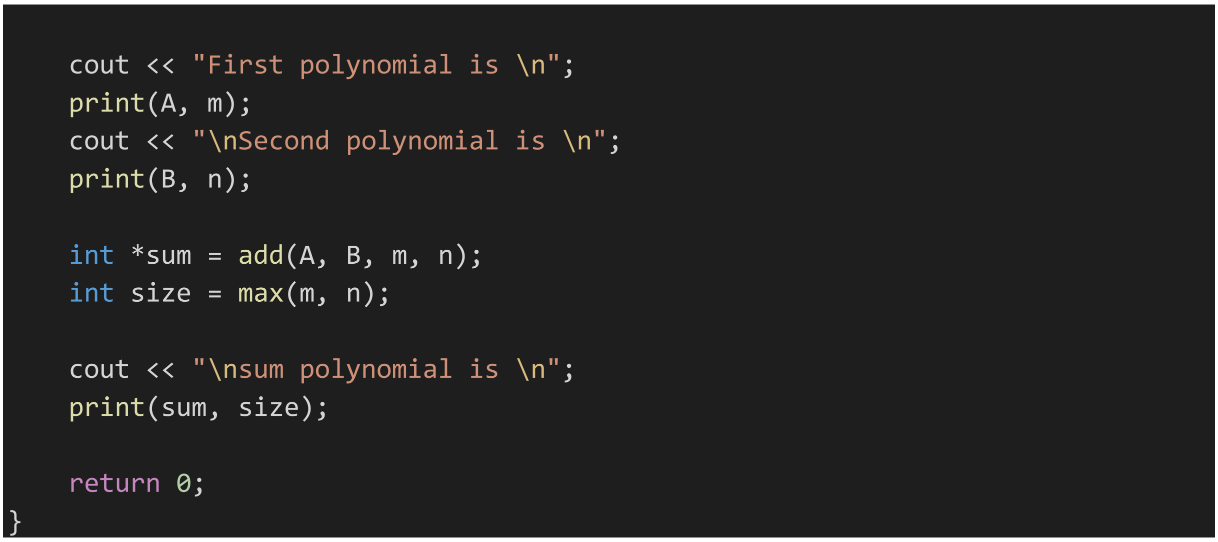
Polynomial Addition with Array

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

// max function
int max(int m, int n)
{
    return (m > n)? m: n;
}

// addition funtion
int *add(int A[], int B[], int m, int n)
{
   int size = max(m, n);
   int *sum = new int[size];

   for (int i = 0; i<m; i++)
     sum[i] = A[i];


   for (int i=0; i<n; i++)
       sum[i] += B[i];

   return sum;
}

// print function
void print(int poly[], int n)
{
    for (int i=0; i<n; i++)
    {
       cout << poly[i];
       if (i != 0)
        cout << "x^" << i ;
       if (i != n-1)
       cout << " + ";
    }
}

