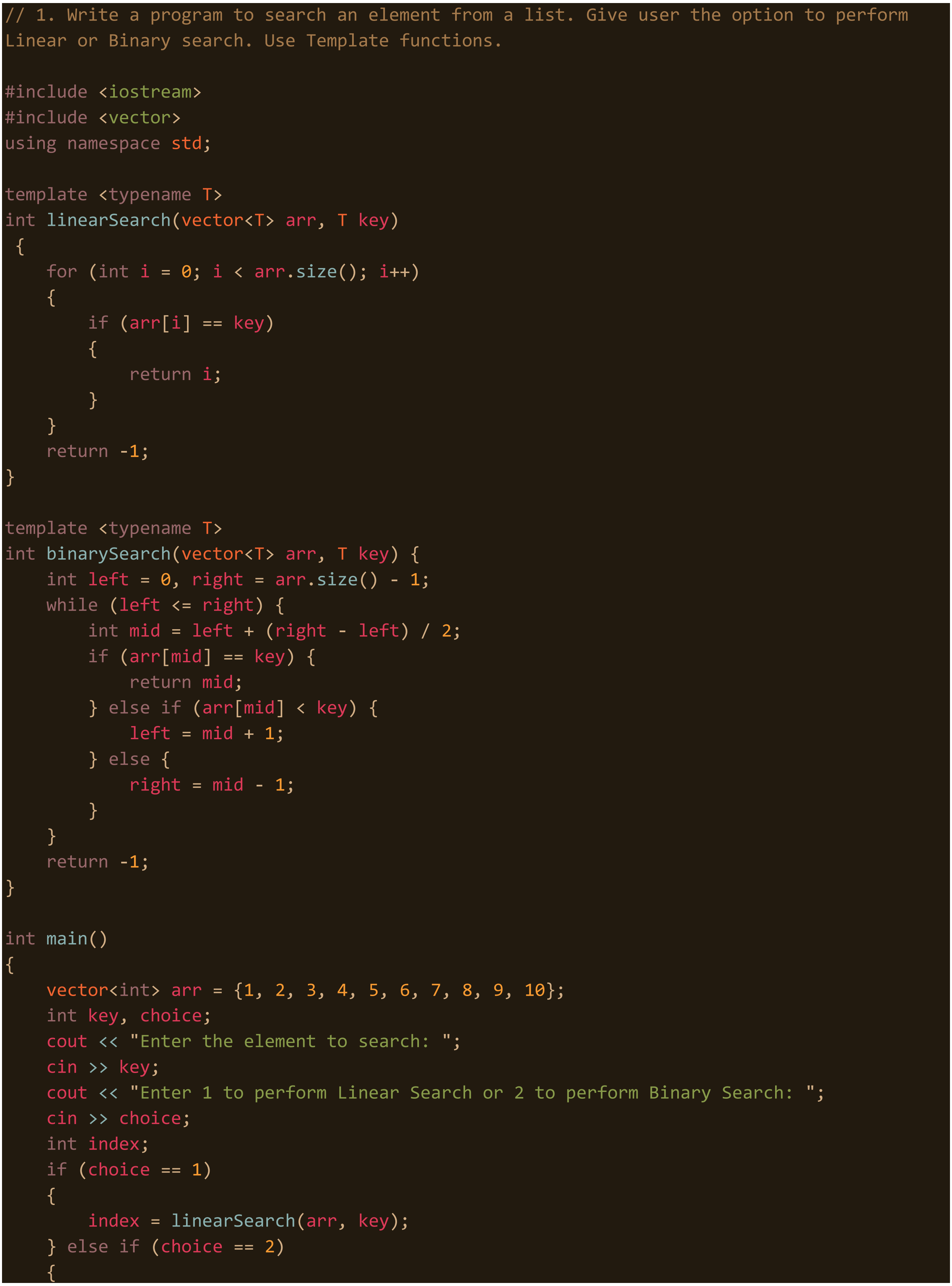
Data Structure 2021

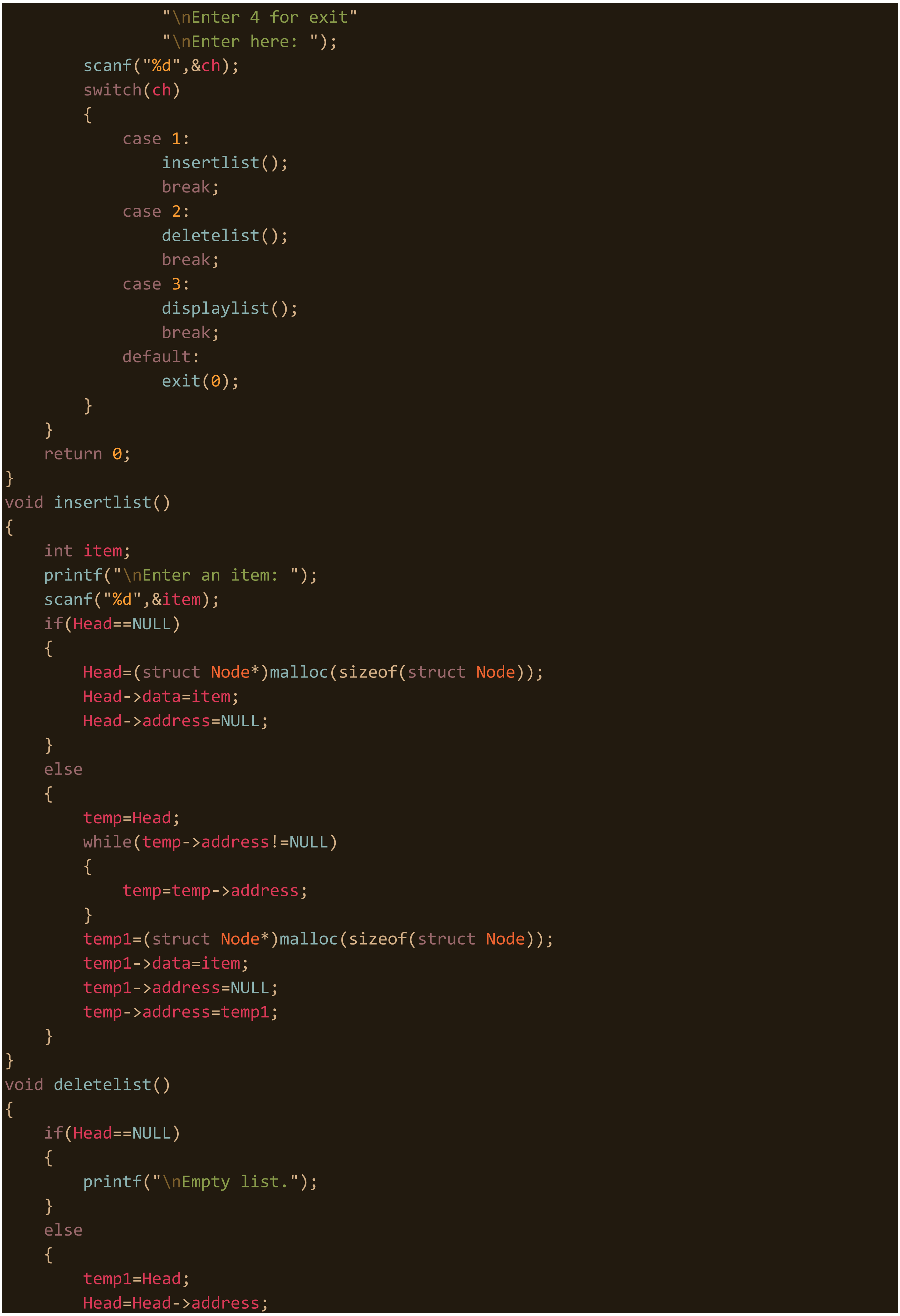


```
// 1. Write a program to search an element from a list. Give user the option to perform
Linear or Binary search. Use Template functions.

#include <iostream>
#include <vector>
using namespace std;

template <typename T>
int linearSearch(vector<T> arr, T key)
 {
    for (int i = 0; i < arr.size(); i++)
    {
        if (arr[i] == key)
        {
            return i;
        }
    }
    return -1;
}

template <typename T>
int binarySearch(vector<T> arr, T key) {
    int left = 0, right = arr.size() - 1;
    while (left <= right) {
        int mid = left + (right - left) / 2;
        if (arr[mid] == key) {
            return mid;
        } else if (arr[mid] < key) {
            left = mid + 1;
        } else {
            right = mid - 1;
        }
    }
    return -1;
}

int main()
{
    vector<int> arr = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
    int key, choice;
    cout << "Enter the element to search: ";
    cin >> key;
    cout << "Enter 1 to perform Linear Search or 2 to perform Binary Search: ";
    cin >> choice;
    int index;
    if (choice == 1)
    {
        index = linearSearch(arr, key);
    } else if (choice == 2)
    {
        index = binarySearch(arr, key);
    }
    else
    {
        cout << "Invalid choice!" << endl;
        return 0;
    }
    if (index == -1)
    {
        cout << "Element not found!" << endl;
    } else
    {
        cout << "Element found at index " << index << endl;
    }
    return 0;
}
```

```
// 2. Write a program to sort a list of elements. Give user the option to perform sorting
using Insertion sort, Bubble sort or Selection sort.

#include<stdio.h>
void Insertion_sort(int[],int);
void Bubble_sort(int[],int);
void Selection_sort(int[],int);
int main()
{
    int a[10],i,n,c;
    printf("\nEnter length of array: ");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
        printf("\nEnter array element: ");
        scanf("%d",&a[i]);
    }
    printf("\nEnter 1 for using Insertion sort"
            "\nEnter 2 for using Selection sort"
            "\nEnter 3 for using Bubble sort"
            "\nEnter here: ");
    scanf("%d",&c);
    switch(c)
    {
        case 1:
            Insertion_sort(a,n);
            break;
        case 2:
            Selection_sort(a,n);
            break;
        case 3:
            Bubble_sort(a,n);
            break;
        default:
            break;
    }
    printf("\nYour array is: \n");
    for(i=0;i<n;i++)
    {
        printf("%5d",a[i]);
    }
    printf("\nSorted array is: \n");
    for(i=0;i<n;i++)
    {
        printf("%5d",a[i]);
    }
    return 0;
}
void Insertion_sort(int a[10],int n)
{
    int i,j,lock;
    for(i=1;i<n;i++)
    {
        lock=a[i];
        j=i-1;
        while(j>=0 && a[j]>lock)
        {
            a[j+1]=a[j];
            j=j-1;
        }
        a[j+1]=lock;
    }
}

void Bubble_sort(int a[10],int n)
{
    int i,j,temp,flag;
    for(i=0;i<n;i++)
    {
        flag=0;
        for(j=0;j<n-i-1;j++)
        {
            if(a[j]>a[j+1])
            {
                flag=1;
                temp=a[j];
                a[j]=a[j+1];
                a[j+1]=temp;
            }
        }
        if(flag==0)
        {
            break;
        }
    }
}

void Selection_sort(int a[10],int n)
```

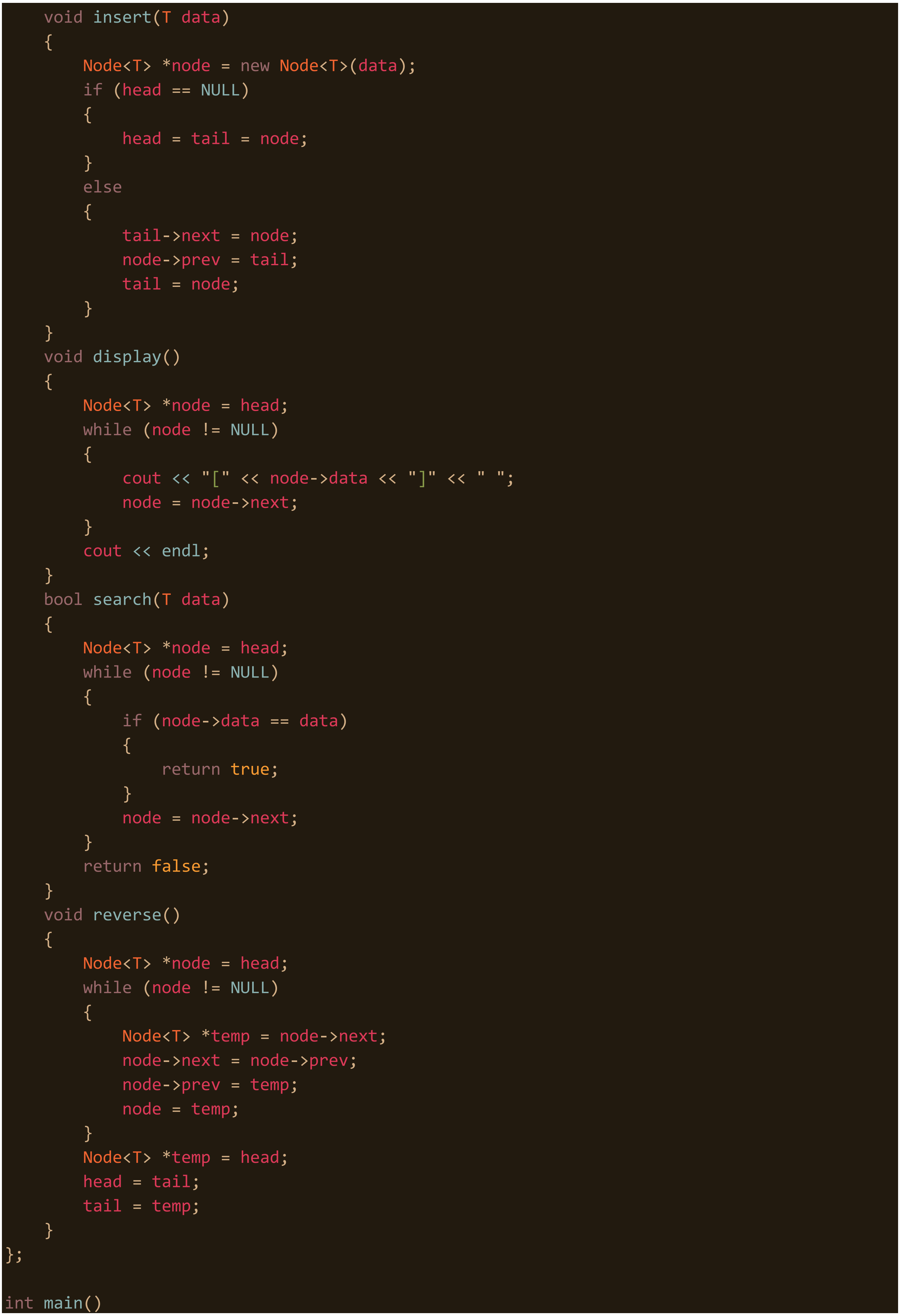
```
{
    int i,j,temp,flag,max,p;
    for(i=0;i<n;i++)
    {
        p=0;
        max=a[p];
        for(j=0;j<n-i-1;j++)
        {
            if(max<a[j+1])
            {
                p=j+1;
                max=a[j+1];
                flag=0;
            }
            else
            {
                flag=1;
            }

        }
        temp=a[p];
        a[p]=a[j];
        a[j]=temp;
        if(flag==0)
        {
            break;
        }
    }
}
```

```
// 3. Write a program to implement Linked List. Include functions for insertion, deletion.

#include<stdio.h>
#include<process.h>
#include<malloc.h>
struct Node
{
    int data;
    struct Node *address;
}*Head, *temp, *temp1;
void insertlist();
void deletelist();
void displaylist();
int main()
{
    int ch;
    while(1)
    {
        printf("\nPress 1 for Insert"
                "\nPress 2 for Delete"
                "\nPress 3 for Display List"
```

