

**Code:**

#include <stdio.h>

#include <stdlib.h>

struct node

{

    int data;

    struct node \*next;

} \*head, \*tail;

void createNodeList(int n)

{

    int i;

    struct node \*newNode;

    for (i = 1; i <= n; i++)

    {

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

        printf(" Enter the data for node %d:", i);

        scanf("%d", &newNode->data);

        newNode->next = NULL;

        if (head == NULL)

        {

            head = tail = newNode;

            tail->next = head;

        }

        else

        {

            tail->next = newNode;

            tail = newNode;

            tail->next = head;

        }

    }

}

void insertAtBeg()

{

    struct node \*newNode;

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

    printf("Enter the value to insert:");

    scanf("%d", &newNode->data);

    newNode->next = NULL;

    if (head == NULL)

    {

        head = tail = newNode;

        tail->next = head;

    }

    else

    {

        newNode->next = head;

        head = newNode;

        tail->next = head;

    }

}

void insertAtEnd()

{

    struct node \*newNode;

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

    printf("Enter the value to insert:");

    scanf("%d", &newNode->data);

    newNode->next = NULL;

    if (head == NULL)

    {

        head = tail = newNode;

        tail->next = head;

    }

    else

    {

        tail->next = newNode;

        tail = newNode;

        tail->next = head;

    }

}

void insertAtSpecificPos(int pos)

{

    struct node \*newNode, \*temp;

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

    printf("Enter the value to insert:");

    scanf("%d", &newNode->data);

    newNode->next = NULL;

    if (head == NULL)

    {

        head = tail = newNode;

        tail->next = head;

    }

    else

    {

        temp = head;

        int cnt = 1;

        while (cnt < pos - 1)

        {

            temp = temp->next;

            cnt++;

        }

        newNode->next = temp->next;

        temp->next = newNode;

    }

}

void deleteAtBeg()

{

    struct node \*temp;

    if (head == NULL)

    {

        printf(" List is empty.");

    }

    else

    {

        temp = head;

        head = head->next;

        tail->next = head;

        free(temp);

    }

}

void deleteAtEnd()

{

    struct node \*currNode, \*prevNode;

    if (head == NULL)

    {

        printf(" List is empty.");

    }

    else

    {

        currNode = head;

        while (currNode->next != head)

        {

            prevNode = currNode;

            currNode = currNode->next;

        }

        prevNode->next = head;

        free(currNode);

    }

}

void deleteAtSpecificPos(int pos)

{

    struct node \*currNode, \*nextNode;

    if (head == NULL)

    {

        printf(" List is empty.");

    }

    else

    {

        currNode = head;

        int cnt = 1;

        while (cnt < pos - 1)

        {

            currNode = currNode->next;

            cnt++;

        }

        nextNode = currNode->next;

        currNode->next = nextNode->next;

        free(nextNode);

    }

}

int countList()

{

    int ctr = 0;

    struct node \*tmp;

    tmp = head;

    // Counting the nodes by traversing the linked list

    do

    {

        ctr++;

        tmp = tmp->next;

    } while (tmp != head);

    return ctr;

}

void displayList()

{

    struct node \*tmp;

    if (head == NULL)

    {

        printf("List is empty.\n");

    }

    else

    {

        tmp = head;

        printf("Linked List: ");

        do

        {

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

            tmp = tmp->next;

        } while (tmp != head);

        printf("(Head)\n");

    }

}

void reverseList()

{

    struct node \*prevNode = tail, \*currNode = head, \*nextNode;

    if (head == NULL || head == tail)

    {

        return;

    }

    do

    {

        nextNode = currNode->next;

        currNode->next = prevNode;

        prevNode = currNode;

        currNode = nextNode;

    } while (currNode != head);

    tail = head;

    head = prevNode;

    tail->next = head;

}

int main()

{

    int n, choice, c, pos;

    printf("\n\n Linked List : To create and display Singly Circular Linked List :\n");

    printf("-------------------------------------------------------------\n");

    printf(" Input the number of nodes : ");

    scanf("%d", &n);

    createNodeList(n);

    while (1)