// main funtion
int main()
{
    int A[] = {10, 20, 30};
    int B[] = {40, 30, 20, 10};
    int m = sizeof(A)/sizeof(A[0]);
    int n = sizeof(B)/sizeof(B[0]);
```

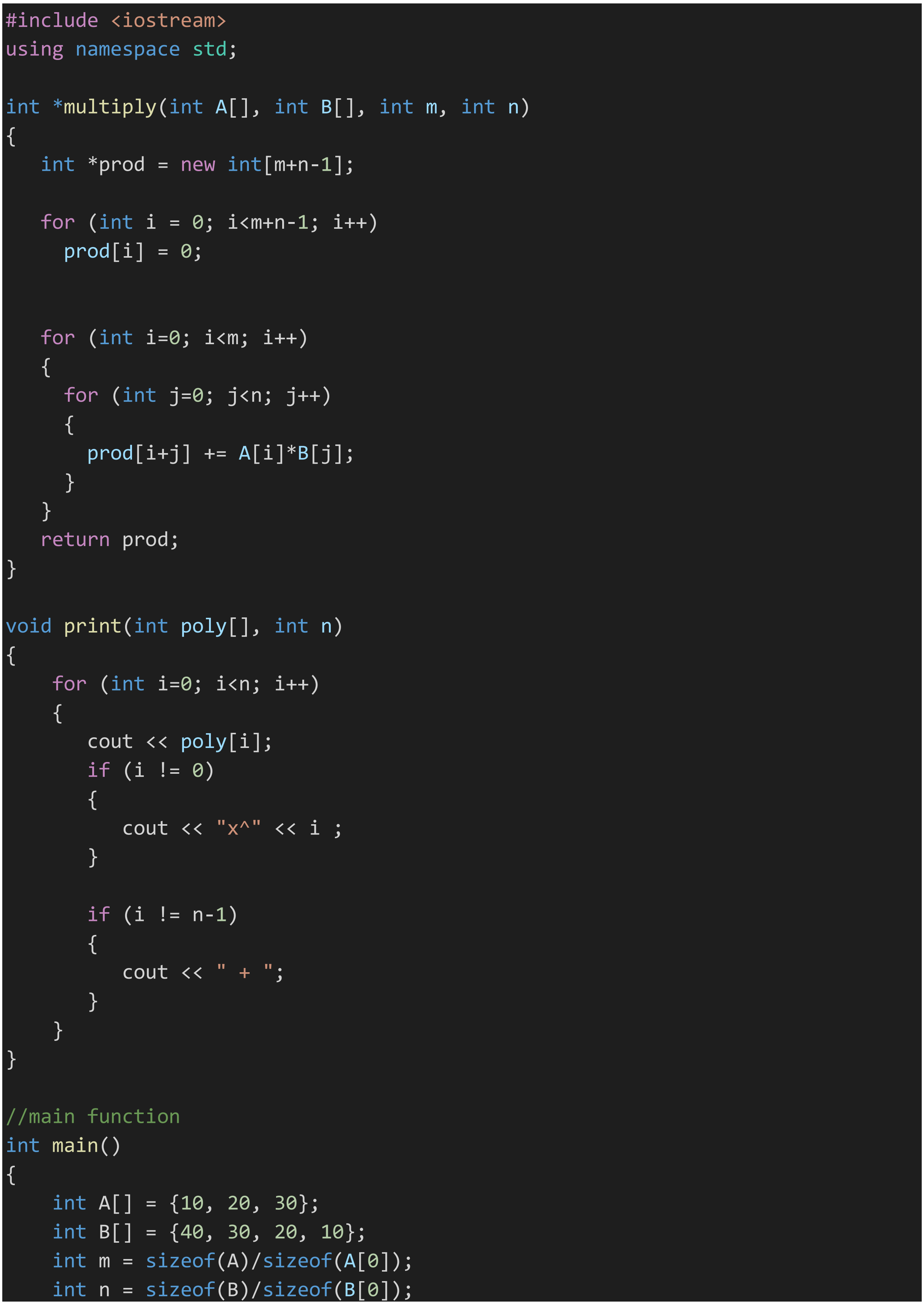


```cpp
    cout << "First polynomial is \n";
    print(A, m);
    cout << "\nSecond polynomial is \n";
    print(B, n);

    int *sum = add(A, B, m, n);
    int size = max(m, n);

    cout << "\nsum polynomial is \n";
    print(sum, size);