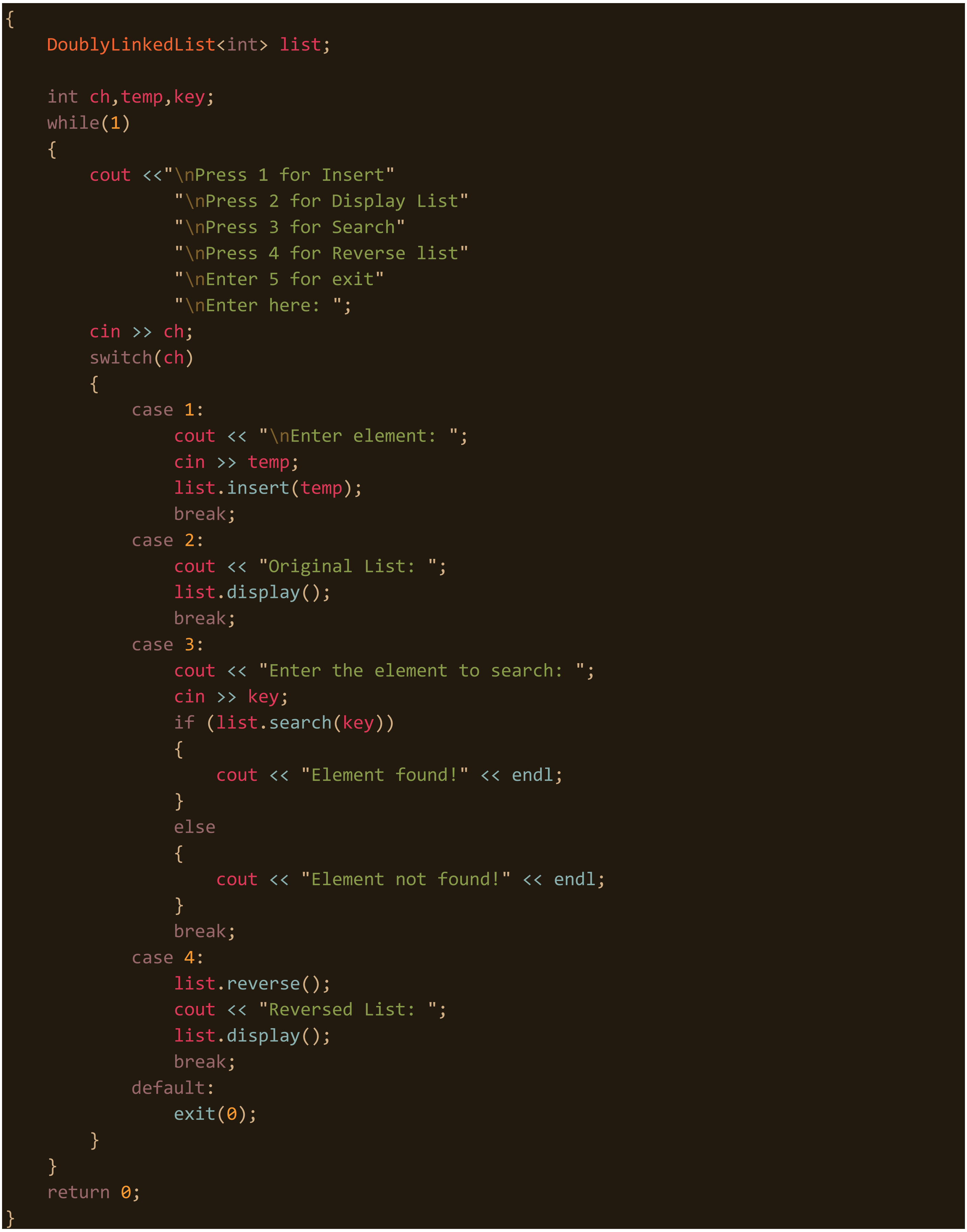


```
                "\nEnter 4 for exit"
                "\nEnter here: ");
        scanf("%d",&ch);
        switch(ch)
        {
            case 1:
                insertlist();
                break;
            case 2:
                deletelist();
                break;
            case 3:
                displaylist();
                break;
            default:
                exit(0);
        }
    }
    return 0;
}
void insertlist()
{
    int item;
    printf("\nEnter an item: ");
    scanf("%d",&item);
    if(Head==NULL)
    {
        Head=(struct Node*)malloc(sizeof(struct Node));
        Head->data=item;
        Head->address=NULL;
    }
    else
    {
        temp=Head;
        while(temp->address!=NULL)
        {
            temp=temp->address;
        }
        temp1=(struct Node*)malloc(sizeof(struct Node));
        temp1->data=item;
        temp1->address=NULL;
        temp->address=temp1;
    }
}
void deletelist()
{
    if(Head==NULL)
    {
        printf("\nEmpty list.");
    }
    else
    {
        temp1=Head;
        Head=Head->address;
        temp1->address=NULL;
        printf("\n%d is deleted.",temp1->data);
        free(temp1);
    }
}
void displaylist()
{
    if(Head==NULL)
    {
        printf("\nEmpty list.");
    }
    else
    {
        temp1=Head;
        while(temp1->address!=NULL)
        {
            printf("[%d]-> ",temp1->data);
            temp1=temp1->address;
        }
        printf("[%d]",temp1->data);
    }
}
```

```
//4.  Implement Doubly Linked List using templates.Include functions for searching of a
number, and reverse the list.

#include <iostream>
using namespace std;

template <typename T>
class Node
{
public:
    T data;
    Node<T> *prev, *next;
    Node(T data)
    {
        this->data = data;
        prev = next = NULL;
    }
};

template <typename T>
class DoublyLinkedList
{
public:
    Node<T> *head, *tail;
    DoublyLinkedList()
    {
        head = tail = NULL;
    }
```

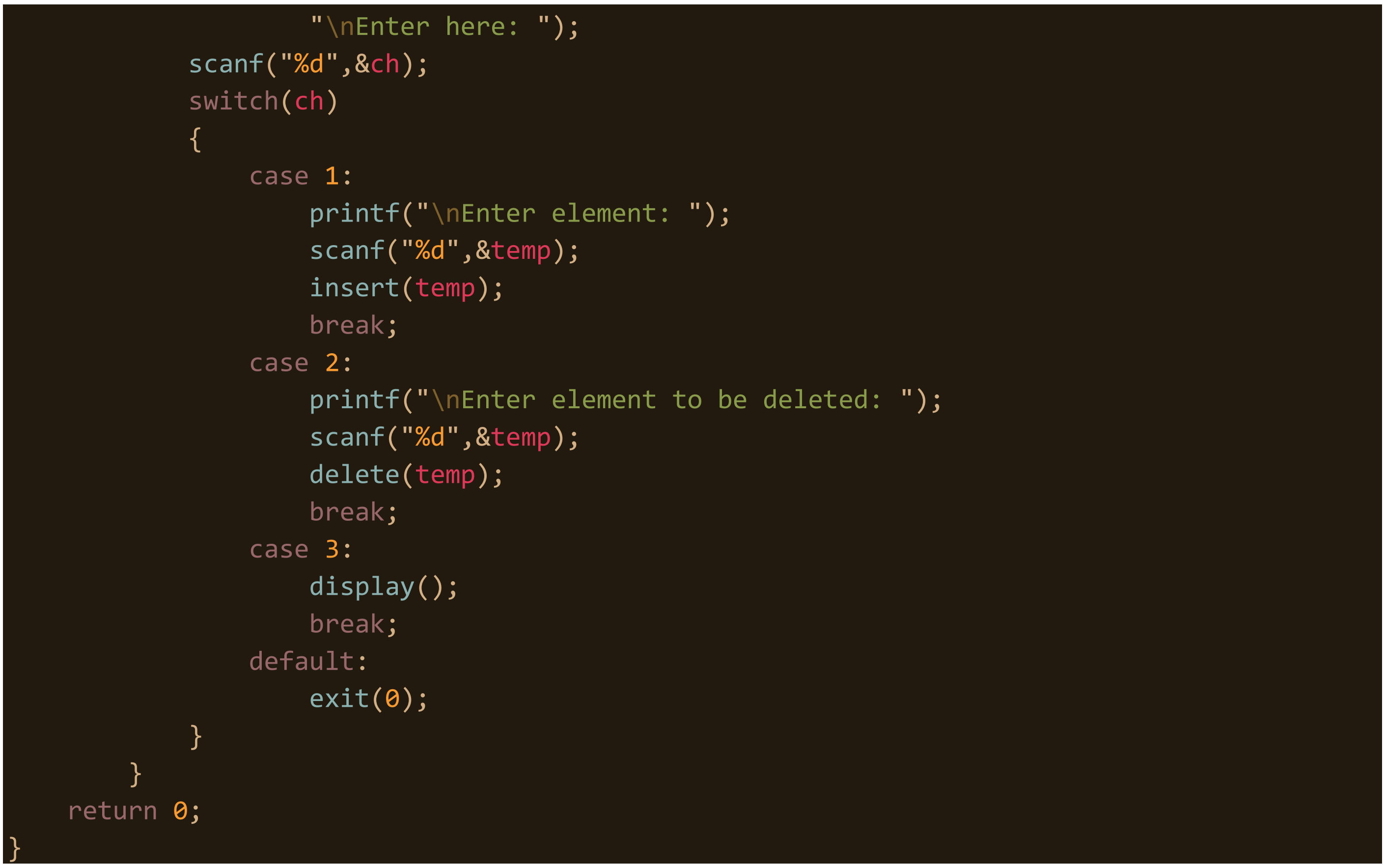


```
    void insert(T data)
    {
        Node<T> *node = new Node<T>(data);
        if (head == NULL)
        {
            head = tail = node;
        }
        else
        {
            tail->next = node;
            node->prev = tail;
            tail = node;
        }
    }
    void display()
    {
        Node<T> *node = head;
        while (node != NULL)
        {
            cout << "[" << node->data << "]" << " ";
            node = node->next;
        }
        cout << endl;
    }
    bool search(T data)
    {
        Node<T> *node = head;
        while (node != NULL)
        {
            if (node->data == data)
            {
                return true;
            }
            node = node->next;
        }
        return false;
    }
    void reverse()
    {
        Node<T> *node = head;
        while (node != NULL)
        {
            Node<T> *temp = node->next;
            node->next = node->prev;
            node->prev = temp;
            node = temp;
        }
        Node<T> *temp = head;
        head = tail;
        tail = temp;
    }
};

int main()
```

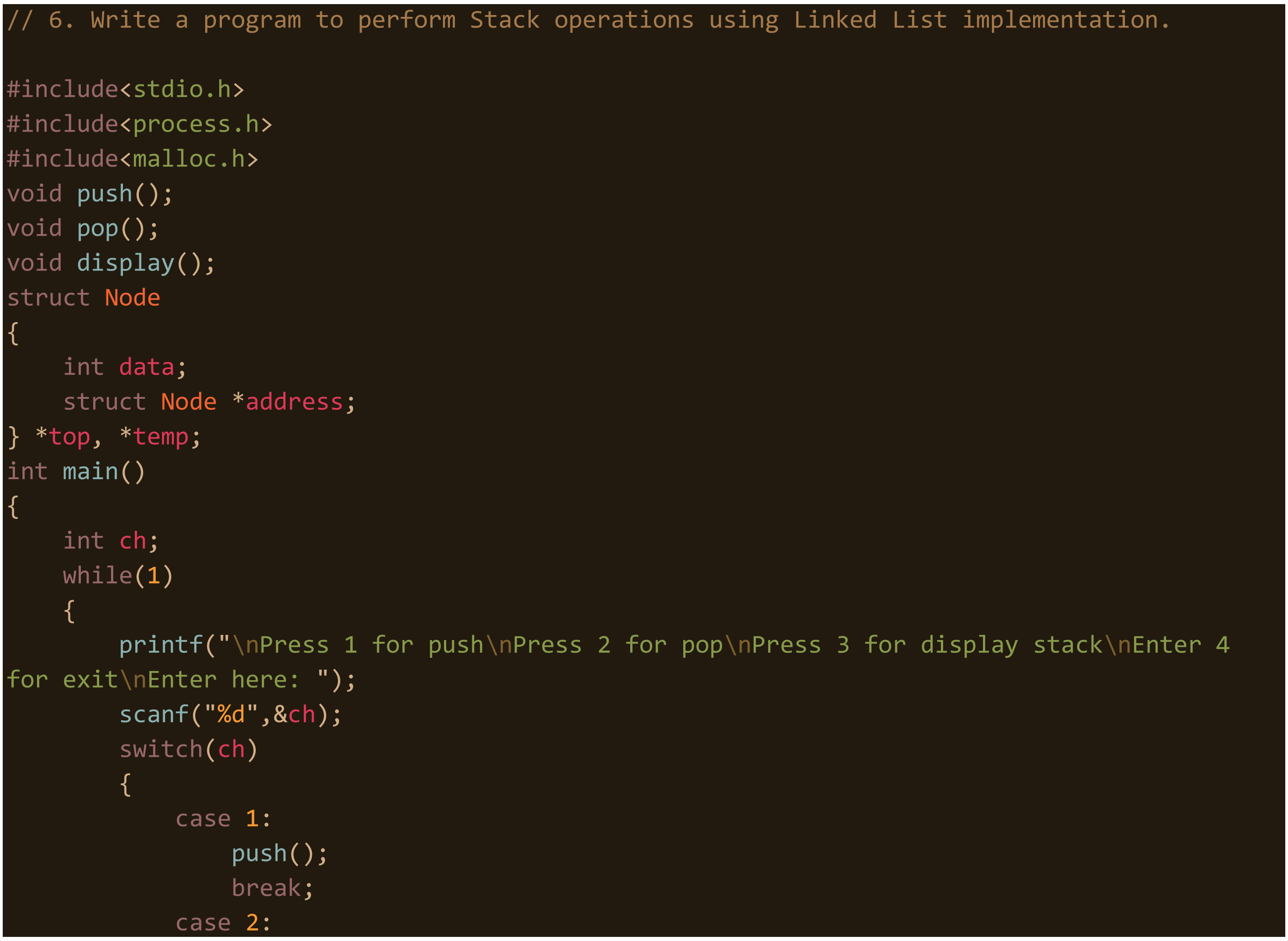


```
{
    DoublyLinkedList<int> list;

    int ch,temp,key;
    while(1)
    {
        cout <<"\nPress 1 for Insert"
                "\nPress 2 for Display List"
                "\nPress 3 for Search"
                "\nPress 4 for Reverse list"
                "\nEnter 5 for exit"
                "\nEnter here: ";
        cin >> ch;
        switch(ch)
        {
            case 1:
                cout << "\nEnter element: ";
                cin >> temp;
                list.insert(temp);
                break;
            case 2:
                cout << "Original List: ";
                list.display();
                break;
            case 3:
                cout << "Enter the element to search: ";
                cin >> key;
                if (list.search(key))
                {
                    cout << "Element found!" << endl;
                }
                else
                {
                    cout << "Element not found!" << endl;
                }
                break;
            case 4:
                list.reverse();
                cout << "Reversed List: ";
                list.display();
                break;
            default:
                exit(0);
        }
    }
    return 0;
}
```

