1.Reverse a String using pointers:

#include<stdio.h>

#include<string.h>

int main()

{

char word[20];

scanf("%s",word);

for(int i=strlen(word);i>=0;i--)

{

printf("%c",word[i]);

}

return 0;

}

2.Add complex numbers using structures:

#include <stdio.h>

typedef struct complex {

float real;

float imag;

} complex;

complex add(complex n1, complex n2);

int main() {

complex n1, n2, result;

scanf("%f %f", &n1.real, &n1.imag);

scanf("%f %f", &n2.real, &n2.imag);

result = add(n1, n2);

printf("Sum = %.1f + %.1fi", result.real, result.imag);

return 0;

}

complex add(complex n1, complex n2) {

complex temp;

temp.real = n1.real + n2.real;

temp.imag = n1.imag + n2.imag;

return (temp);

}

3. Count the number of vowels and consonants in a string using a pointer:

#include <stdio.h>

int main()

{

char str1[100];

char \*pt;

int ctrV,ctrC;

printf(" Input a string: ");

fgets(str1, sizeof str1, stdin);

pt=str1;

ctrV=ctrC=0;

while(\*pt!='\0')

{

if(\*pt=='A' ||\*pt=='E' ||\*pt=='I' ||\*pt=='O' ||\*pt=='U' ||\*pt=='a' ||\*pt=='e' ||\*pt=='i' ||\*pt=='o' ||\*pt=='u')

ctrV++;

else

ctrC++;

pt++;

}

printf(" Number of vowels : %d\n Number of consonants : %d\n",ctrV,ctrC-1);

return 0;

}

4. Struct example:

**Write a program to add two distances in feet and inches using structure.**

#include <stdio.h>

struct Distance {

int feet;

int inches;

};

struct Distance addDistances(struct Distance d1, struct Distance d2) {

struct Distance result;

result.feet = d1.feet + d2.feet;

result.inches = d1.inches + d2.inches;

if (result.inches >= 12) {

result.inches -= 12;

result.feet++;

}

return result;

}

int main() {

struct Distance distance1, distance2, distanceSum;

printf("Enter first distance:\n");

printf("Feet: ");

scanf("%d", &distance1.feet);

printf("Inches: ");

scanf("%d", &distance1.inches);

printf("Enter second distance:\n");

printf("Feet: ");

scanf("%d", &distance2.feet);

printf("Inches: ");

scanf("%d", &distance2.inches);

distanceSum = addDistances(distance1, distance2);

printf("Total distance- Feet: %d, Inches: %d", distanceSum.feet, distanceSum.inches);

return 0;

}

5.Sum of elements using pointers:

#include <stdio.h>

int main() {

int arr[5],n;

int \*ptr = arr;

int sum = 0;

printf("enter total numbers:");

scanf("%d",&n);

for(int i=0; i<n; i++) {

scanf("%d",&arr[i]);

}

for(int i=0; i<n; i++) {

sum += \*(ptr+i);

}

printf("The sum of array is : %d", sum);

return 0;

}

6.Store and display using sturct:

#include <stdio.h>

struct student

{

char name[50];

int year;

float marks;

};

int main()

{

struct student s;

scanf("%s",s.name);

scanf("%d",&s.year);

scanf("%f",&s.marks);

printf("%s\n",s.name);

printf("%d\n",s.year);

printf("%.2f\n",s.marks);

return 0;

}

7.Stack operations: Push, pop and display operations

#include<stdio.h>

#define max 50

int stack[max];

int top=-1;

void push(int data){

if(top==max-1){

printf("stack overflow");

return;

}

top=top+1;

stack[top]=data;

}

void pop(){

if(top==-1){

printf("stack underflow");

}

int value=stack[top];

top=top-1;

}

void print(){

if(top==-1)

printf("Stack empty");

else

printf("Top element is %d",stack[top]);

}

int main(){

int num;

char x;

scanf("%c",&x);

while(x!='c'){

if(x=='a'){

scanf("%d",&num);

push(num);