    return 0;
}
```

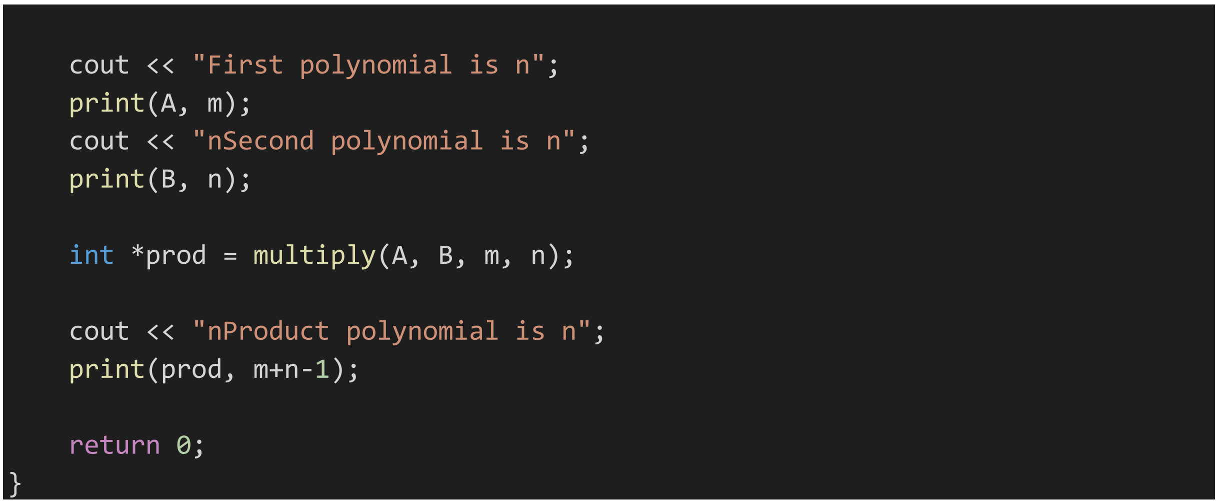
Output :

```
First polynomial is

10 + 20x^1 + 30x^2

Second polynomial is

40 + 30x^1 + 20x^2 + 10x^3

sum polynomial is

50 + 50x^1 + 50x^2 + 10x^3
```

Polynomial Multiplication with Array



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

int *multiply(int A[], int B[], int m, int n)
{
   int *prod = new int[m+n-1];

   for (int i = 0; i<m+n-1; i++)
     prod[i] = 0;


   for (int i=0; i<m; i++)
   {
     for (int j=0; j<n; j++)
     {
       prod[i+j] += A[i]*B[j];
     }
   }
   return prod;
}

void print(int poly[], int n)
{
    for (int i=0; i<n; i++)
    {
       cout << poly[i];
       if (i != 0)
       {
          cout << "x^" << i ;
       }

       if (i != n-1)
       {
          cout << " + ";
       }
    }
}

//main function
int main()
{
    int A[] = {10, 20, 30};
    int B[] = {40, 30, 20, 10};
    int m = sizeof(A)/sizeof(A[0]);
    int n = sizeof(B)/sizeof(B[0]);
```



```
    cout << "First polynomial is n";
    print(A, m);
    cout << "nSecond polynomial is n";
    print(B, n);

    int *prod = multiply(A, B, m, n);

    cout << "nProduct polynomial is n";
    print(prod, m+n-1);

    return 0;
}
```

Output :