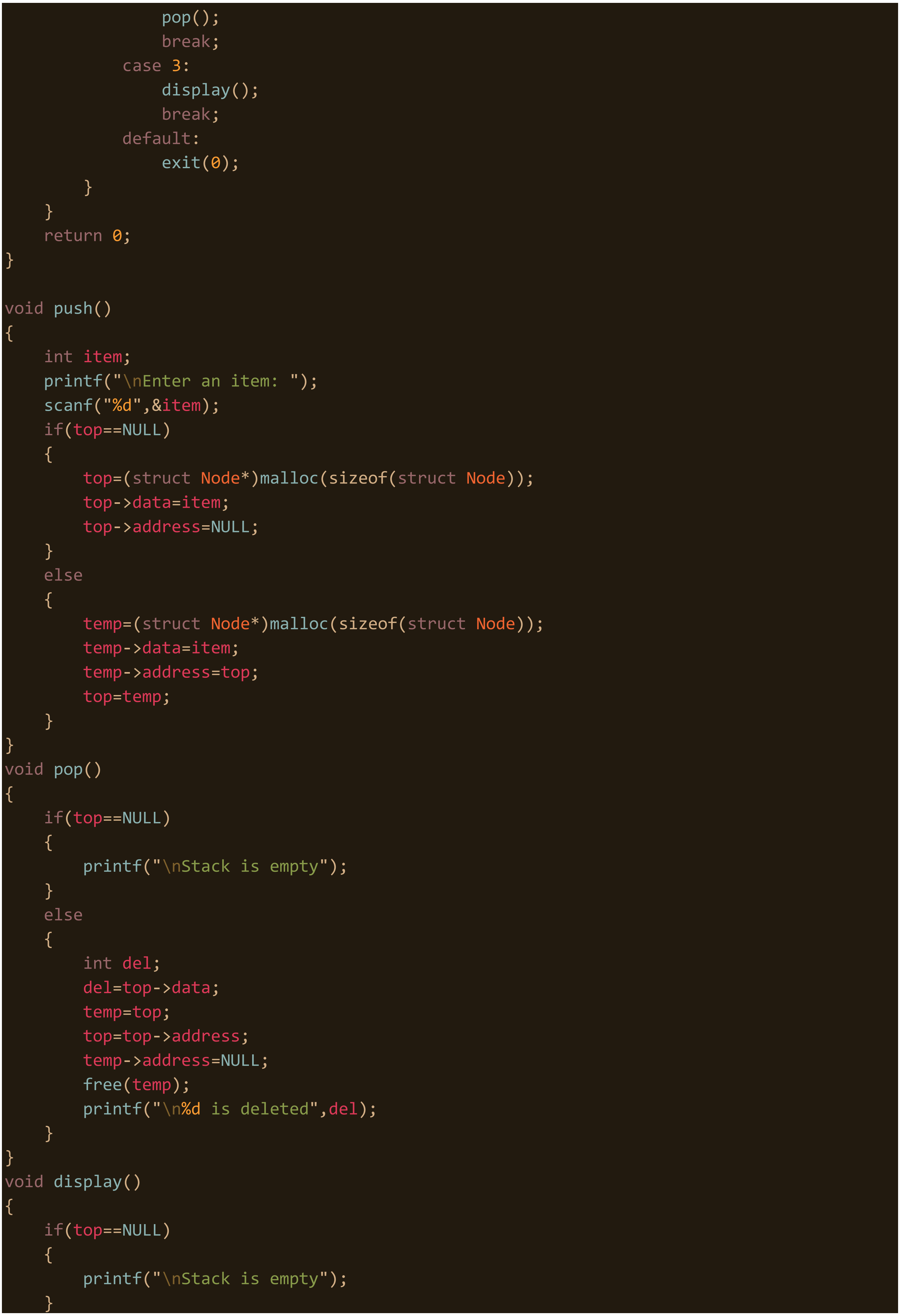
```c
// 5. Write a program to implement Circular Linked List. Include functions for insertion
and deletion.

#include <stdio.h>
#include <stdlib.h>

struct Node
{
    int data;
    struct Node *next;
};

struct Node *head = NULL;

void insert(int data)
{
    struct Node *node = (struct Node*) malloc(sizeof(struct Node));
    node->data = data;
    if (head == NULL)
    {
        head = node;
        node->next = head;
    }
    else
    {
        struct Node *temp = head;
        while (temp->next != head)
        {
            temp = temp->next;
        }
        temp->next = node;
        node->next = head;
    }
}

void delete(int data)
{
    if (head == NULL)
    {
        printf("List is empty!\n");
        return;
    }
    struct Node *temp = head, *prev;
    while (temp->data != data)
    {
        if (temp->next == head)
        {
            printf("Element not found!\n");
            return;
        }
        prev = temp;
        temp = temp->next;
    }
    if (temp == head && temp->next == head)
    {
        head = NULL;
        free(temp);
        return;
    }
    if (temp == head)
    {
        prev = head;
        while (prev->next != head)
        {
            prev = prev->next;
        }
        head = temp->next;
        prev->next = head;
        free(temp);
    }
    else if (temp->next == head)
    {
        prev->next = head;
        free(temp);
    }
    else
    {
        prev->next = temp->next;
        free(temp);
    }
}

void display()
{
    struct Node *temp = head;
    if (head == NULL)
    {
        printf("List is empty!\n");
        return;
    }
    printf("List: ");
    do
    {
        printf("%d ", temp->data);
        temp = temp->next;
    } while (temp != head);
    printf("\n");
}

int main()
{
    int ch,temp;
        while(1)
        {
            printf("\nPress 1 for Insert"
                    "\nPress 2 for Delete"
                    "\nPress 3 for Display List"
                    "\nEnter 4 for exit"
```



```
                    "\nEnter here: ");
            scanf("%d",&ch);
            switch(ch)
            {
                case 1:
                    printf("\nEnter element: ");
                    scanf("%d",&temp);
                    insert(temp);
                    break;
                case 2:
                    printf("\nEnter element to be deleted: ");
                    scanf("%d",&temp);
                    delete(temp);
                    break;
                case 3:
                    display();
                    break;
                default:
                    exit(0);
            }
        }
    return 0;
}
```



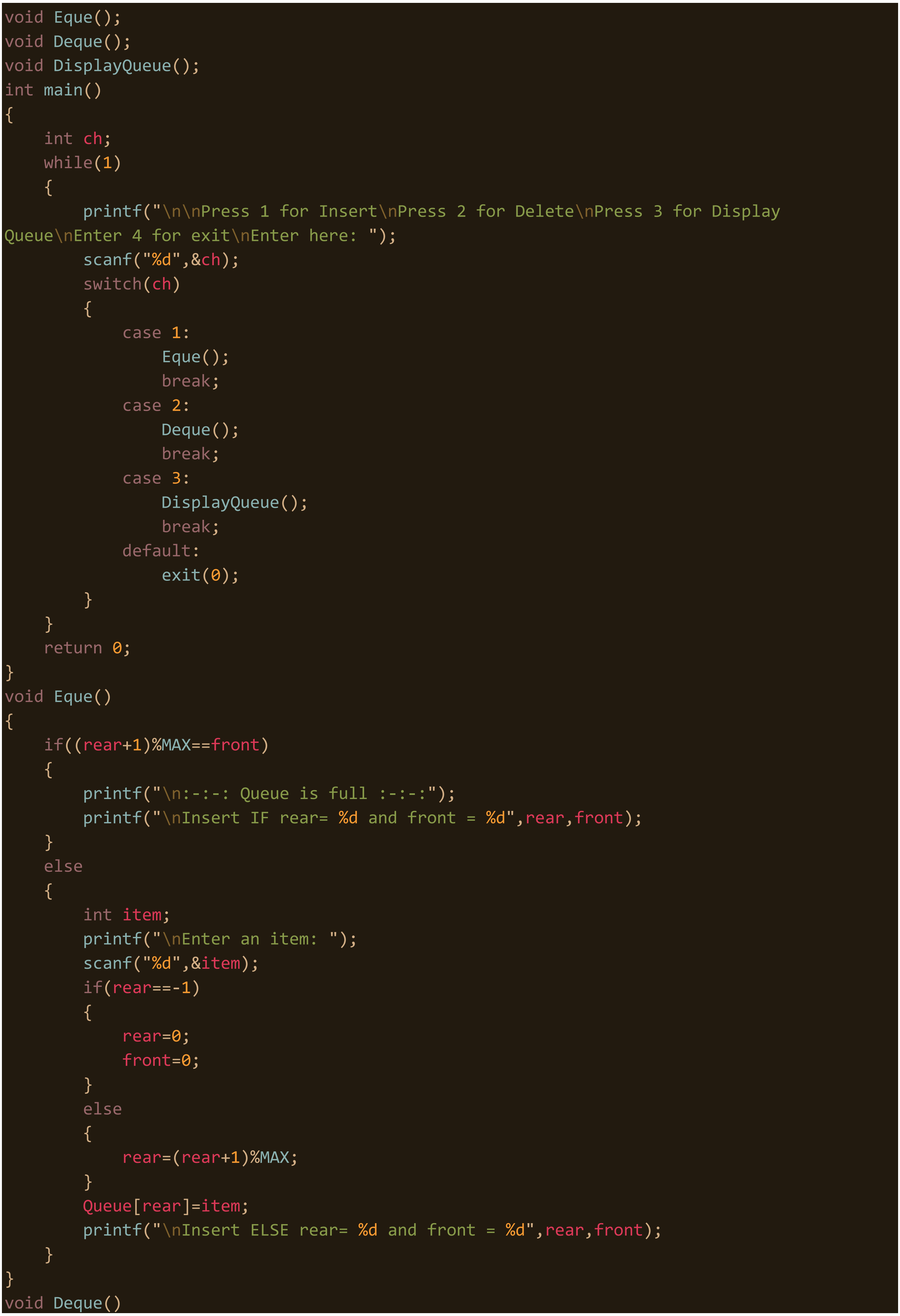
```
// 6. Write a program to perform Stack operations using Linked List implementation.

#include<stdio.h>
#include<process.h>
#include<malloc.h>
void push();
void pop();
void display();
struct Node
{
    int data;
    struct Node *address;
} *top, *temp;
int main()
{
    int ch;
    while(1)
    {
        printf("\nPress 1 for push\nPress 2 for pop\nPress 3 for display stack\nEnter 4
for exit\nEnter here: ");
        scanf("%d",&ch);
        switch(ch)
        {
            case 1:
                push();
                break;
            case 2:
```



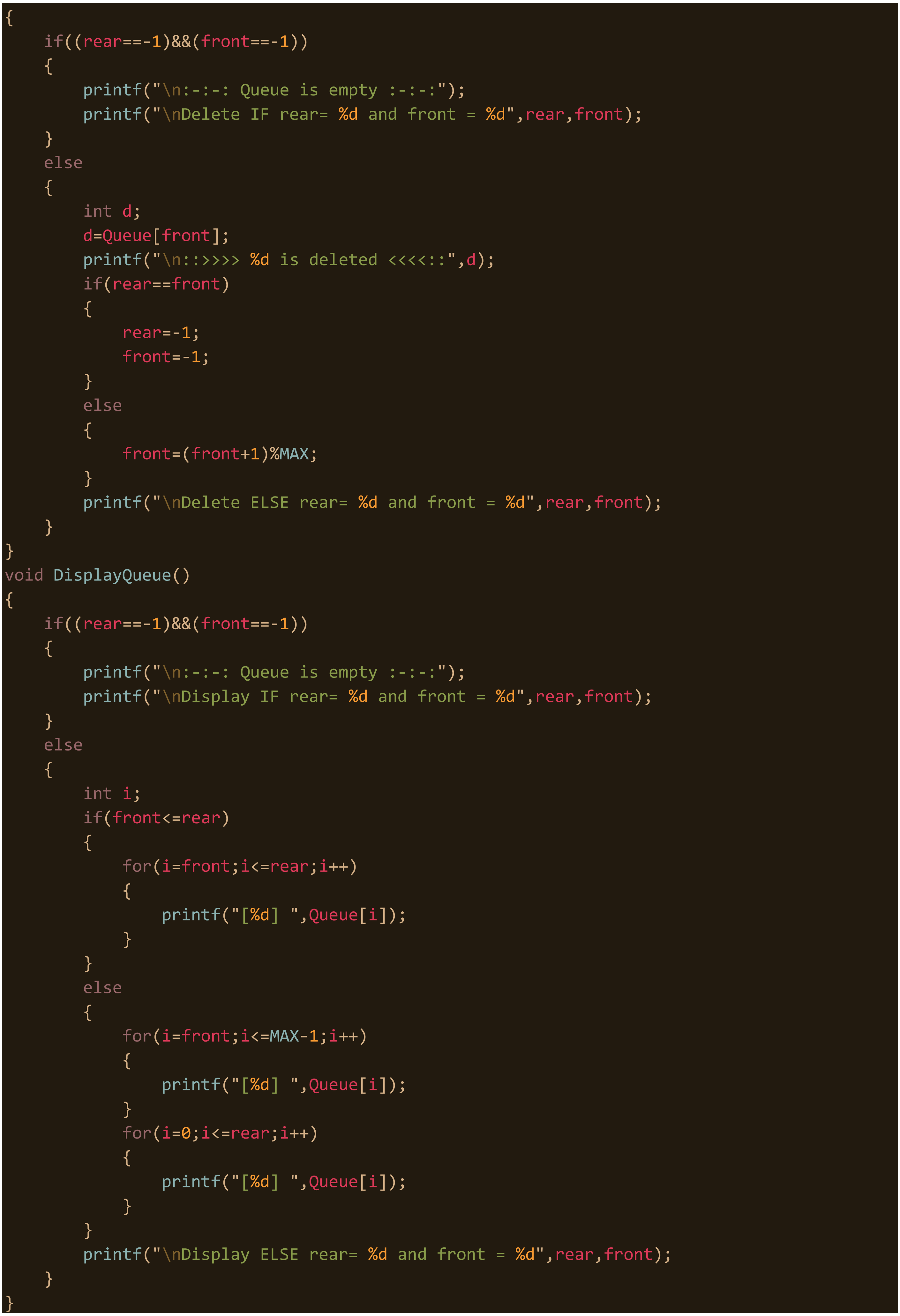
```c
                pop();
                break;
            case 3:
                display();
                break;
            default:
                exit(0);
        }
    }
    return 0;
}

void push()
{
    int item;
    printf("\nEnter an item: ");
    scanf("%d",&item);
    if(top==NULL)
    {
        top=(struct Node*)malloc(sizeof(struct Node));
        top->data=item;
        top->address=NULL;
    }
    else
    {
        temp=(struct Node*)malloc(sizeof(struct Node));
        temp->data=item;
        temp->address=top;
        top=temp;
    }
}
void pop()
{
    if(top==NULL)
    {
        printf("\nStack is empty");
    }
    else
    {
        int del;
        del=top->data;
        temp=top;
        top=top->address;
        temp->address=NULL;
        free(temp);
        printf("\n%d is deleted",del);
    }
}
void display()
{
    if(top==NULL)
    {
        printf("\nStack is empty");
    }
    else
    {
        temp=top;
        while(temp->address!=NULL)
        {
            printf("[%d]->",temp->data);
            temp=temp->address;
        }
        printf("[%d]",temp->data);
    }
}
```

```
// 7. Write a program to perform Stack operations using Array implementation.

#include<stdio.h>
#include<process.h>
#define MAX 4
int stack[4];
int top = -1;
void push();
void pop();
void display();
int main()
{
    int ch;
    while(1)
    {
        printf("\nPress 1 for push\nPress 2 for pop\nPress 3 for display stack\nEnter 4
for exit\nEnter here: ");
        scanf("%d",&ch);
        switch(ch)
        {
            case 1:
                push();
                break;
            case 2:
                pop();
                break;
            case 3:
                display();
                break;
            default:
                exit(0);
        }
    }
    return 0;
}
void push()
{
    if(top==MAX-1)
    {
        printf("\nStack is full");
    }
    else
    {
        int item;
        printf("\nEnter an item: ");
        scanf("%d",&item);
        top=top+1;
        stack[top]=item;
    }
}
void pop()
{
    if(top==-1)
    {
        printf("\nStack is empty");
    }
    else
    {
        int d;
        d=stack[top];
        top=top-1;
        printf("\n%d is deleted",d);
    }
}
void display()
{
    if(top==-1)
    {
        printf("\nString is empty");
    }
    else
    {
        int t;
        t=top;
        while(t>=0)
        {
            printf("\n[%d]",stack[t]);
            t--;
        }
    }
}
```

```
// 8. Write a program to perform Queue operations using Circular Array implementation.

#include<stdio.h>
#include<process.h>
#define MAX 4
int Queue[4];
int rear = -1;
int front = -1;
```