    {

        printf("\nMenu: \n");

        printf("1.insertAtBeg\n");

        printf("2.insertAtEnd\n");

        printf("3.insertAtSpecificPos\n");

        printf("4.deleteAtBeg\n");

        printf("5.deleteAtEnd\n");

        printf("6.deleteAtSpecificPos\n");

        printf("7.displayList\n");

        printf("8.countList\n");

        printf("9.reverseList\n");

        printf("10.Exit\n");

        printf("Enter your choice: ");

        scanf("%d", &choice);

        switch (choice)

        {

        case 1:

            insertAtBeg();

            break;

        case 2:

            insertAtEnd();

            break;

        case 3:

            printf("Enter the position of insertion of the node:");

            scanf("%d", &pos);

            insertAtSpecificPos(pos);

            break;

        case 4:

            deleteAtBeg();

            break;

        case 5:

            deleteAtEnd();

            break;

        case 6:

            printf("Enter the position of deletion of the node:");

            scanf("%d", &pos);

            deleteAtSpecificPos(pos);

            break;

        case 7:

            printf("\nData entered in the list : \n");

            displayList();

            break;

        case 8:

            c = countList();

            printf("\nNumber of nodes : %d\n", c);

            break;

        case 9:

            reverseList();

            break;

        case 10:

            printf("Exiting!");

            return 0;

        default:

            printf("Invalid choice!");

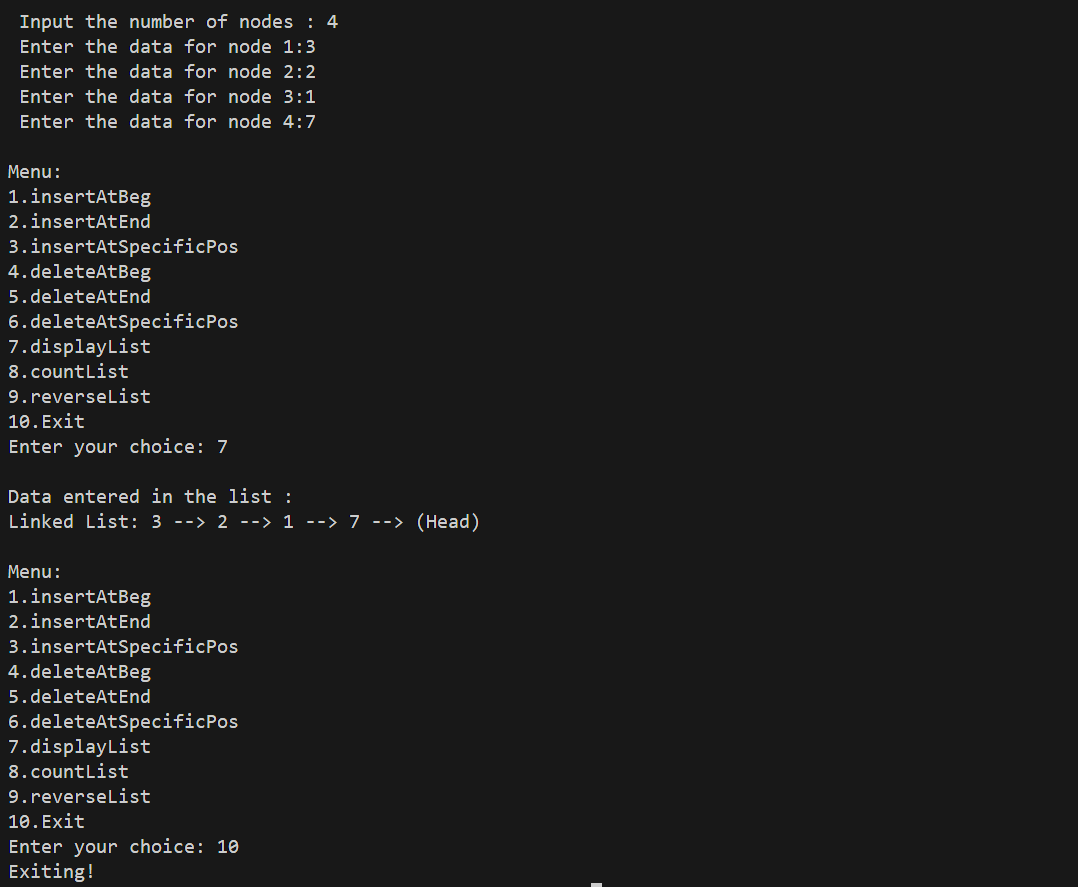
            break;