}

else if(x=='b'){

pop();

}

else if(x=='c'){

print();

}

scanf("%c",&x);

}

print();

return 0;

}

8. Infix to Prefix:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX\_SIZE 100

struct Stack {

int top;

char items[MAX\_SIZE];

};

void initialize(struct Stack\* stack) {

stack->top = -1;

}

int isEmpty(struct Stack\* stack) {

return stack->top == -1;

}

void push(struct Stack\* stack, char item) {

if (stack->top == MAX\_SIZE - 1) {

printf("Stack Overflow\n");

return;

}

stack->items[++stack->top] = item;

}

char pop(struct Stack\* stack) {

if (isEmpty(stack)) {

printf("Stack Underflow\n");

return -1;

}

return stack->items[stack->top--];

}

int isOperand(char ch) {

return (ch >= 'A' && ch <= 'Z') || (ch >= 'a' && ch <= 'z');

}

int isOperator(char ch) {

return ch == '+' || ch == '-' || ch == '\*' || ch == '/' || ch == '^';

}

int getPrecedence(char ch) {

switch (ch) {

case '+':

case '-':

return 1;

case '\*':

case '/':

return 2;

case '^':

return 3;

}

return -1;

}

void reverse(char\* str) {

int length = strlen(str);

int i, j;

char temp;

for (i = 0, j = length - 1; i < j; i++, j--) {

temp = str[i];

str[i] = str[j];

str[j] = temp;

}

}

void infixToPrefix(char\* infix, char\* prefix) {

struct Stack stack;

initialize(&stack);

reverse(infix);

int i, j = 0;

for (i = 0; infix[i] != '\0'; i++) {

if (isOperand(infix[i])) {

prefix[j++] = infix[i];

} else if (infix[i] == ')') {

push(&stack, infix[i]);

} else if (infix[i] == '(') {

while (!isEmpty(&stack) && stack.items[stack.top] != ')') {

prefix[j++] = pop(&stack);

}

if (!isEmpty(&stack) && stack.items[stack.top] != ')') {

printf("Invalid Expression\n");

return;

} else {

pop(&stack);

}

} else if (isOperator(infix[i])) {

while (!isEmpty(&stack) && getPrecedence(infix[i]) <= getPrecedence(stack.items[stack.top])) {

prefix[j++] = pop(&stack);

}

push(&stack, infix[i]);

}

}

while (!isEmpty(&stack)) {

prefix[j++] = pop(&stack);

}

prefix[j] = '\0';

reverse(prefix);

}

int main() {

char infix[MAX\_SIZE];

char prefix[MAX\_SIZE];

printf("Enter the infix expression: ");

scanf("%s", infix);

infixToPrefix(infix, prefix);

printf("Prefix expression: %s\n", prefix);

return 0;

}

9.Infix to Postfix:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX\_SIZE 100

int is\_operator(char c) {

return c == '+' || c == '-' || c == '\*' || c == '/' || c == '^';

}

int get\_precedence(char c) {

if (c == '^')

return 3;

else if (c == '\*' || c == '/')

return 2;

else if (c == '+' || c == '-')

return 1;

else

return 0;

}

void infix\_to\_postfix(char \*infix, char \*postfix) {

char stack[MAX\_SIZE];

int top = -1, i, j = 0;

for (i = 0; infix[i] != '\0'; i++) {

if (!is\_operator(infix[i]) && infix[i] != '(' && infix[i] != ')') {

postfix[j++] = infix[i];

}

else if (is\_operator(infix[i])) {

while (top != -1 && stack[top] != '(' && get\_precedence(stack[top]) >= get\_precedence(infix[i])) {

postfix[j++] = stack[top--];

}

stack[++top] = infix[i];

}

else if (infix[i] == '(') {

stack[++top] = infix[i];

}

else if (infix[i] == ')') {

while (stack[top] != '(') {

postfix[j++] = stack[top--];

}

top--;