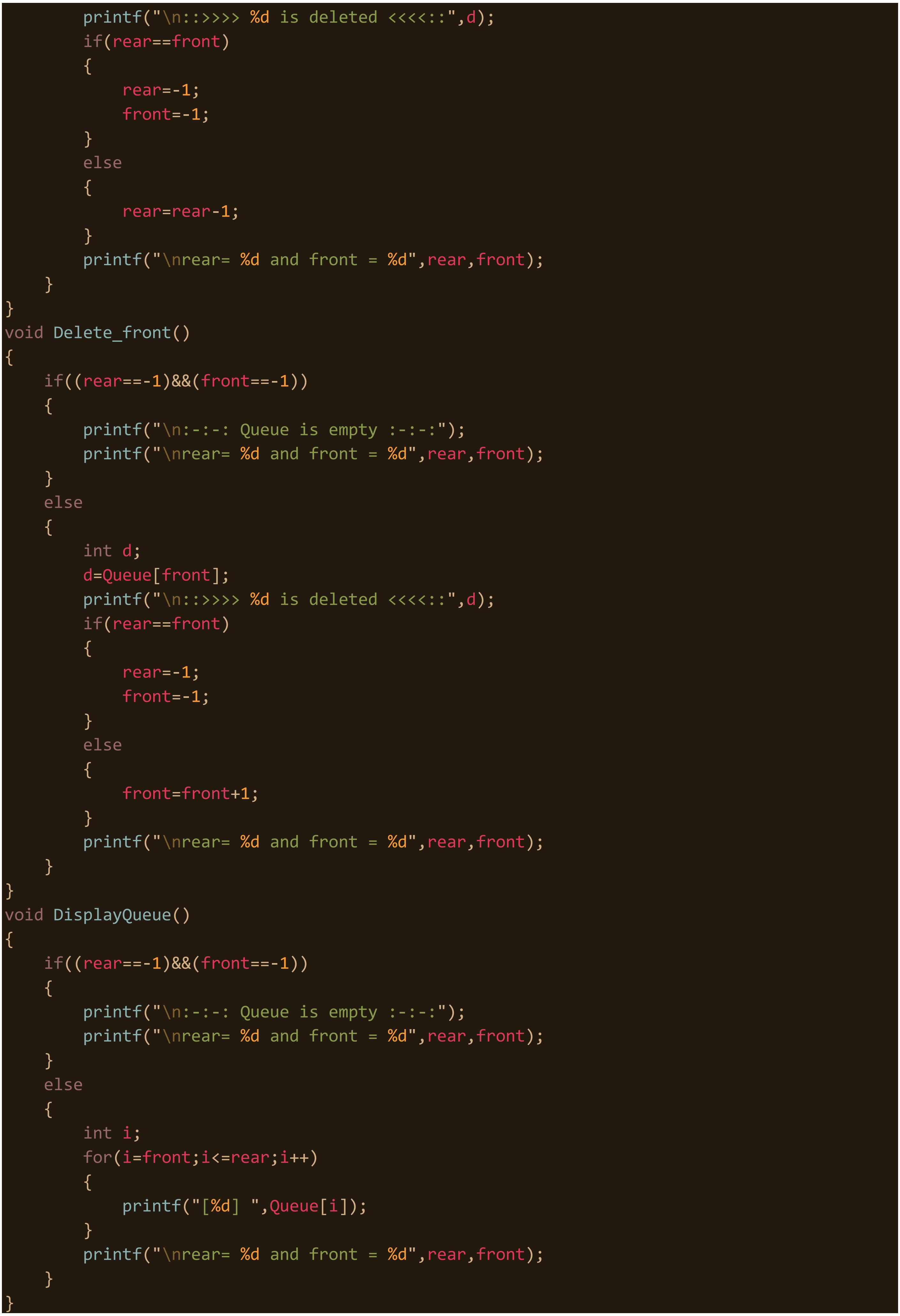
```
void Eque();
void Deque();
void DisplayQueue();
int main()
{
    int ch;
    while(1)
    {
        printf("\n\nPress 1 for Insert\nPress 2 for Delete\nPress 3 for Display
Queue\nEnter 4 for exit\nEnter here: ");
        scanf("%d",&ch);
        switch(ch)
        {
            case 1:
                Eque();
                break;
            case 2:
                Deque();
                break;
            case 3:
                DisplayQueue();
                break;
            default:
                exit(0);
        }
    }
    return 0;
}
void Eque()
{
    if((rear+1)%MAX==front)
    {
        printf("\n:-:-: Queue is full :-:-:");
        printf("\nInsert IF rear= %d and front = %d",rear,front);
    }
    else
    {
        int item;
        printf("\nEnter an item: ");
        scanf("%d",&item);
        if(rear==-1)
        {
            rear=0;
            front=0;
        }
        else
        {
            rear=(rear+1)%MAX;
        }
        Queue[rear]=item;
        printf("\nInsert ELSE rear= %d and front = %d",rear,front);
    }
}
void Deque()
```



```
{
    if((rear==-1)&&(front==-1))
    {
        printf("\n:-:-: Queue is empty :-:-:");
        printf("\nDelete IF rear= %d and front = %d",rear,front);
    }
    else
    {
        int d;
        d=Queue[front];
        printf("\n::>>>> %d is deleted <<<<::",d);
        if(rear==front)
        {
            rear=-1;
            front=-1;
        }
        else
        {
            front=(front+1)%MAX;
        }
        printf("\nDelete ELSE rear= %d and front = %d",rear,front);
    }
}
void DisplayQueue()
{
    if((rear==-1)&&(front==-1))
    {
        printf("\n:-:-: Queue is empty :-:-:");
        printf("\nDisplay IF rear= %d and front = %d",rear,front);
    }
    else
    {
        int i;
        if(front<=rear)
        {
            for(i=front;i<=rear;i++)
            {
                printf("[%d] ",Queue[i]);
            }
        }
        else
        {
            for(i=front;i<=MAX-1;i++)
            {
                printf("[%d] ",Queue[i]);
            }
            for(i=0;i<=rear;i++)
            {
                printf("[%d] ",Queue[i]);
            }
        }
        printf("\nDisplay ELSE rear= %d and front = %d",rear,front);
    }
}
```

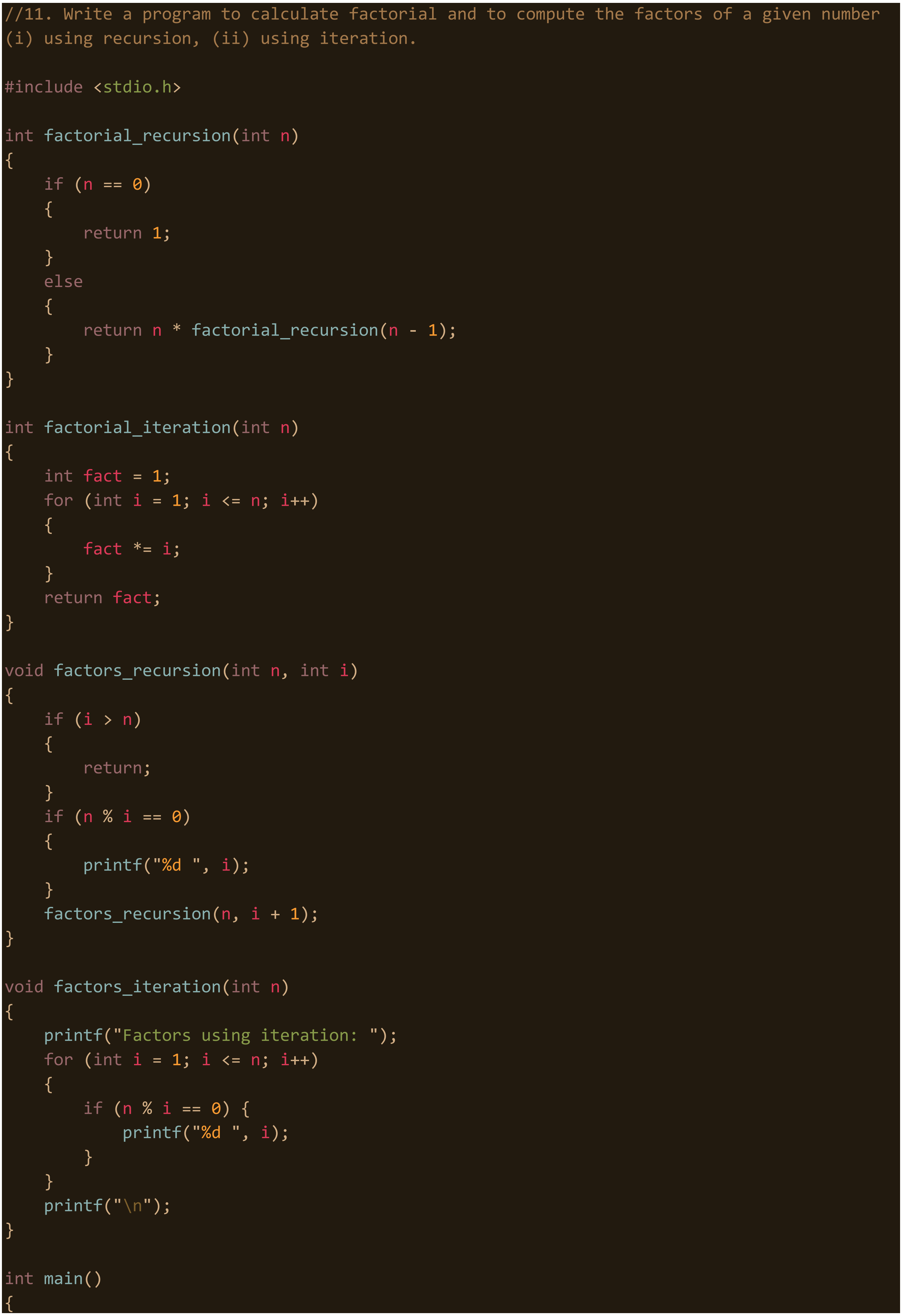
```
// 9. Write a program to create and perform different operations on Double-ended Queues
using Linked List implementation.

#include<stdio.h>
#include<process.h>
#define MAX 4
int Queue[4];
int rear = -1;
int front = -1;
void Insert_rear();
void Insert_front();
void Delete_rear();
void Delete_front();
void DisplayQueue();
int main()
{
    int ch;
    while(1)
    {
        printf("\n\nPress 1 for Insert from rear\nPress 2 for Insert from front\nPress 3
for Delete from rere\nEnter 4 for Delete from front\nEnter 5 for Display Queue\nEnter 6
for EXIT\nEnter here: ");
        scanf("%d",&ch);
        switch(ch)
        {
            case 1:
                Insert_rear();
                break;
            case 2:
                Insert_front();
                break;
            case 3:
                Delete_rear();
                break;
            case 4:
                Delete_front();
                break;
            case 5:
                DisplayQueue();
                break;
            default:
                exit(0);
        }
    }
    return 0;
}
void Insert_rear()
{
    if(rear==MAX-1)
    {
        printf("\n:-:-: Queue is full :-:-:");
        printf("\nrear= %d and front = %d",rear,front);
    }
    else
```

```c
    {
        int item;
        printf("\nEnter an item: ");
        scanf("%d",&item);
        if((rear==-1)&&(front==-1))
        {
            rear=0;
            front=0;
        }
        else
        {
            rear=rear+1;
        }
        Queue[rear]=item;
        printf("\nrear= %d and front = %d",rear,front);
    }
}
void Insert_front()
{
    if(front==0)
    {
        printf("\n:-:-: Queue is full :-:-:");
        printf("\nrear= %d and front = %d",rear,front);
    }
    else
    {
        int item;
        printf("\nEnter an item: ");
        scanf("%d",&item);
        if((rear==-1)&&(front==-1))
        {
            rear=0;
            front=0;
        }
        else
        {
            front=front+1;
        }
        Queue[front]=item;
        printf("\nrear= %d and front = %d",rear,front);

    }
}
void Delete_rear()
{
    if((rear==-1)&&(front==-1))
    {
        printf("\n:-:-: Queue is empty :-:-:");
        printf("\nrear= %d and front = %d",rear,front);
    }
    else
    {
        int d;
        d=Queue[rear];
```



```
        printf("\n::>>>> %d is deleted <<<<::",d);
        if(rear==front)
        {
            rear=-1;
            front=-1;
        }
        else
        {
            rear=rear-1;
        }
        printf("\nrear= %d and front = %d",rear,front);
    }
}
void Delete_front()
{
    if((rear==-1)&&(front==-1))
    {
        printf("\n:-:-: Queue is empty :-:-:");
        printf("\nrear= %d and front = %d",rear,front);
    }
    else
    {
        int d;
        d=Queue[front];
        printf("\n::>>>> %d is deleted <<<<::",d);
        if(rear==front)
        {
            rear=-1;
            front=-1;
        }
        else
        {
            front=front+1;
        }
        printf("\nrear= %d and front = %d",rear,front);
    }
}
void DisplayQueue()
{
    if((rear==-1)&&(front==-1))
    {
        printf("\n:-:-: Queue is empty :-:-:");
        printf("\nrear= %d and front = %d",rear,front);
    }
    else
    {
        int i;
        for(i=front;i<=rear;i++)
        {
            printf("[%d] ",Queue[i]);
        }
        printf("\nrear= %d and front = %d",rear,front);
    }
}
```

```c
//10. Write a program to represent a polynomial using linked list and add two polynomials.

#include <stdio.h>
#include <stdlib.h>

struct Node
{
    int coeff, exp;
    struct Node *next;
};

struct Node *head1 = NULL, *head2 = NULL, *head3 = NULL;

void insert(struct Node **head, int coeff, int exp)
{
    struct Node *node = (struct Node*) malloc(sizeof(struct Node));
    node->coeff = coeff;
    node->exp = exp;
    node->next = NULL;
    if (*head == NULL)
    {
        *head = node;
    }
    else
    {
        struct Node *temp = *head;
        while (temp->next != NULL)
        {
            temp = temp->next;
        }
        temp->next = node;
    }
}

void addPolynomials(struct Node *head1, struct Node *head2, struct Node **head3)
{
    while (head1 != NULL && head2 != NULL)
    {
        if (head1->exp == head2->exp)
        {
            insert(head3, head1->coeff + head2->coeff, head1->exp);
            head1 = head1->next;
            head2 = head2->next;
        } else if (head1->exp > head2->exp)
        {
            insert(head3, head1->coeff, head1->exp);
            head1 = head1->next;
        }
        else
        {
            insert(head3, head2->coeff, head2->exp);
            head2 = head2->next;
        }
    }
    while (head1 != NULL)
    {
        insert(head3, head1->coeff, head1->exp);
        head1 = head1->next;
    }
    while (head2 != NULL)
    {
        insert(head3, head2->coeff, head2->exp);
        head2 = head2->next;
    }
}

void display(struct Node *head)
{
    struct Node *temp = head;
    while (temp != NULL)
    {
        printf("%dx^%d", temp->coeff, temp->exp);
        temp = temp->next;
        if (temp != NULL)
        {
            printf(" + ");
        }
    }
    printf("\n");
}

int main()
{
    insert(&head1, 5, 2);
    insert(&head1, 4, 1);
    insert(&head1, 2, 0);
    printf("Polynomial 1: ");
    display(head1);
    insert(&head2, 3, 2);
    insert(&head2, 2, 1);
    insert(&head2, 1, 0);
    printf("Polynomial 2: ");
    display(head2);
    addPolynomials(head1, head2, &head3);
    printf("Result: ");
    display(head3);
    return 0;
}
```