        }

    }

}

**Output:**

****

**Problem Statement 18** : Write a program to perform the following operations on a Doubly Linked List:

a) Insert at the beginning  
b) Insert at a specified position  
c) Insert at the end  
d) Delete an element from a specified position  
e) Search for an element  
f) Count the number of nodes  
g) Sort the entire list  
h) Reverse the entire list  
i) Display the list

**Code:**

#include <stdio.h>

#include <stdlib.h>

struct node

{

    int data;

    struct node \*next;

    struct node \*prev;

} \*head, \*tail;

void createNodeList(int n)

{

    int i;

    struct node \*newNode;

    for (i = 1; i <= n; i++)

    {

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

        printf(" Enter the data for node %d:", i);

        scanf("%d", &newNode->data);

        newNode->next = NULL;

        newNode->prev = NULL;

        if (head == NULL)

        {

            head = tail = newNode;

        }

        else

        {

            tail->next = newNode;

            newNode->prev = tail;

            tail = newNode;

        }

    }

}

void insertAtBeg()

{

    struct node \*newNode;

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

    printf("Enter the value to insert:");

    scanf("%d", &newNode->data);

    newNode->next = NULL;

    newNode->prev = NULL;

    if (head = NULL)

    {

        head = tail = newNode;

    }

    else

    {

        newNode->next = head;

        head->prev = newNode;

        head = newNode;

    }

}

void insertAtEnd()

{

    struct node \*newNode;

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

    printf("Enter the value to insert:");

    scanf("%d", &newNode->data);

    newNode->next = NULL;

    newNode->prev = NULL;

    if (head = NULL)

    {

        head = tail = newNode;

    }

    else

    {

        tail->next = newNode;

        newNode->prev = tail;

        tail = newNode;

    }

}

void insertAtSpecificPos(int pos)

{

    struct node \*newNode, \*currNode, \*nextNode;

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

    printf("Enter the value to insert:");

    scanf("%d", &newNode->data);

    newNode->next = NULL;

    newNode->prev = NULL;

    if (head = NULL)

    {

        head = tail = newNode;

    }

    else

    {

        currNode = head;

        int cnt = 1;

        while (cnt < pos - 1)

        {

            currNode = currNode->next;

            cnt++;

        }

        nextNode = currNode->next;

        newNode->prev = currNode;

        newNode->next = nextNode;

        currNode->next = newNode;

        nextNode->prev = newNode;

    }

}

void deleteAtBeg()

{

    struct node \*temp;

    if (head == NULL)

    {

        printf(" List is empty.");

    }

    else

    {

        temp = head;

        head = head->next;

        head->prev = NULL;

        free(temp);

    }

}

void deleteAtEnd()

{

    struct node \*temp;

    if (head == NULL)

    {

        printf(" List is empty.");

    }

    else

    {

        temp=tail;

        tail=tail->prev;

        free(temp);

    }

}

void deleteAtSpecificPos(int pos)

{

    struct node \*currNode, \*nextNode;

    if (head == NULL)

    {

        printf(" List is empty.");

    }

    else

    {

        currNode = head;

        int cnt = 1;

        while (cnt < pos - 1)

        {

            currNode = currNode->next;

            cnt++;

        }

        nextNode = currNode->next;

        currNode->next = nextNode->next;

        nextNode->next->prev = currNode;

        free(nextNode);

    }

}

int countList()

{

    int ctr = 0;

    struct node \*tmp;

    tmp = head;

    // Counting the nodes by traversing the linked list

    while (tmp != NULL)

    {

        ctr++;

        tmp = tmp->next;

    }

    return ctr;

}

void displayList()

{

    struct node \*tmp;

    if (head == NULL)

    {

        printf(" List is empty.");

    }

    else

    {

        tmp = head;

        printf("Linked List :\n");

        printf("NULL<-->");

        while (tmp != NULL)

        {

            printf("%d<-->", tmp->data);

            tmp = tmp->next;

        }

    }

    printf("NULL");