}

}

while (top != -1) {

postfix[j++] = stack[top--];

}

postfix[j] = '\0';

}

int main() {

char infix[MAX\_SIZE], postfix[MAX\_SIZE];

printf("Enter the infix expression: ");

fgets(infix, MAX\_SIZE, stdin);

infix\_to\_postfix(infix, postfix);

printf("Postfix Expression: %s", postfix);

return 0;

}

10.Evalution of postfix expression:

#include <stdio.h>

#include <stdlib.h>

#include <ctype.h>

#define MAX\_STACK\_SIZE 100

typedef struct {

int top;

int stack[MAX\_STACK\_SIZE];

} Stack;

void push(Stack \*s, int value) {

if (s->top < MAX\_STACK\_SIZE - 1) {

s->top++;

s->stack[s->top] = value;

} else {

printf("Stack Overflow\n");

exit(1);

}

}

int pop(Stack \*s) {

if (s->top >= 0) {

int value = s->stack[s->top];

s->top--;

return value;

} else {

printf("Stack Underflow\n");

exit(1);

}

}

int evaluate\_postfix(char \*expression) {

Stack stack;

stack.top = -1;

int i = 0;

while (expression[i] != '\0') {

char ch = expression[i];

if (isdigit(ch)) {

push(&stack, ch - '0'); // Convert char to integer

} else {

int operand2 = pop(&stack);

int operand1 = pop(&stack);

int result;

switch (ch) {

case '+':

result = operand1 + operand2;

break;

case '-':

result = operand1 - operand2;

break;

case '\*':

result = operand1 \* operand2;

break;

case '/':

result = operand1 / operand2;

break;

default:

printf("Invalid operator: %c\n", ch);

exit(1);

}

push(&stack, result);

}

i++;

}

return pop(&stack);

}

int main() {

char expression[100];

scanf("%s",expression);

int result = evaluate\_postfix(expression);

printf("The result is: %d\n", result);

return 0;

}

11. Linear Queue Operations:

#include<stdio.h>

#define N 100

int queue[N];

int front=-1,rear=-1;

void enqueue(){

if(front==-1 && rear==-1){

int m;

scanf("%d",&m);

front=rear=0;

queue[front]=m;

}

else if(rear==N-1){

printf("Queue is full");

}

else

{

int m;

scanf("%d",&m);

rear++;

queue[rear]=m;

}

}

void dequeue(){

if(front==-1 && rear==-1){

printf("Queue empty");

}

else if(front==rear){

front=rear=-1;

}

else{

front++;

}

}

void peek(){

if(front==-1 && rear==-1){

printf("Queue empty");

}

else{

printf("Front element is %d",queue[front]);

}

}

int main(){

char a;

do{

scanf("%c",&a);

switch(a){

case 'a':enqueue();

break;

case 'b':dequeue();

break;

case 'c':peek();

break;

}

}while(a!='c');

}

12.Circular Queue Operations:

#include<stdio.h>

#define max 7

int q[max];

int front=-1,rear=-1;

int isEmpty()

{

if(front==-1&&rear==-1)

{

return 1;

}

else

{

return 0;

}

}

int isFull()

{

if((rear+1)%max==front)

{

return 1;

}

else

{

return 0;

}

}

void enqueue()

{

int n;

scanf("%d",&n);

if(isFull())

{

printf("Queue is Full");

}

else

{

if(isEmpty())

{

front=rear=0;

}

else

{

rear=(rear+1)%max;

}

q[rear]=n;

}

}

void dequeue()

{

if(isEmpty())

{

printf("Queue is Empty");

}

else if(front==rear)

{

front=rear=-1;

}

else

{

front=(front+1)%max;

}

}

void display()

{

if(isEmpty())

{

printf("Queue is empty");

}

else

{

printf("Elements in a Queue are :");

for(int i=front;i<=rear;i++)

{

printf("%d",q[i]);

}

}

}

void main()

{

char ch;

do

{

scanf("%c",&ch);

switch(ch)