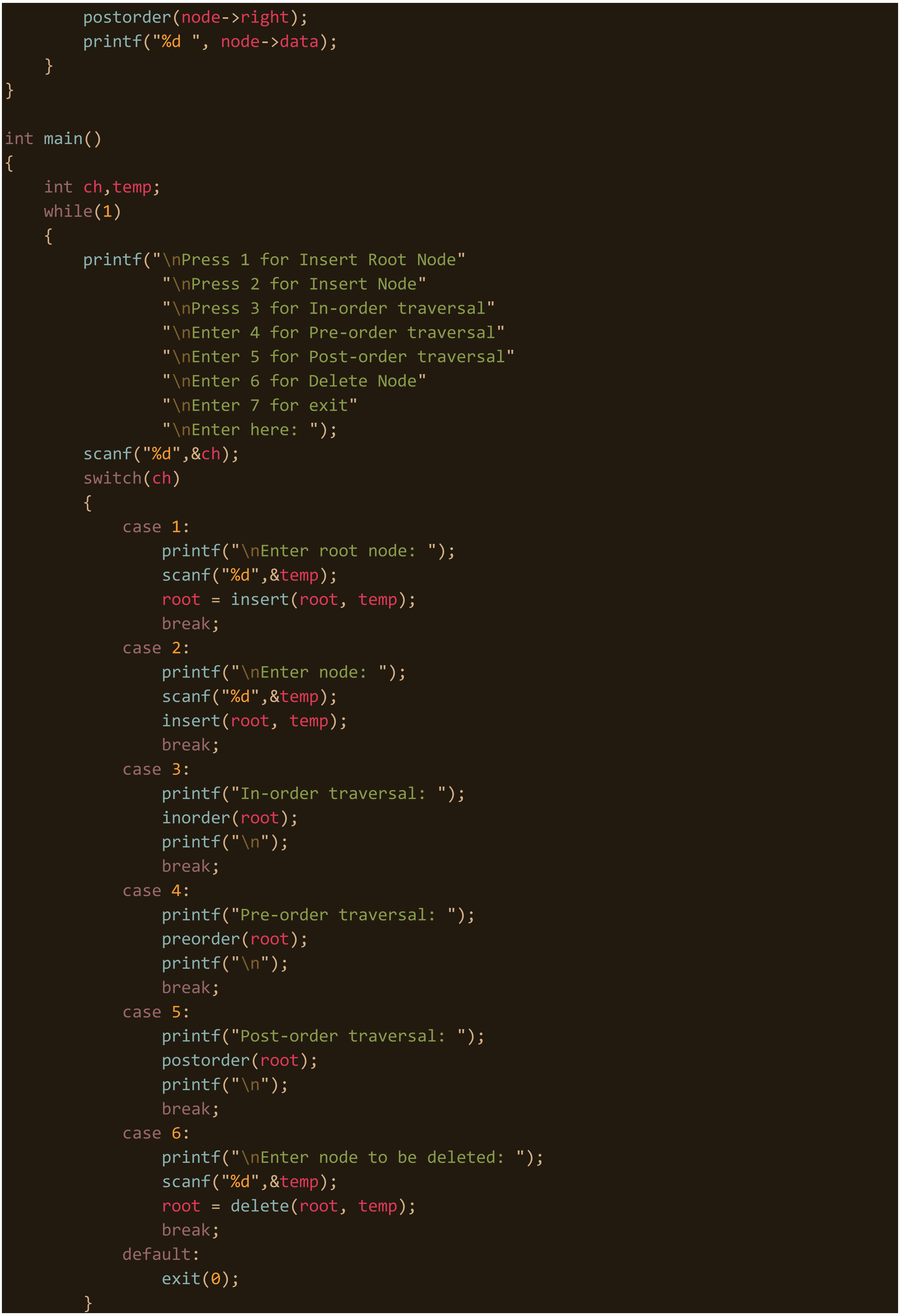
```
//11. Write a program to calculate factorial and to compute the factors of a given number
(i) using recursion, (ii) using iteration.

#include <stdio.h>

int factorial_recursion(int n)
{
    if (n == 0)
    {
        return 1;
    }
    else
    {
        return n * factorial_recursion(n - 1);
    }
}

int factorial_iteration(int n)
{
    int fact = 1;
    for (int i = 1; i <= n; i++)
    {
        fact *= i;
    }
    return fact;
}

void factors_recursion(int n, int i)
{
    if (i > n)
    {
        return;
    }
    if (n % i == 0)
    {
        printf("%d ", i);
    }
    factors_recursion(n, i + 1);
}

void factors_iteration(int n)
{
    printf("Factors using iteration: ");
    for (int i = 1; i <= n; i++)
    {
        if (n % i == 0) {
            printf("%d ", i);
        }
    }
    printf("\n");
}

int main()
{
    int n;
    printf("Enter a number: ");
    scanf("%d", &n);
    printf("Factorial using recursion: %d\n", factorial_recursion(n));
    printf("Factorial using iteration: %d\n", factorial_iteration(n));
    printf("Factor using recursion: ");
    factors_recursion(n, 1);
    printf("\n");
    factors_iteration(n);
    return 0;
}
```

```
//12. Write a program to display Fibonacci series (i) using recursion, (ii) using
iteration.

#include <stdio.h>

int fibonacci_recursion(int n)
{
    if (n == 0 || n == 1)
    {
        return n;
    }
    else
    {
        return fibonacci_recursion(n - 1) + fibonacci_recursion(n - 2);
    }
}

void fibonacci_iteration(int n)
{
    int a = 0, b = 1, c;
    printf("Fibonacci series using iteration: ");
    for (int i = 0; i < n; i++)
    {
        printf("%d ", a);
        c = a + b;
        a = b;
        b = c;
    }
    printf("\n");
}

int main()
{
    int n;
    printf("Enter a number: ");
    scanf("%d", &n);
    printf("Fibonacci series using recursion: ");
    for (int i = 0; i < n; i++)
    {
        printf("%d ", fibonacci_recursion(i));
    }
    printf("\n");
    fibonacci_iteration(n);
    return 0;
}
```

```
//13. Write a program to calculate GCD of two numbers (i) with recursion (ii) without
recursion.

#include <stdio.h>

int gcd_recursion(int a, int b)
{
    if (b == 0)
    {
        return a;
    }
    else
    {
        return gcd_recursion(b, a % b);
    }
}

int gcd_iteration(int a, int b)
{
    while (b != 0)
    {
        int temp = b;
        b = a % b;
        a = temp;
    }
    return a;
}

int main()
{
    int a, b;
    printf("Enter two numbers: ");
    scanf("%d %d", &a, &b);
    printf("GCD using recursion: %d\n", gcd_recursion(a, b));
    printf("GCD using iteration: %d\n", gcd_iteration(a, b));
    return 0;
}
```

```
// 14. Write a program to create a Binary Search Tree and include following operations in
tree: (a) Insertion (b) Deletion (c) Display its pre-order, post-order and in- order
traversals.
```

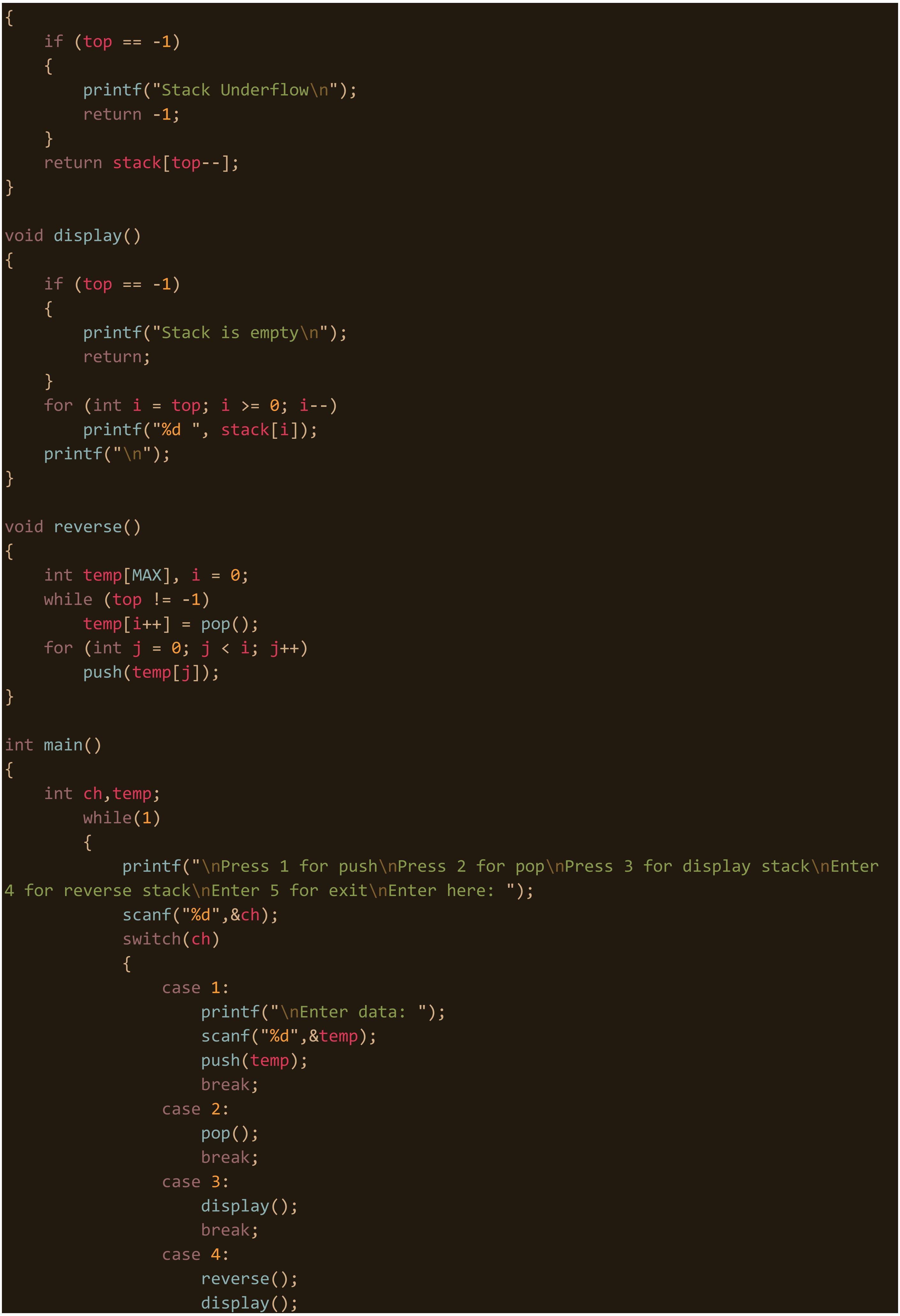
```c
#include <stdio.h>
#include <stdlib.h>

struct Node
{
    int data;
    struct Node *left, *right;
};

struct Node *root = NULL;

struct Node* create_node(int data)
{
    struct Node *node = (struct Node*) malloc(sizeof(struct Node));
    node->data = data;
    node->left = NULL;
    node->right = NULL;
    return node;
}

struct Node* insert(struct Node *node, int data)
{
    if (node == NULL)
    {
        return create_node(data);
    }
    if (data < node->data)
    {
        node->left = insert(node->left, data);
    }
    else if (data > node->data)
    {
        node->right = insert(node->right, data);
    }
    return node;
}

struct Node* find_min(struct Node *node)
{
    while (node->left != NULL)
    {
        node = node->left;
    }
    return node;
}

struct Node* delete(struct Node *node, int data)
{
    if (node == NULL)
    {
        return node;
    }
    if (data < node->data)
    {

        node->left = delete(node->left, data);
    }
    else if (data > node->data)
    {
        node->right = delete(node->right, data);
    }
    else
    {
        if (node->left == NULL)
        {
            struct Node *temp = node->right;
            free(node);
            return temp;
        }
        else if (node->right == NULL)
        {
            struct Node *temp = node->left;
            free(node);
            return temp;
        }
        struct Node *temp = find_min(node->right);
        node->data = temp->data;
        node->right = delete(node->right, temp->data);
    }
    return node;
}

void preorder(struct Node *node)
{
    if (node != NULL)
    {
        printf("%d ", node->data);
        preorder(node->left);
        preorder(node->right);
    }
}

void inorder(struct Node *node)
{
    if (node != NULL)
    {
        inorder(node->left);
        printf("%d ", node->data);
        inorder(node->right);
    }
}

void postorder(struct Node *node)
{
    if (node != NULL)
    {
        postorder(node->left);
```



```c
        postorder(node->right);
        printf("%d ", node->data);
    }
}

int main()
{
    int ch,temp;
    while(1)
    {
        printf("\nPress 1 for Insert Root Node"
                "\nPress 2 for Insert Node"
                "\nPress 3 for In-order traversal"
                "\nEnter 4 for Pre-order traversal"
                "\nEnter 5 for Post-order traversal"
                "\nEnter 6 for Delete Node"
                "\nEnter 7 for exit"
                "\nEnter here: ");
        scanf("%d",&ch);
        switch(ch)
        {
            case 1:
                printf("\nEnter root node: ");
                scanf("%d",&temp);
                root = insert(root, temp);
                break;
            case 2:
                printf("\nEnter node: ");
                scanf("%d",&temp);
                insert(root, temp);
                break;
            case 3:
                printf("In-order traversal: ");
                inorder(root);
                printf("\n");
                break;
            case 4:
                printf("Pre-order traversal: ");
                preorder(root);
                printf("\n");
                break;
            case 5:
                printf("Post-order traversal: ");
                postorder(root);
                printf("\n");
                break;
            case 6:
                printf("\nEnter node to be deleted: ");
                scanf("%d",&temp);
                root = delete(root, temp);
                break;
            default:
                exit(0);
        }
    }
    return 0;
}
```

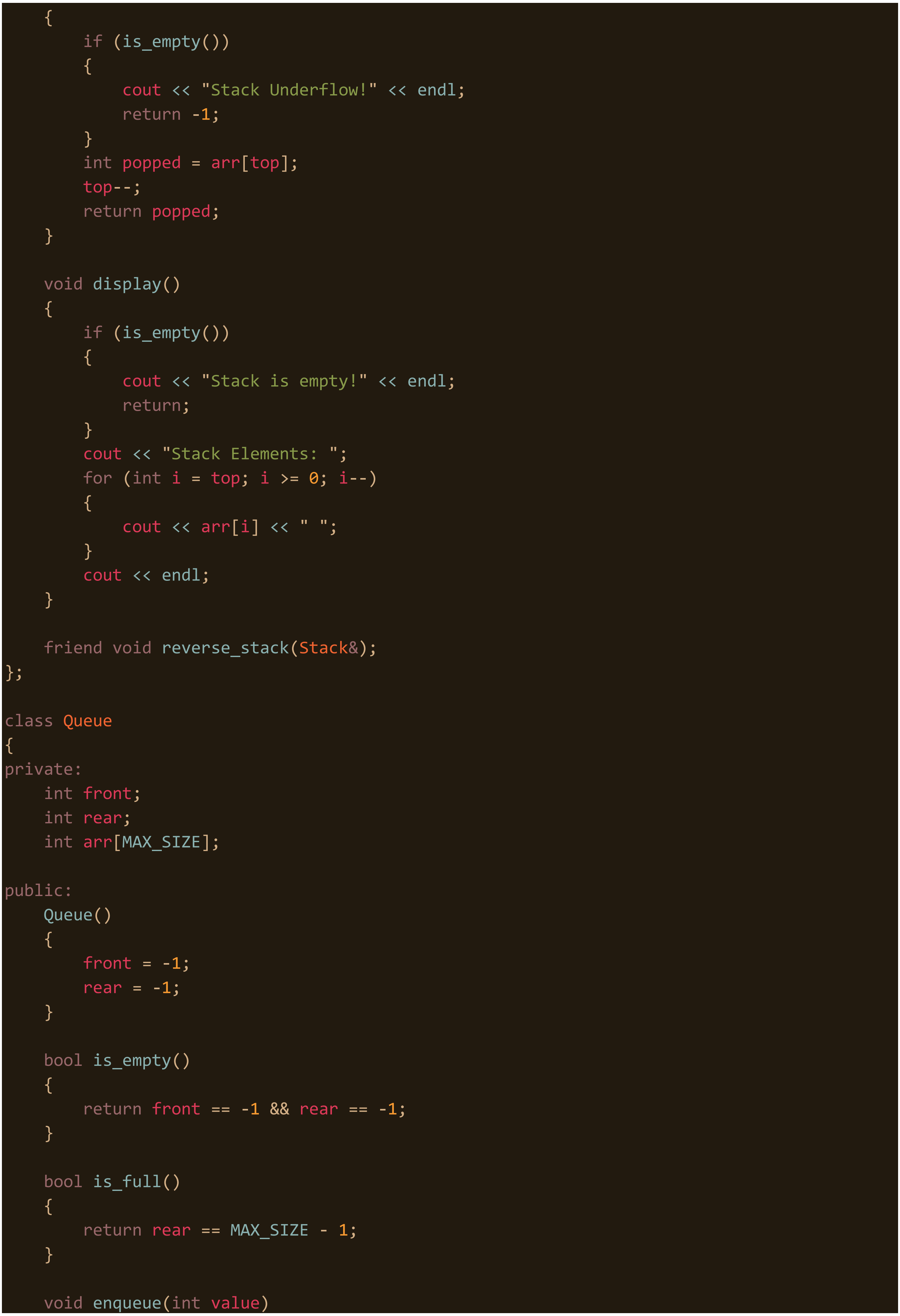
```
//15. Write a program to convert the Sparse Matrix into non-zero form and vice-versa.

#include <iostream>
using namespace std;

int main()
{
    int m, n, num_non_zero;

    // Reading the size of the matrix and number of non-zero elements
    cout << "Enter the number of rows of the matrix: ";
    cin >> m;
    cout << "Enter the number of columns of the matrix: ";
    cin >> n;
    cout << "Enter the number of non-zero elements: ";
    cin >> num_non_zero;

    int sparse_matrix[num_non_zero][3];

    // Reading the non-zero elements of the sparse matrix
    cout << "Enter the non-zero elements of the matrix: " << endl;
    for (int i = 0; i < num_non_zero; i++)
    {
        cout << "Enter the row index: ";
        cin >> sparse_matrix[i][0];
        cout << "Enter the column index: ";
        cin >> sparse_matrix[i][1];
        cout << "Enter the value: ";
        cin >> sparse_matrix[i][2];
    }

    // Converting the sparse matrix to non-zero form
    int non_zero_matrix[m][n] = {};
    for (int i = 0; i < num_non_zero; i++)
    {
        non_zero_matrix[sparse_matrix[i][0]][sparse_matrix[i][1]] = sparse_matrix[i][2];
    }

    // Displaying the non-zero matrix
    cout << "The non-zero matrix is: " << endl;
    for (int i = 0; i < m; i++)
    {
        for (int j = 0; j < n; j++)
        {
            cout << non_zero_matrix[i][j] << "\t";
        }
        cout << endl;
    }

    // Converting the non-zero matrix to sparse form
    num_non_zero = 0;
    for (int i = 0; i < m; i++)
    {
        for (int j = 0; j < n; j++)
        {
            if (non_zero_matrix[i][j] != 0)
            {
                sparse_matrix[num_non_zero][0] = i;
                sparse_matrix[num_non_zero][1] = j;
                sparse_matrix[num_non_zero][2] = non_zero_matrix[i][j];
                num_non_zero++;
            }
        }
    }

    // Displaying the sparse matrix
    cout << "The sparse matrix is: " << endl;
    for (int i = 0; i < num_non_zero; i++)
    {
        cout << sparse_matrix[i][0] << "\t" << sparse_matrix[i][1] << "\t" <<
sparse_matrix[i][2] << endl;
    }

    return 0;
}
```

```
//16. Write a program to reverse the order of the elements in the stack using additional
stack.

#include <stdio.h>
#include <stdlib.h>
#define MAX 10

int stack[MAX];
int top = -1;

void push(int data)
{
    if (top == MAX - 1)
    {
        printf("Stack Overflow\n");
        return;
    }
    stack[++top] = data;
}

int pop()
```