}

void reverseList(){

    struct node \*currNode,\*nextNode,\*temp;

    currNode=head;

    while(currNode!=NULL){

       nextNode=currNode->next;

       currNode->next=currNode->prev;

       currNode->prev=nextNode;

       currNode=nextNode;

    }

    temp=head;

    head=tail;

    tail=temp;

}

int main()

{

    int n, choice, c, pos;

    printf("\n\n Linked List : To create and display Doubly Linked List :\n");

    printf("-------------------------------------------------------------\n");

    printf(" Input the number of nodes : ");

    scanf("%d", &n);

    createNodeList(n);

    while (1)

    {

        printf("\nMenu: \n");

        printf("1.insertAtBeg\n");

        printf("2.insertAtEnd\n");

        printf("3.insertAtSpecificPos\n");

        printf("4.deleteAtBeg\n");

        printf("5.deleteAtEnd\n");

        printf("6.deleteAtSpecificPos\n");

        printf("7.displayList\n");

        printf("8.countList\n");

        printf("9.reverseList\n");

        printf("10.Exit\n");

        printf("Enter your choice: ");

        scanf("%d", &choice);

        switch (choice)

        {

        case 1:

            insertAtBeg();

            break;

        case 2:

            insertAtEnd();

            break;

        case 3:

            printf("Enter the position of insertion of the node:");

            scanf("%d", &pos);

            insertAtSpecificPos(pos);

            break;

        case 4:

            deleteAtBeg();

            break;

        case 5:

            deleteAtEnd();

            break;

        case 6:

            printf("Enter the position of deletion of the node:");

            scanf("%d", &pos);

            deleteAtSpecificPos(pos);

            break;

        case 7:

            printf("\nData entered in the list : \n");

            displayList();

            break;

        case 8:

            c = countList();

            printf("\nNumber of nodes : %d\n", c);

            break;

        case 9:

            reverseList();

            break;

        case 10:

            printf("Exiting!");

            return 0;

        default:

            printf("Invalid choice!");

            break;

        }

    }

}

**Output:**

A screenshot of a computer program

Description automatically generated

**Problem Statement** **19**:Write a menu-driven program to implement Insert, Delete, and Display operations in a Stack using a Linked List.

**Code:**

#include <stdio.h>

#include <stdlib.h>

struct node {

    int data;

    struct node\* link;

};

struct node\* top = NULL;

int isEmpty() {

    return top == NULL;

}

void push(int data) {

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

    if (newNode == NULL) {

        printf("Stack Overflow.\n");

        exit(1);

    }

    newNode->data = data;

    newNode->link = top;

    top = newNode;

}

int pop() {

    if (isEmpty()) {

        printf("Stack Underflow.\n");

        exit(1);

    }

    struct node\* temp = top;

    int val = temp->data;

    top = top->link;

    free(temp);

    return val;

}

int peek() {

    if (isEmpty()) {

        printf("Stack Underflow.\n");

        exit(1);

    }

    return top->data;

}

void print() {

    if (isEmpty()) {

        printf("Stack Underflow.\n");

        exit(1);

    }

    struct node\* temp = top;

    printf("The stack elements are: ");

    while (temp) {

        printf("%d ", temp->data);

        temp = temp->link;

    }

    printf("\n");

}

int main() {

    int choice, data;

    while (1) {

        printf("\n1. Push\n");

        printf("2. Pop\n");

        printf("3. Peek\n");

        printf("4. Print all elements of the stack\n");

        printf("5. Quit\n");

        printf("Please enter your choice: ");

        scanf("%d", &choice);

        switch (choice) {

            case 1:

                printf("Enter the element to be pushed: ");

                scanf("%d", &data);

                push(data);

                break;

            case 2:

                data = pop();

                printf("Deleted element is %d\n", data);

                break;

            case 3:

                printf("The topmost element of the stack is %d\n", peek());

                break;

            case 4:

                print();

                break;

            case 5:

                exit(0);//  Successful Termination ---> Indicates the program ran successfully.

            default:

                printf("Wrong choice\n");

        }

    }

    return 0;

}

**Output:**

A screenshot of a computer program

Description automatically generated

**Problem Statement 20** :Write a menu-driven program to implement Insert, Delete, and Display operations in a Queue using a Linked List.