{

case 'a':

enqueue();

break;

case 'b':

dequeue();

break;

case 'c':

display();

break;

}

}while(ch!='c');

}

13.Singly Linked List Insertion:

#include <stdio.h>

#include <stdlib.h>

struct node {

int data;

struct node \* next;

};

struct node \* insertionAtFirst(int x,struct node \* ptr){

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

temp->data=x;

temp->next = ptr;

ptr = temp;

return ptr;

}

struct node \* insertionAtLast(int x,struct node \* ptr){

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

temp->data = x;

temp->next = NULL;

if(ptr == NULL){

ptr = temp;

}

else{

struct node \* temp1 = ptr;

while(temp1->next!=NULL){

temp1 = temp1->next;

}

temp1->next = temp;

}

return ptr;

}

struct node \* display(struct node \* ptr){

struct node \* temp = ptr;

printf("Elements in the linked list : \n");

while(temp!=NULL){

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

temp = temp->next;

if(temp!=NULL){

printf( "-> ");

}

}

return ptr;

}

void main(){

struct node \* head=NULL;

int choice = 1;

while(choice){

char n;

scanf("%c",&n);

if(n == 'a'){

head = display(head);

choice = 0;

}

else if(n == 'b'){

int x;

scanf("%d",&x);

head = insertionAtFirst(x,head);

}

else if(n == 'c'){

int x;

scanf("%d",&x);

head = insertionAtLast(x,head);

}

}

}

14.Singly Linked List Deletion:

#include<stdio.h>

#include<stdlib.h>

struct Node{

int data;

struct Node\*add;

};

void insertBegin(struct Node\*\*headL,int val)

{

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

newNode->data=val;

newNode->add=\*headL;

\*headL=newNode;

}

void insertEnd(struct Node\*\*headL,int val)

{

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

newNode->data=val;

newNode->add=NULL;

if (\*headL==NULL)

{

\*headL=newNode;

}

else

{

struct Node\*temp = \*headL;

while(temp->add!=NULL)

{

temp=temp->add;

}

temp->add=newNode;

}

}

struct Node\* deleteBegin(struct Node\* head)

{

if (head == NULL)

return NULL;

struct Node\* temp = head;

head = head->add;

free(temp);

return head;

}

struct Node\* deleteEnd(struct Node\* head)

{

if (head == NULL)

return NULL;

if (head->add == NULL) {

free(head);

return NULL;

}

struct Node\* second\_last = head;

while (second\_last->add->add != NULL)

second\_last = second\_last->add;

free(second\_last->add);

second\_last->add = NULL;

return head;

}

void displayList(struct Node\* head) {

if (head == NULL) {

printf("Linked list is empty.\n");

return;

}

printf("Elements in the linked list:\n");

while (head != NULL) {

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

if (head->add != NULL)

printf(" -> ");

head = head->add;

}

}

int main()

{

struct Node\*head=NULL;

char c;

scanf("\n%c",&c);

while(c!='a')

{

if (c=='b')

{

int num;

scanf(" %d",&num);

insertBegin(&head,num);

}

else if (c=='c')

{

int num;

scanf(" %d",&num);

insertEnd(&head,num);

}

else if (c=='e')

{

head = deleteBegin(head);

}

else if (c=='f')

{

head = deleteEnd(head);

}

else

{

printf("!!! ERROR !!!");

break;

}

scanf("\n%c",&c);

}

if (c=='a')

{

displayList(head);

}

}

15.Insertion of element at specific position in linked list:

#include <stdio.h>

#include <stdlib.h>

struct node

{

int data;

struct node\*next;

};

struct node\*head=NULL;

void inbeg()

{

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

scanf("%d",&newnode->data);

newnode->next=head;

head=newnode;

}

void inend()

{

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

scanf("%d",&newnode->data);

newnode->next=NULL;

if(head==NULL)

{

head=newnode;

return;

}

struct node\*temp=head;

while(temp->next!=NULL)