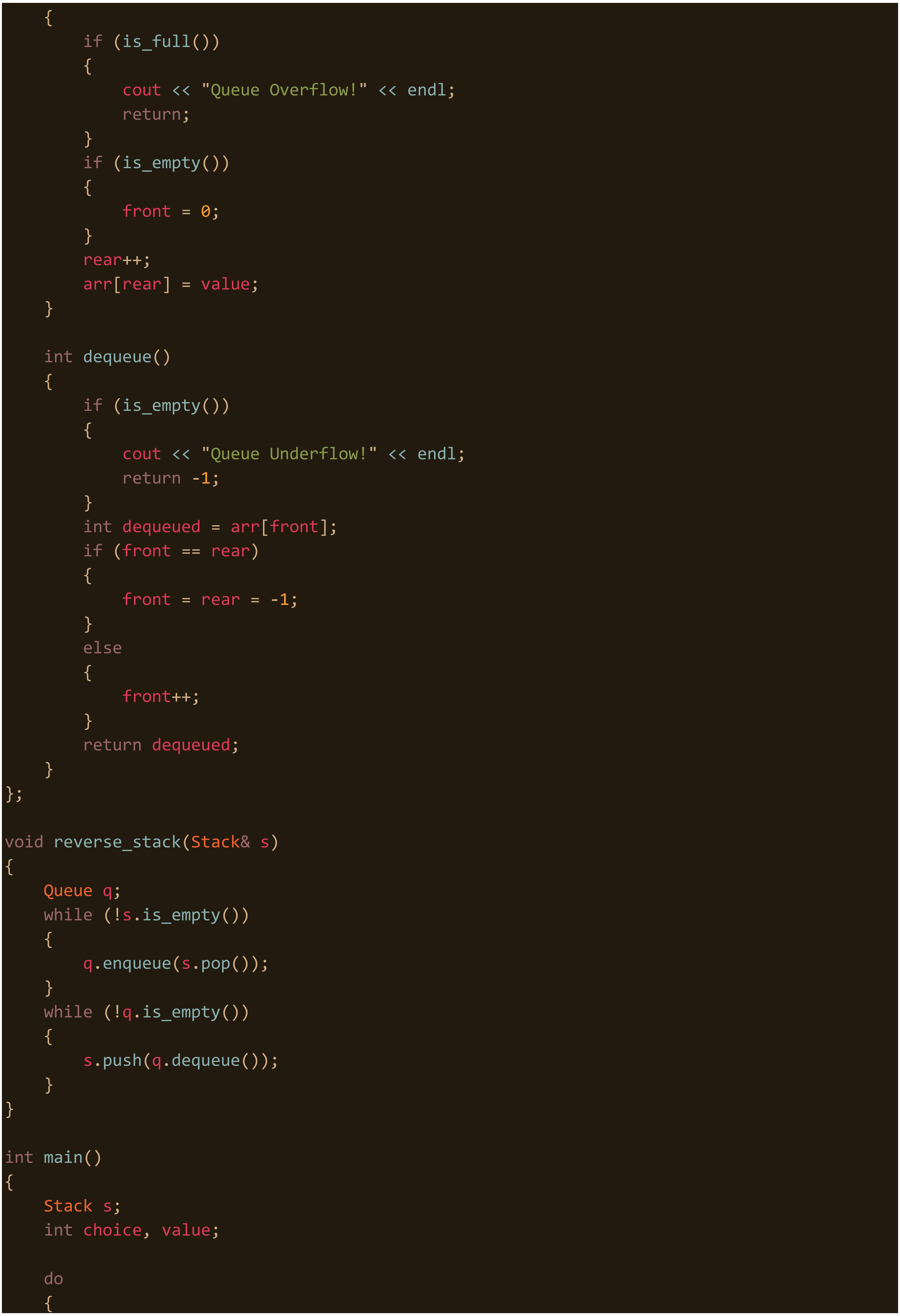


```
{
    if (top == -1)
    {
        printf("Stack Underflow\n");
        return -1;
    }
    return stack[top--];
}

void display()
{
    if (top == -1)
    {
        printf("Stack is empty\n");
        return;
    }
    for (int i = top; i >= 0; i--)
        printf("%d ", stack[i]);
    printf("\n");
}

void reverse()
{
    int temp[MAX], i = 0;
    while (top != -1)
        temp[i++] = pop();
    for (int j = 0; j < i; j++)
        push(temp[j]);
}

int main()
{
    int ch,temp;
        while(1)
        {
            printf("\nPress 1 for push\nPress 2 for pop\nPress 3 for display stack\nEnter 
4 for reverse stack\nEnter 5 for exit\nEnter here: ");
            scanf("%d",&ch);
            switch(ch)
            {
                case 1:
                    printf("\nEnter data: ");
                    scanf("%d",&temp);
                    push(temp);
                    break;
                case 2:
                    pop();
                    break;
                case 3:
                    display();
                    break;
                case 4:
                    reverse();
                    display();
                    break;
                default:
                    exit(0);
            }
        }
    return 0;
}
```

```
//17. Write a program to reverse the order of the elements in the stack using additional
Queue.

#include <iostream>
using namespace std;

#define MAX_SIZE 100

class Stack
{
private:
    int top;
    int arr[MAX_SIZE];

public:
    Stack()
    {
        top = -1;
    }

    bool is_empty()
    {
        return top == -1;
    }

    bool is_full()
    {
        return top == MAX_SIZE - 1;
    }

    void push(int value)
    {
        if (is_full())
        {
            cout << "Stack Overflow!" << endl;
            return;
        }
        top++;
        arr[top] = value;
    }

    int pop()
```



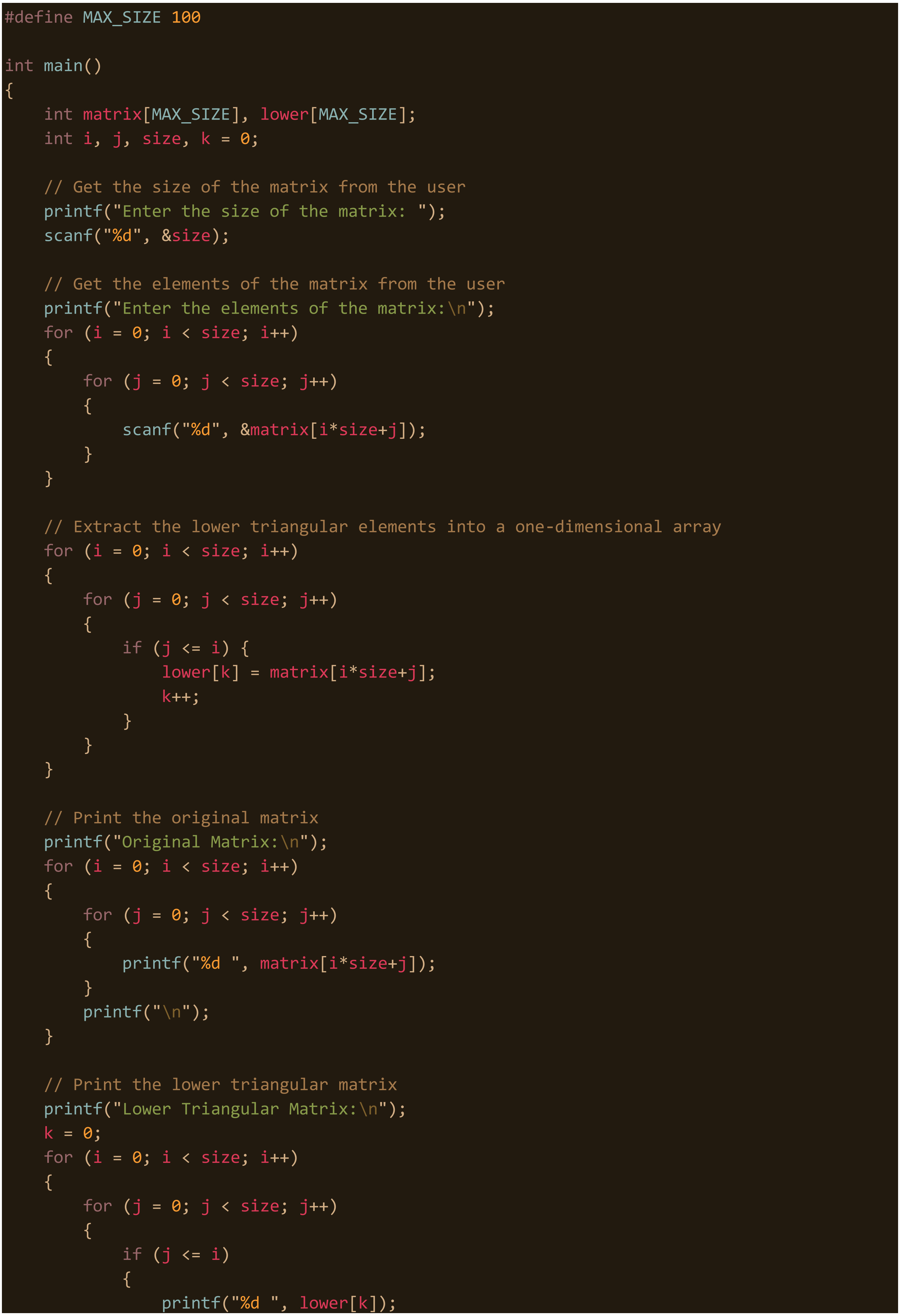
```cpp
    {
        if (is_empty())
        {
            cout << "Stack Underflow!" << endl;
            return -1;
        }
        int popped = arr[top];
        top--;
        return popped;
    }

    void display()
    {
        if (is_empty())
        {
            cout << "Stack is empty!" << endl;
            return;
        }
        cout << "Stack Elements: ";
        for (int i = top; i >= 0; i--)
        {
            cout << arr[i] << " ";
        }
        cout << endl;
    }

    friend void reverse_stack(Stack&);
};

class Queue
{
private:
    int front;
    int rear;
    int arr[MAX_SIZE];

public:
    Queue()
    {
        front = -1;
        rear = -1;
    }

    bool is_empty()
    {
        return front == -1 && rear == -1;
    }

    bool is_full()
    {
        return rear == MAX_SIZE - 1;
    }

    void enqueue(int value)
```



```
    {
        if (is_full())
        {
            cout << "Queue Overflow!" << endl;
            return;
        }
        if (is_empty())
        {
            front = 0;
        }
        rear++;
        arr[rear] = value;
    }

    int dequeue()
    {
        if (is_empty())
        {
            cout << "Queue Underflow!" << endl;
            return -1;
        }
        int dequeued = arr[front];
        if (front == rear)
        {
            front = rear = -1;
        }
        else
        {
            front++;
        }
        return dequeued;
    }
};

void reverse_stack(Stack& s)
{
    Queue q;
    while (!s.is_empty())
    {
        q.enqueue(s.pop());
    }
    while (!q.is_empty())
    {
        s.push(q.dequeue());
    }
}

int main()
{
    Stack s;
    int choice, value;

    do
    {
        cout << "\n1. Push Element" << endl;
        cout << "2. Pop Element" << endl;
        cout << "3. Display Stack" << endl;
        cout << "4. Reverse Stack" << endl;
        cout << "5. Exit" << endl;
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice)
        {
            case 1:
                cout << "Enter the value to push: ";
                cin >> value;
                s.push(value);
                break;
            case 2:
                cout << "Popped Element: " << s.pop() << endl;
                break;
            case 3:
                s.display();
                break;
            case 4:
                reverse_stack(s);
                cout << "Stack Reversed Successfully!" << endl;
                break;
            case 5:
                cout << "Exiting Program..." << endl;
                break;
            default:
                cout << "Invalid Choice!" << endl;
        }
        cout << endl;
    } while (choice != 5);

    return 0;
}
```

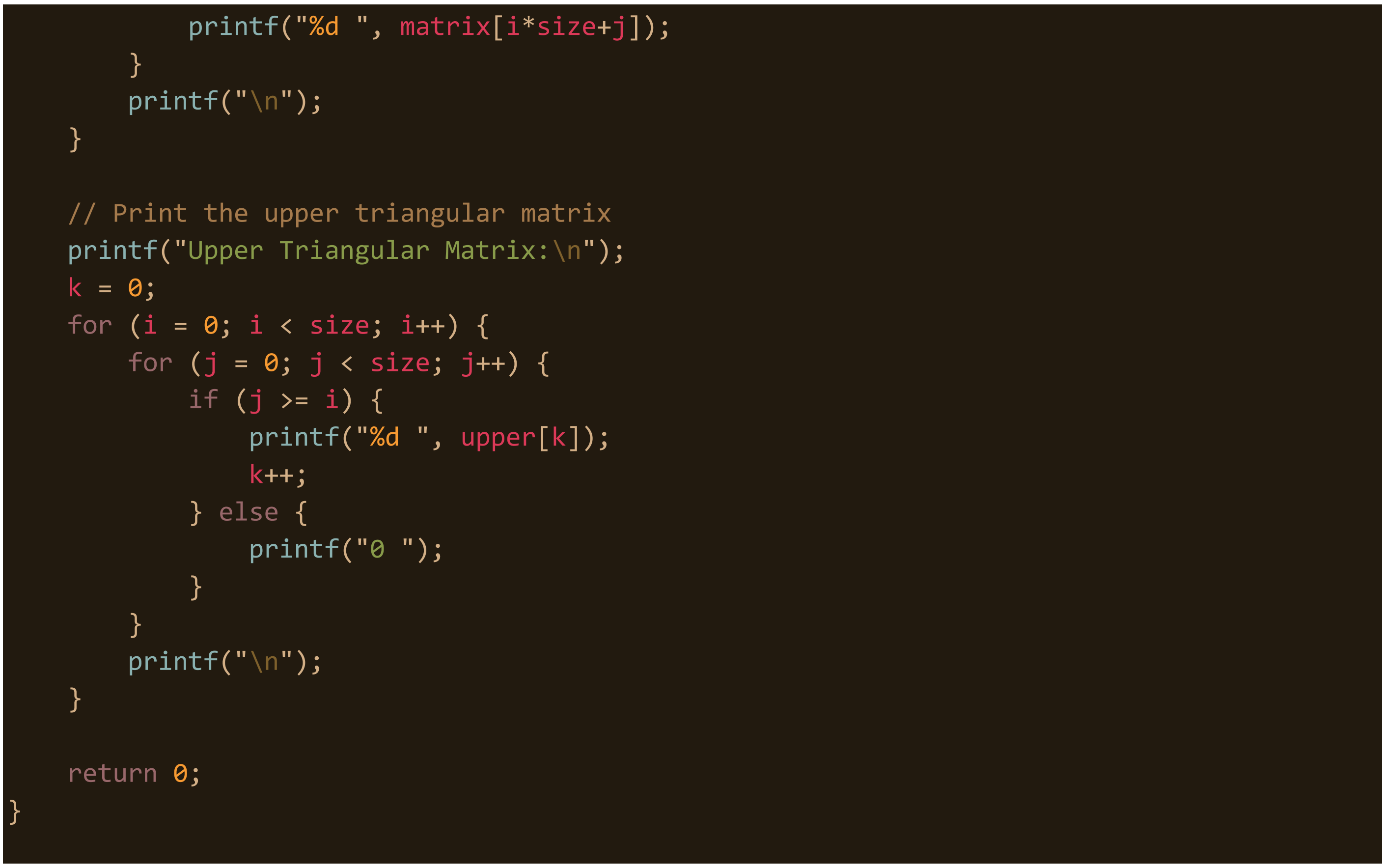
```
//18. Write a program to implement Diagonal Matrix using one-dimensional array.

#include <stdio.h>

#define MAX_SIZE 100

int main()
{
    int matrix[MAX_SIZE][MAX_SIZE], diagonal[MAX_SIZE];
    int i, j, size;

    // Get the size of the matrix from the user
    printf("Enter the size of the matrix: ");
    scanf("%d", &size);

    // Get the elements of the matrix from the user
    printf("Enter the elements of the matrix:\n");
    for (i = 0; i < size; i++)
    {
        for (j = 0; j < size; j++)
        {
            scanf("%d", &matrix[i][j]);
        }
    }

    // Extract the diagonal elements into a one-dimensional array
    for (i = 0; i < size; i++)
    {
        diagonal[i] = matrix[i][i];
    }

    // Print the original matrix
    printf("Original Matrix:\n");
    for (i = 0; i < size; i++)
    {
        for (j = 0; j < size; j++)
        {
            printf("%d ", matrix[i][j]);
        }
        printf("\n");
    }

    // Print the diagonal matrix
    printf("Diagonal Matrix:\n");
    for (i = 0; i < size; i++) {
        for (j = 0; j < size; j++) {
            if (i == j) {
                printf("%d ", diagonal[i]);
            } else {
                printf("0 ");
            }
        }
        printf("\n");
    }

    return 0;
}
```

```
//19. Write a program to implement Lower Triangular Matrix using the one-dimensional
array.

#include <stdio.h>
```