**Code:**

#include <stdio.h>

#include <stdlib.h>

struct node{

    int data;

    struct node\* link;

};

struct node \*front = NULL;

struct node \*rear = NULL;

void enqueue(int val){

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

    newNode->data = val;

    newNode->link= NULL;

    if(newNode==NULL){

        printf("Overflow");

    }

    else if(  front==NULL && rear==NULL){

        front=rear=newNode;

    }

    else{

        rear->link=newNode;

        rear=newNode;

    }

}

int dequeue(){

    struct node\* temp;

    if(front==NULL && rear==NULL){

        printf("Underflow");

    }

    if(rear==front){

        temp=front;

        front=rear=NULL;

        free(temp);

    }

    else{

        temp=front;

        front=front->link;

        free(temp);

    }

}

void display(){

    struct node\* temp;

    if(front==NULL && rear==NULL){

        printf("Underflow");

    }

    temp=front;

    while(temp!=NULL){

        printf("%d ",temp->data);

        temp=temp->link;

    }

}

int main()

{

   int choice, val;

   while (1)

   {

      printf("\nMenu: \n");

      printf("1.Insert\n");

      printf("2.Delete\n");

      printf("3.Display\n");

      printf("4.Exit\n");

      printf("Enter your choice: ");

      scanf("%d", &choice);

      switch (choice)

      {

      case 1:

         printf("Enter the value to insert:");

         scanf("%d", &val);

         enqueue(val);

         break;

      case 2:

         val = dequeue();

         printf("The dequeued element is %d", val);

         break;

      case 3:

         display();

         break;

      case 4:

         printf("Exiting...");

         exit(0);//successful termination

      default:

         printf("\nEnter Valid input!");

      }

   }

   return 0;

}

**Output:**

A screenshot of a computer program

Description automatically generated



**Problem Statement 21** :Write a program to implement Binary Search Tree Traversal.

**Code:**

#include <stdio.h>

#include <stdlib.h>

struct Node

{

    int data;

    struct Node \*left;

    struct Node \*right;

};

struct Node \*createNode(int data)

{

    struct Node \*newNode = (struct Node \*)malloc(sizeof(struct Node));

    newNode->data = data;

    newNode->left = newNode->right = NULL;

    return newNode;

}

void inorderTraversal(struct Node \*root)

{

    if (root == NULL)

        return;

    inorderTraversal(root->left);

    printf("%d ", root->data);

    inorderTraversal(root->right);

}

void preorderTraversal(struct Node \*root)

{

    if (root == NULL)

        return;

    printf("%d ", root->data);

    preorderTraversal(root->left);

    preorderTraversal(root->right);

}

void postorderTraversal(struct Node \*root)

{

    if (root == NULL)

        return;

    postorderTraversal(root->left);

    postorderTraversal(root->right);

    printf("%d ", root->data);

}

struct Node \*insertNode(struct Node \*root, int data)

{

    if (root == NULL)

        return createNode(data);

    if (data < root->data)

        root->left = insertNode(root->left, data);

    else if (data > root->data)

        root->right = insertNode(root->right, data);

    return root;

}

int main()

{

    struct Node \*root = NULL;

    int choice, data;

    while (1)

    {

        printf("\n--- Binary Search Tree Menu ---\n");

        printf("1. Insert a node\n");

        printf("2. Inorder Traversal\n");

        printf("3. Preorder Traversal\n");

        printf("4. Postorder Traversal\n");

        printf("5. Exit\n");

        printf("Enter your choice: ");

        scanf("%d", &choice);

        switch (choice)

        {

        case 1:

            printf("Enter the value to insert: ");

            scanf("%d", &data);

            root = insertNode(root, data);

            printf("Node inserted.\n");

            break;

        case 2:

            printf("Inorder Traversal: ");

            inorderTraversal(root);

            printf("\n");

            break;

        case 3:

            printf("Preorder Traversal: ");

            preorderTraversal(root);

            printf("\n");

            break;

        case 4:

            printf("Postorder Traversal: ");

            postorderTraversal(root);

            printf("\n");

            break;

        case 5:

            printf("Exiting...\n");

            exit(0);

        default:

            printf("Invalid choice. Please try again.\n");

        }

    }

    return 0;

}

Output:

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**