{

temp=temp->next;

}

temp->next=newnode;

}

void display()

{

struct node\*temp=head;

while(temp!=NULL)

{

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

temp=temp->next;

printf("-> ");

}

printf("NULL");

}

void inpos()

{

int pos,i=1;

struct node\*temp;

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

scanf("%d",&newnode->data);

scanf("%d",&pos);

temp=head;

while(i<pos-1)

{

temp=temp->next;

i++;

}

newnode->next=temp->next;

temp->next=newnode;

}

int main()

{

char ch;

do{

scanf("%c",&ch);

switch(ch)

{

case 'a':

display();

break;

case 'b':

inbeg();

break;

case 'c':

inend();

break;

case 'd':

inpos();

break;

case 'e':

exit(0);

break;

}

}while(ch!='a');

return 0;

}

16.Deletion of an element at specific position from linked list:

#include <stdio.h>

#include <stdlib.h>

struct node

{

int data;

struct node\*next;

};

struct node\*head=NULL;

struct node\*temp;

void inbeg()

{

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

scanf("%d",&newnode->data);

newnode->next=head;

head=newnode;

}

void inend()

{

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

scanf("%d",&newnode->data);

newnode->next=NULL;

if(head==NULL)

{

head=newnode;

return;

}

temp=head;

while(temp->next!=NULL)

{

temp=temp->next;

}

temp->next=newnode;

}

void display()

{

temp=head;

while(temp!=NULL)

{

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

temp=temp->next;

printf("-> ");

}

printf("NULL");

}

void frompos()

{

struct node\*nextnode;

int pos,i=1;

scanf("%d",&pos);

temp=head;

while(i<pos-1)

{

temp=temp->next;

i++;

}

nextnode=temp->next;

temp->next=nextnode->next;

free(nextnode);

}

int main()

{

char ch;

do{

scanf("%c",&ch);

switch(ch)

{

case 'a':

display();

break;

case 'b':

inbeg();

break;

case 'c':

inend();

break;

case 'd':

frompos();

break;

}

}while(ch!='a');

return 0;

}

17.Doubly Linked list Operations:

#include <stdio.h>

#include <stdlib.h>

struct node{

int data;

struct node\* next;

struct node\* prev;

};

void display(struct node\* ptr)

{

printf("Elements in the linked list :\n");

while(ptr!=NULL)

{

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

if(ptr->next!=NULL)

{

printf(" <-> ");

}

ptr=ptr->next;

}

}

struct node\* insertatfirst(struct node\* head,int data)

{

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

if(head==NULL)

{

ptr->data=data;

ptr->prev=NULL;

ptr->next=head;

return ptr;

}

else

{

ptr->data=data;

ptr->prev=head->prev;

ptr->next=head;

return ptr;

}

};

struct node\* insertatend(struct node\*head,int data)

{

struct node\*ptr=head;

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

while(ptr->next!=NULL)

{

ptr=ptr->next;

}

p->data=data;

p->next=ptr->next;

ptr->next=p;

return head;

};

struct node\* insertatindex(struct node\* head,int n,int data)

{

int i=0;

struct node\* ptr=head;

struct node\* q=head->next;

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

while(i!=n-1)

{

ptr=ptr->next;

i++;

}

p->data=data;

p->next=ptr->next;

p->prev=ptr->prev;

q->prev=p;

ptr->next=p;

return head;

};

struct node\* deleteatfirst(struct node\* head)

{

struct node\* ptr=head;

head=ptr->next;

free(ptr);

return head;

};

struct node\* deleteatend(struct node\* head)

{

struct node\* p=head;

// struct node\* q=head->next;

while(p->next->next!=NULL)

{

p=p->next;

}

p->next=NULL;

return head;

};

struct node\* deleteatindex(struct node\* head,int n)

{

int i;

struct node\* p=head;

struct node\* q=head->next;

while(i!=n-1)

{

p=p->next;

q=q->next;

}

p->next=q->next;

free(q);

return head;