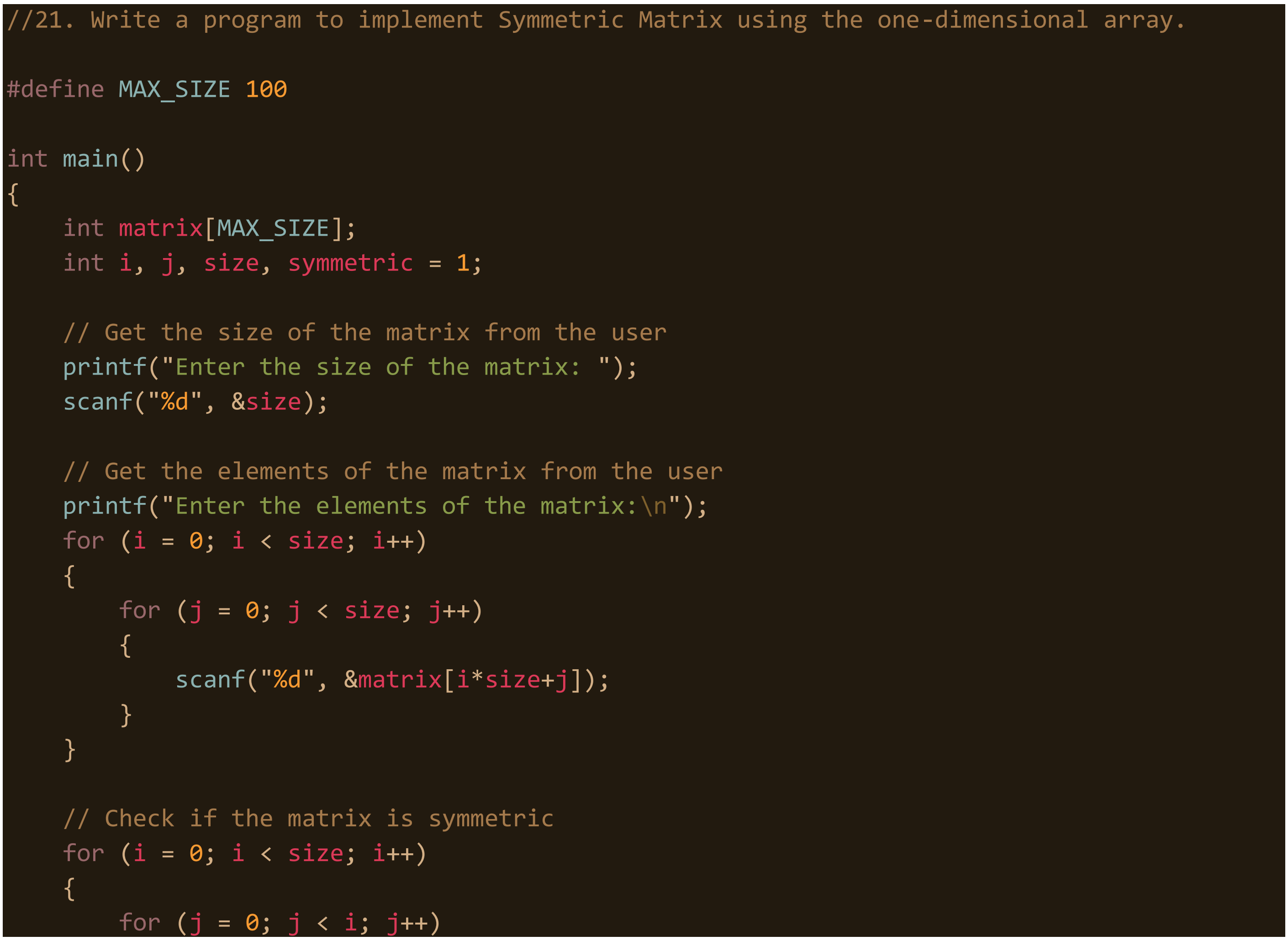


```c
#define MAX_SIZE 100

int main()
{
    int matrix[MAX_SIZE], lower[MAX_SIZE];
    int i, j, size, k = 0;

    // Get the size of the matrix from the user
    printf("Enter the size of the matrix: ");
    scanf("%d", &size);

    // Get the elements of the matrix from the user
    printf("Enter the elements of the matrix:\n");
    for (i = 0; i < size; i++)
    {
        for (j = 0; j < size; j++)
        {
            scanf("%d", &matrix[i*size+j]);
        }
    }

    // Extract the lower triangular elements into a one-dimensional array
    for (i = 0; i < size; i++)
    {
        for (j = 0; j < size; j++)
        {
            if (j <= i) {
                lower[k] = matrix[i*size+j];
                k++;
            }
        }
    }

    // Print the original matrix
    printf("Original Matrix:\n");
    for (i = 0; i < size; i++)
    {
        for (j = 0; j < size; j++)
        {
            printf("%d ", matrix[i*size+j]);
        }
        printf("\n");
    }

    // Print the lower triangular matrix
    printf("Lower Triangular Matrix:\n");
    k = 0;
    for (i = 0; i < size; i++)
    {
        for (j = 0; j < size; j++)
        {
            if (j <= i)
            {
                printf("%d ", lower[k]);
                k++;
            }
            else
            {
                printf("0 ");
            }
        }
        printf("\n");
    }

    return 0;
}
```

```
//20. Write a program to implement Upper Triangular Matrix using the one-dimensional
array.

#include <stdio.h>

#define MAX_SIZE 100

int main() {
    int matrix[MAX_SIZE], upper[MAX_SIZE];
    int i, j, size, k = 0;

    // Get the size of the matrix from the user
    printf("Enter the size of the matrix: ");
    scanf("%d", &size);

    // Get the elements of the matrix from the user
    printf("Enter the elements of the matrix:\n");
    for (i = 0; i < size; i++) {
        for (j = 0; j < size; j++) {
            scanf("%d", &matrix[i*size+j]);
        }
    }

    // Extract the upper triangular elements into a one-dimensional array
    for (i = 0; i < size; i++) {
        for (j = 0; j < size; j++) {
            if (j >= i) {
                upper[k] = matrix[i*size+j];
                k++;
            }
        }
    }

    // Print the original matrix
    printf("Original Matrix:\n");
    for (i = 0; i < size; i++) {
        for (j = 0; j < size; j++) {
```

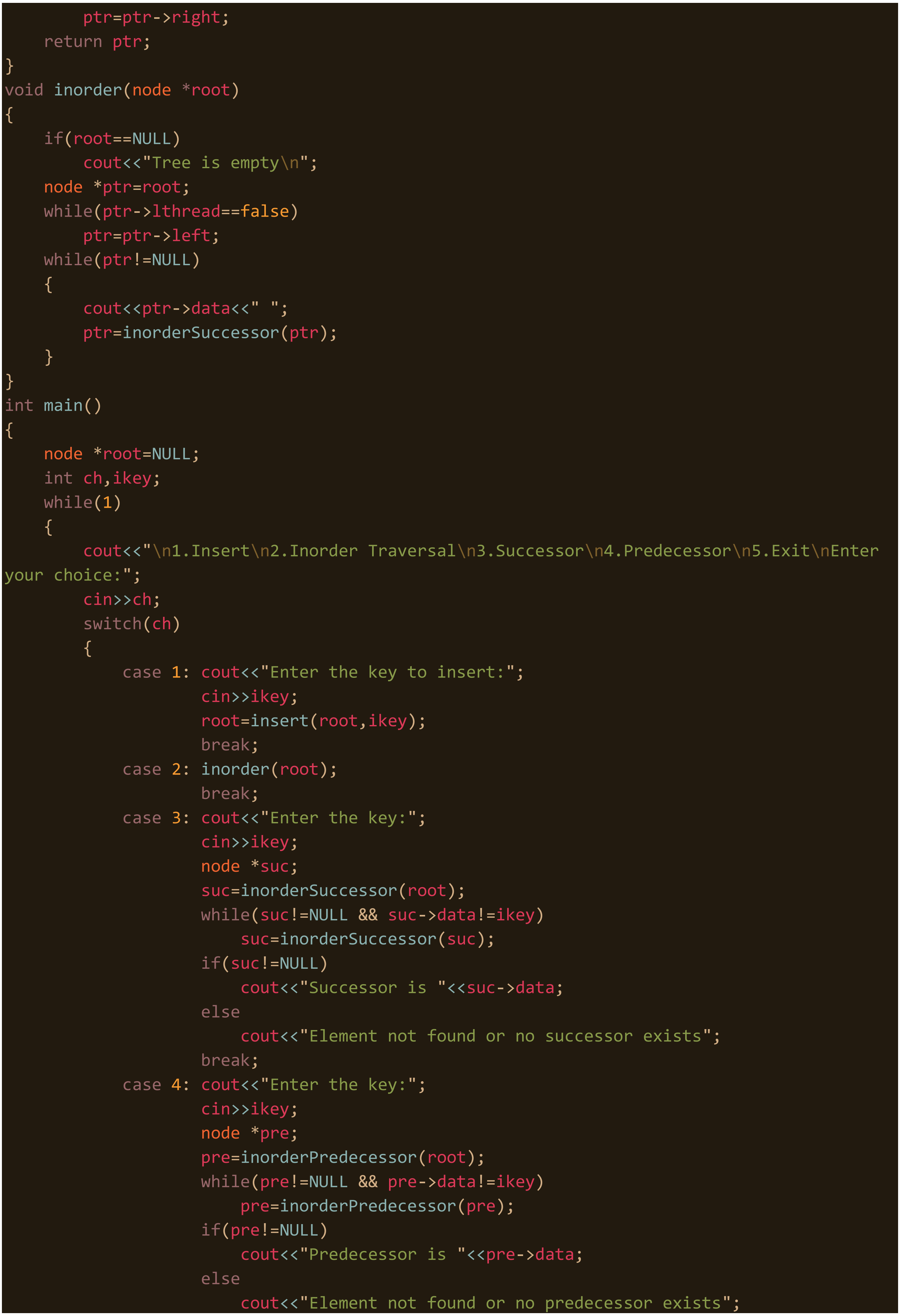


```
            printf("%d ", matrix[i*size+j]);
        }
        printf("\n");
    }

    // Print the upper triangular matrix
    printf("Upper Triangular Matrix:\n");
    k = 0;
    for (i = 0; i < size; i++) {
        for (j = 0; j < size; j++) {
            if (j >= i) {
                printf("%d ", upper[k]);
                k++;
            } else {
                printf("0 ");
            }
        }
        printf("\n");
    }

    return 0;
}
```



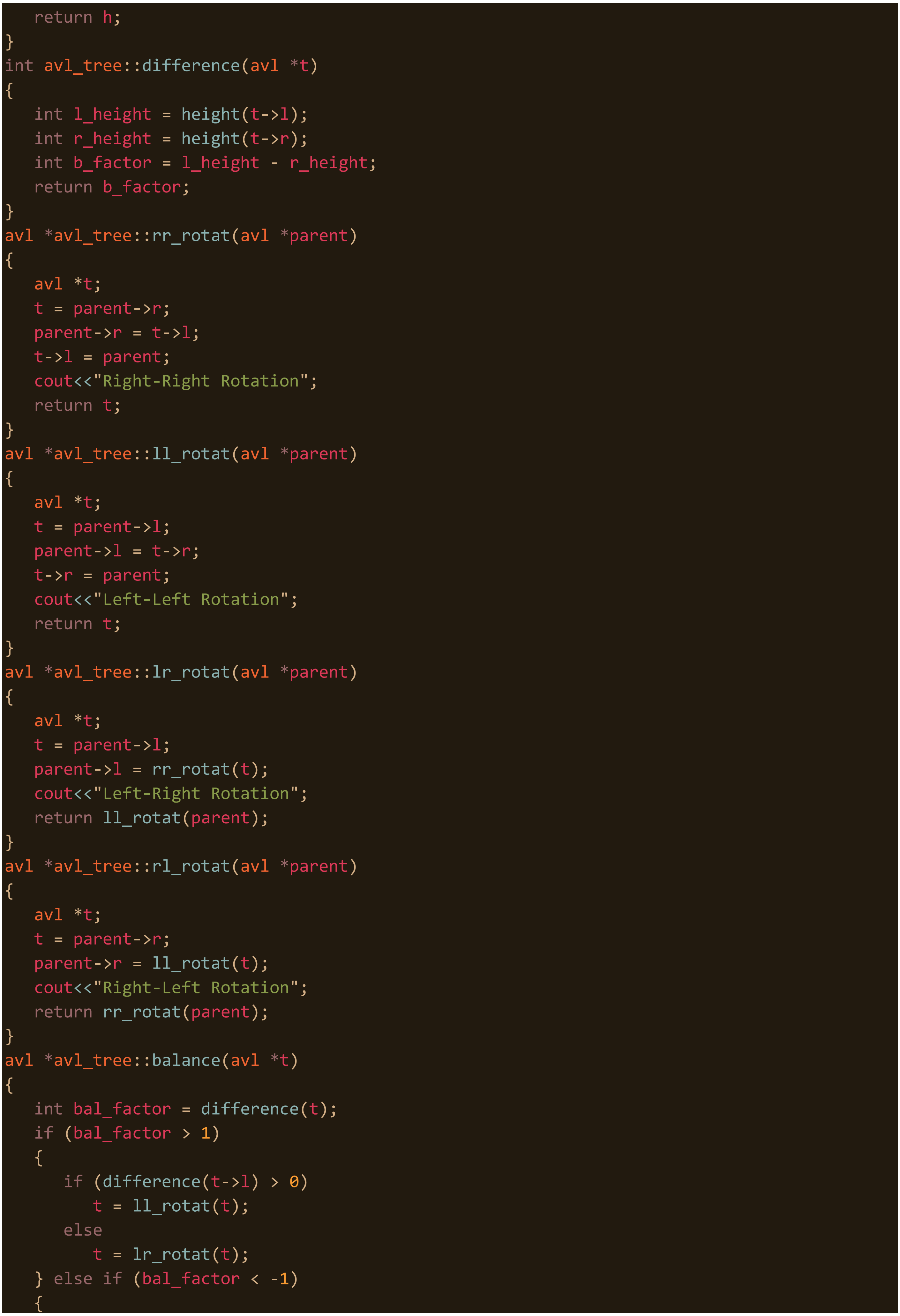
```
//21. Write a program to implement Symmetric Matrix using the one-dimensional array.

#define MAX_SIZE 100

int main()
{
    int matrix[MAX_SIZE];
    int i, j, size, symmetric = 1;

    // Get the size of the matrix from the user
    printf("Enter the size of the matrix: ");
    scanf("%d", &size);

    // Get the elements of the matrix from the user
    printf("Enter the elements of the matrix:\n");
    for (i = 0; i < size; i++)
    {
        for (j = 0; j < size; j++)
        {
            scanf("%d", &matrix[i*size+j]);
        }
    }

    // Check if the matrix is symmetric
    for (i = 0; i < size; i++)
    {
        for (j = 0; j < i; j++)
        {
            if (matrix[i*size+j] != matrix[j*size+i]) {
                symmetric = 0;
                break;
            }
        }
        if (symmetric == 0)
        {
            break;
        }
    }

    // Print the result
    if (symmetric == 1)
    {
        printf("The matrix is symmetric.\n");
    }
    else
    {
        printf("The matrix is not symmetric.\n");
    }

    return 0;
}
```

```
//22. Write a program to create a Threaded Binary Tree as per in-order traversal and
implement operations like finding the successor and predecessor of an element, inserting
an element, and in-order traversal.

#include<iostream>
using namespace std;
struct node
{
    int data;
    node *left,*right;
    bool lthread,rthread;
};
node *insert(node *root,int ikey)
{
    node *ptr=root,*par=NULL;
    while(ptr!=NULL)
    {
        if(ikey==ptr->data)
        {
            cout<<"Duplicate Key\n";
            return root;
        }
        par=ptr;
        if(ikey<ptr->data)
        {
            if(ptr->lthread==false)
                ptr=ptr->left;
            else
                break;
        }
        else
        {
            if(ptr->rthread==false)
                ptr=ptr->right;
            else
                break;
        }
    }
    node *tmp=new node;
    tmp->data=ikey;
    tmp->lthread=true;
    tmp->rthread=true;
    if(par==NULL)
    {
        root=tmp;
        tmp->left=NULL;
        tmp->right=NULL;
    }
    else if(ikey<par->data)
    {
        tmp->left=par->left;
        tmp->right=par;
        par->lthread=false;
        par->left=tmp;
    }
    else
    {
        tmp->left=par;
        tmp->right=par->right;
        par->rthread=false;
        par->right=tmp;
    }
    return root;
}
node *inorderSuccessor(node *ptr)
{
    if(ptr->rthread==true)
        return ptr->right;
    ptr=ptr->right;
    while(ptr->lthread==false)
        ptr=ptr->left;
    return ptr;
}
node *inorderPredecessor(node *ptr)
{
    if(ptr->lthread==true)
        return ptr->left;
    ptr=ptr->left;
    while(ptr->rthread==false)
```

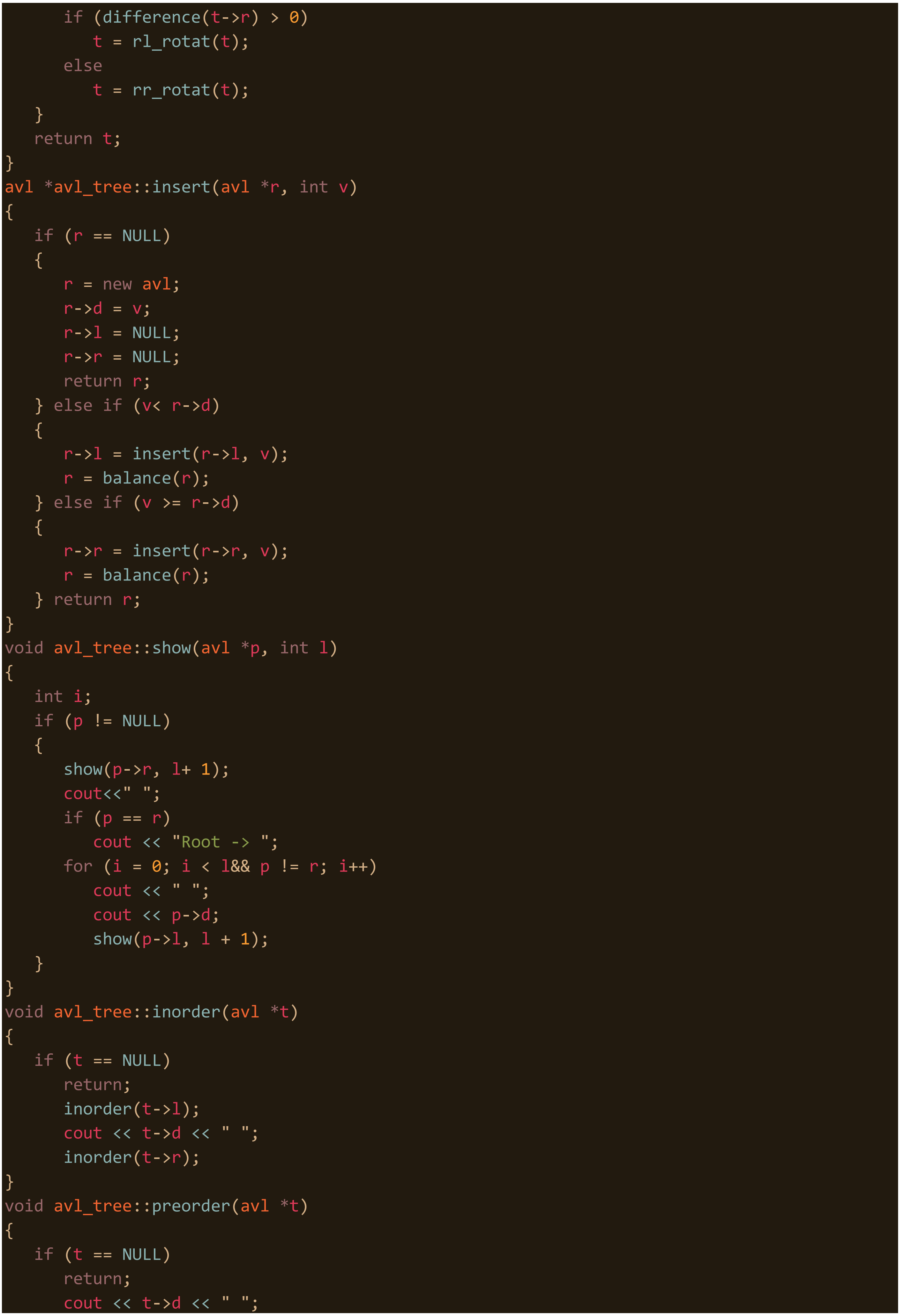


```
        ptr=ptr->right;
    return ptr;
}
void inorder(node *root)
{
    if(root==NULL)
        cout<<"Tree is empty\n";
    node *ptr=root;
    while(ptr->lthread==false)
        ptr=ptr->left;
    while(ptr!=NULL)
    {
        cout<<ptr->data<<" ";
        ptr=inorderSuccessor(ptr);
    }
}
int main()
{
    node *root=NULL;
    int ch,ikey;
    while(1)
    {
        cout<<"\n1.Insert\n2.Inorder Traversal\n3.Successor\n4.Predecessor\n5.Exit\nEnter
your choice:";
        cin>>ch;
        switch(ch)
        {
            case 1: cout<<"Enter the key to insert:";
                    cin>>ikey;
                    root=insert(root,ikey);
                    break;
            case 2: inorder(root);
                    break;
            case 3: cout<<"Enter the key:";
                    cin>>ikey;
                    node *suc;
                    suc=inorderSuccessor(root);
                    while(suc!=NULL && suc->data!=ikey)
                        suc=inorderSuccessor(suc);
                    if(suc!=NULL)
                        cout<<"Successor is "<<suc->data;
                    else
                        cout<<"Element not found or no successor exists";
                    break;
            case 4: cout<<"Enter the key:";
                    cin>>ikey;
                    node *pre;
                    pre=inorderPredecessor(root);
                    while(pre!=NULL && pre->data!=ikey)
                        pre=inorderPredecessor(pre);
                    if(pre!=NULL)
                        cout<<"Predecessor is "<<pre->data;
                    else
                        cout<<"Element not found or no predecessor exists";
                    break;
            case 5: exit(0);
        }
    }
    return 0;
}
```

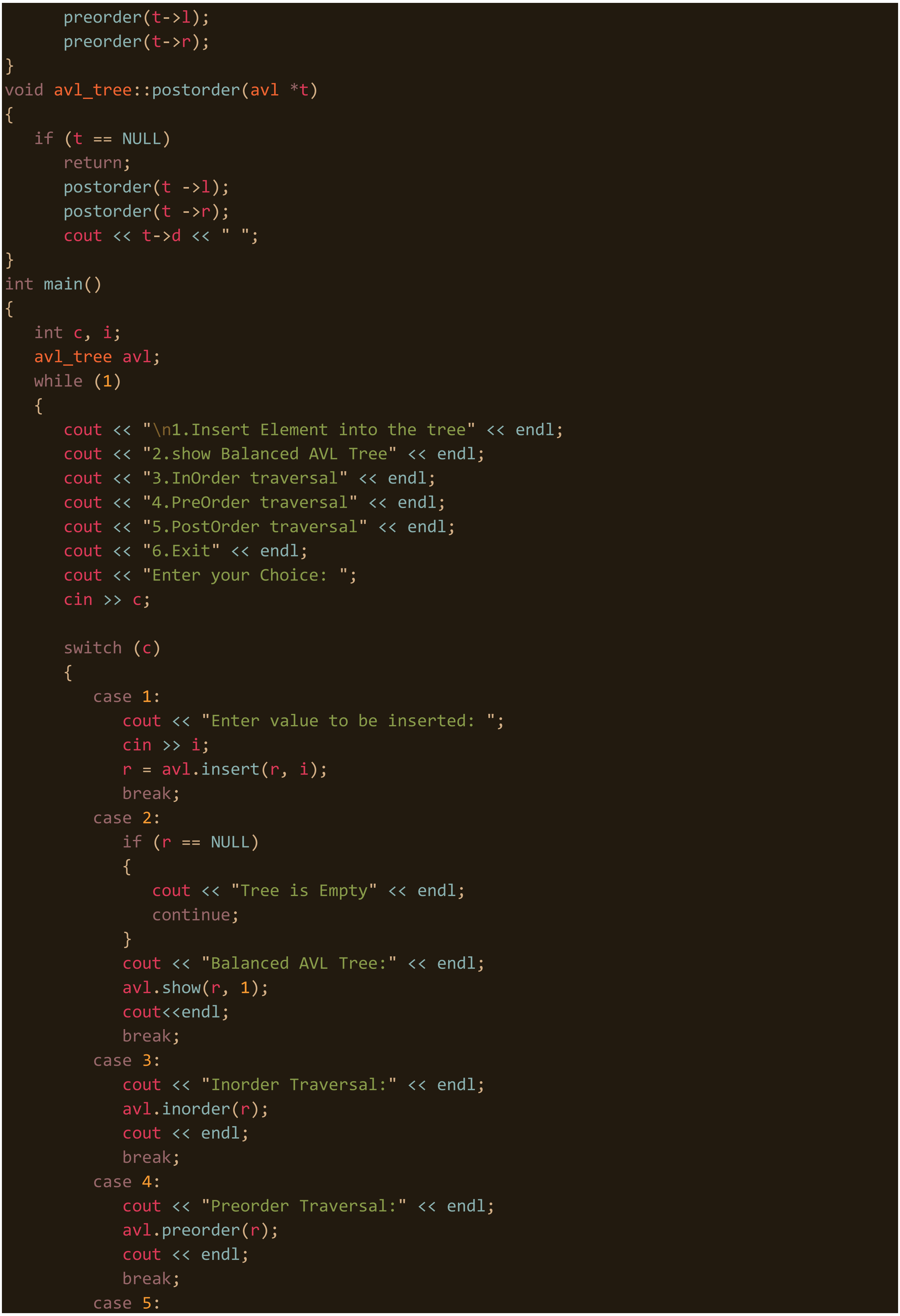
```
//23. Write a program to implement various operations on AVL Tree.

#include<iostream>
#include<cstdio>
#include<sstream>
#include<algorithm>
#define pow2(n) (1 << (n))
using namespace std;
struct avl
{
   int d;
   struct avl *l;
   struct avl *r;
}*r;
class avl_tree
{
   public:
      int height(avl *);
      int difference(avl *);
      avl *rr_rotat(avl *);
      avl *ll_rotat(avl *);
      avl *lr_rotat(avl*);
      avl *rl_rotat(avl *);
      avl * balance(avl *);
      avl * insert(avl*, int);
      void show(avl*, int);
      void inorder(avl *);
      void preorder(avl *);
      void postorder(avl*);
      avl_tree()
      {
         r = NULL;
      }
};
int avl_tree::height(avl *t)
{
   int h = 0;
   if (t != NULL)
   {
      int l_height = height(t->l);
      int r_height = height(t->r);
      int max_height = max(l_height, r_height);
      h = max_height + 1;
   }
```



```
   return h;
}
int avl_tree::difference(avl *t)
{
   int l_height = height(t->l);
   int r_height = height(t->r);
   int b_factor = l_height - r_height;
   return b_factor;
}
avl *avl_tree::rr_rotat(avl *parent)
{
   avl *t;
   t = parent->r;
   parent->r = t->l;
   t->l = parent;
   cout<<"Right-Right Rotation";
   return t;
}
avl *avl_tree::ll_rotat(avl *parent)
{
   avl *t;
   t = parent->l;
   parent->l = t->r;
   t->r = parent;
   cout<<"Left-Left Rotation";
   return t;
}
avl *avl_tree::lr_rotat(avl *parent)
{
   avl *t;
   t = parent->l;
   parent->l = rr_rotat(t);
   cout<<"Left-Right Rotation";
   return ll_rotat(parent);
}
avl *avl_tree::rl_rotat(avl *parent)
{
   avl *t;
   t = parent->r;
   parent->r = ll_rotat(t);
   cout<<"Right-Left Rotation";
   return rr_rotat(parent);
}
avl *avl_tree::balance(avl *t)
{
   int bal_factor = difference(t);
   if (bal_factor > 1)
   {
      if (difference(t->l) > 0)
         t = ll_rotat(t);
      else
         t = lr_rotat(t);
   } else if (bal_factor < -1)
   {
```



```
      if (difference(t->r) > 0)
         t = rl_rotat(t);
      else
         t = rr_rotat(t);
   }
   return t;
}
avl *avl_tree::insert(avl *r, int v)
{
   if (r == NULL)
   {
      r = new avl;
      r->d = v;
      r->l = NULL;
      r->r = NULL;
      return r;
   } else if (v< r->d)
   {
      r->l = insert(r->l, v);
      r = balance(r);
   } else if (v >= r->d)
   {
      r->r = insert(r->r, v);
      r = balance(r);
   } return r;
}
void avl_tree::show(avl *p, int l)
{
   int i;
   if (p != NULL)
   {
      show(p->r, l+ 1);
      cout<<" ";
      if (p == r)
         cout << "Root -> ";
      for (i = 0; i < l&& p != r; i++)
         cout << " ";
         cout << p->d;
         show(p->l, l + 1);
   }
}
void avl_tree::inorder(avl *t)
{
   if (t == NULL)
      return;
      inorder(t->l);
      cout << t->d << " ";
      inorder(t->r);
}
void avl_tree::preorder(avl *t)
{
   if (t == NULL)
      return;
      cout << t->d << " ";
```



```
      preorder(t->l);
      preorder(t->r);
}
void avl_tree::postorder(avl *t)
{
   if (t == NULL)
      return;
      postorder(t ->l);
      postorder(t ->r);
      cout << t->d << " ";
}
int main()
{
   int c, i;
   avl_tree avl;
   while (1)
   {
      cout << "\n1.Insert Element into the tree" << endl;
      cout << "2.show Balanced AVL Tree" << endl;
      cout << "3.InOrder traversal" << endl;
      cout << "4.PreOrder traversal" << endl;
      cout << "5.PostOrder traversal" << endl;
      cout << "6.Exit" << endl;
      cout << "Enter your Choice: ";
      cin >> c;

      switch (c)
      {
         case 1:
            cout << "Enter value to be inserted: ";
            cin >> i;
            r = avl.insert(r, i);
            break;
         case 2:
            if (r == NULL)
            {
               cout << "Tree is Empty" << endl;
               continue;
            }
            cout << "Balanced AVL Tree:" << endl;
            avl.show(r, 1);
            cout<<endl;
            break;
         case 3:
            cout << "Inorder Traversal:" << endl;
            avl.inorder(r);
            cout << endl;
            break;
         case 4:
            cout << "Preorder Traversal:" << endl;
            avl.preorder(r);
            cout << endl;
            break;
         case 5:
            cout << "Postorder Traversal:" << endl;
            avl.postorder(r);
            cout << endl;
            break;
         case 6:
            exit(1);
            break;
         default:
            cout << "Wrong Choice" << endl;
      }
   }
   return 0;
}
```