};

int main()

{

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

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

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

head=NULL;

int n=1,x;

char ch;

while(n==1)

{

scanf(" %c",&ch);

switch(ch)

{

case 'a':

{

display(head);

n=0;

break;

}

case 'b':

{

scanf("%d",&x);

head=insertatfirst(head,x);

break;

}

case 'c':

{

scanf("%d",&x);

head=insertatend(head,x);

break;

}

case 'e':

{

head=deleteatfirst(head);

break;

}

case 'f':

{

head=deleteatend(head);

break;

}

default:

{

printf("!!! ERROR !!!");

n=0;

}

}

}

return 0;

}

18.BST creation and deletion:

#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;

}

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;

}

void inorder(struct Node\* root) {

if (root == NULL)

return;

inorder(root->left);

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

inorder(root->right);

}

struct Node\* minValueNode(struct Node\* node) {

struct Node\* current = node;

while (current && current->left != NULL)

current = current->left;

return current;

}

struct Node\* deleteNode(struct Node\* root, int key) {

if (root == NULL)

return root;

if (key < root->data)

root->left = deleteNode(root->left, key);

else if (key > root->data)

root->right = deleteNode(root->right, key);

else {

if (root->left == NULL) {

struct Node\* temp = root->right;

free(root);

return temp;

} else if (root->right == NULL) {

struct Node\* temp = root->left;

free(root);

return temp;

}

struct Node\* minValue = minValueNode(root->right);

root->data = minValue->data;

root->right = deleteNode(root->right, minValue->data);

}

return root;

}

int main() {

struct Node\* root = NULL;

int numElements, data;

char option, newline;

scanf("%d", &numElements);

for (int i = 0; i < numElements; i++) {

scanf("%d", &data);

root = insertNode(root, data);

}

scanf(" %c", &option);

if (option == 'a') {

int nodeToDelete;

scanf("%d", &nodeToDelete);

root = deleteNode(root, nodeToDelete);

scanf(" %c", &option);

}

if (option == 'b') {

printf("Inorder: ");

inorder(root);

printf("\n");

}

return 0;

}

19.BST Traversal:

#include <stdio.h>

#include <stdlib.h>

// Structure for a node of the binary search tree

struct Node {

int data;

struct Node\* left;

struct Node\* right;

};

// Function to create a new node

struct Node\* createNode(int item) {

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

newNode->data = item;

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

return newNode;

}

// Function to insert a new node into the binary search tree

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;

}

// Function to perform the delete operation in a binary search tree

struct Node\* deleteNode(struct Node\* root, int key) {

if (root == NULL)

return root;

if (key < root->data)

root->left = deleteNode(root->left, key);

else if (key > root->data)

root->right = deleteNode(root->right, key);

else {

if (root->left == NULL) {

struct Node\* temp = root->right;

free(root);

return temp;

}

else if (root->right == NULL) {

struct Node\* temp = root->left;

free(root);

return temp;

}

struct Node\* temp = root->right;

while (temp->left != NULL)

temp = temp->left;

root->data = temp->data;

root->right = deleteNode(root->right, temp->data);

}

return root;

}

// Function to print the elements of the binary search tree in order

void inorderTraversal(struct Node\* root) {

if (root != NULL) {

inorderTraversal(root->left);

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

inorderTraversal(root->right);

}

}

int main() {

struct Node\* root = NULL;

int numElements, data, i;

char option;

int nodeToDelete;

// Read the number of elements in the tree

scanf("%d", &numElements);

// Read the tree data and create the binary search tree

for (i = 0; i < numElements; i++) {

scanf("%d", &data);

root = insertNode(root, data);

}

// Read the menu option

scanf(" %c", &option);

if (option == 'a') {

// Read the node to be deleted

scanf("%d", &nodeToDelete);

root = deleteNode(root, nodeToDelete);

}

// Print the elements in order

printf("Inorder: ");

inorderTraversal(root);

printf("\n");

return 0;

}