First polynomial is

10 + 20x^1 + 30x^2

Second polynomial is

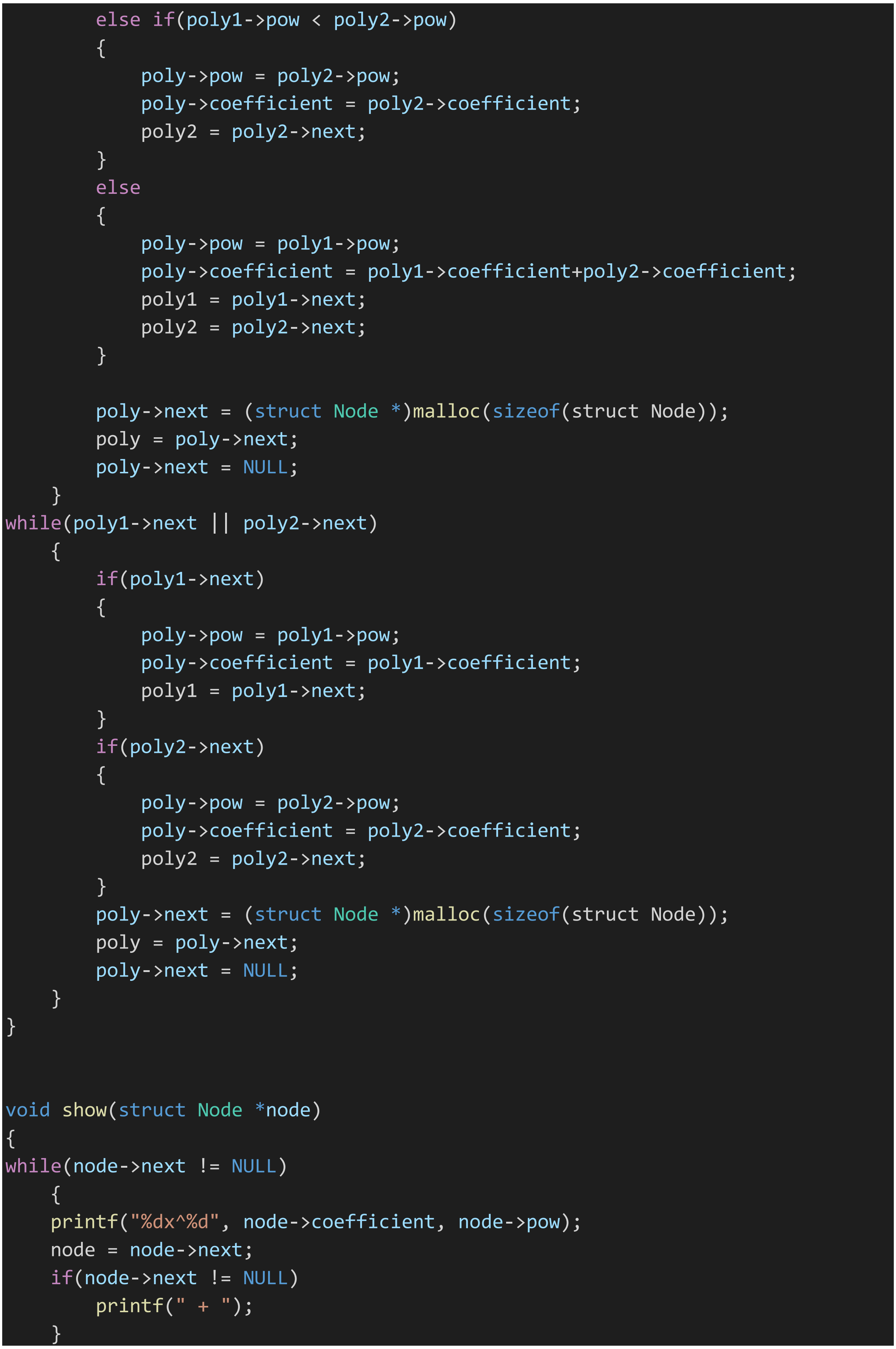
40 + 30x^1 + 20x^2 + 10x^3

Product polynomial is

400 + 1100x^1 + 2000x^2 + 1400x^3 + 800x^4 + 300x^5

Polynomial Addition with LL

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

struct Node
{
    int coefficient;
    int pow;
    struct Node *next;
};

void create_node(int x, int y, struct Node **temp)
{
    struct Node *r, *z;
    z = *temp;
    if(z == NULL)
    {
        r =(struct Node*)malloc(sizeof(struct Node));
        r->coefficient = x;
        r->pow = y;
        *temp = r;
        r->next = (struct Node*)malloc(sizeof(struct Node));
        r = r->next;
        r->next = NULL;
    }
    else
    {
        r->coefficient = x;
        r->pow = y;
        r->next = (struct Node*)malloc(sizeof(struct Node));
        r = r->next;
        r->next = NULL;
    }
}

void polyadd(struct Node *poly1, struct Node *poly2, struct Node *poly)
{
while(poly1->next && poly2->next)
    {
        if(poly1->pow > poly2->pow)
        {
            poly->pow = poly1->pow;
            poly->coefficient = poly1->coefficient;
            poly1 = poly1->next;
        }
```



```
        else if(poly1->pow < poly2->pow)
        {
            poly->pow = poly2->pow;
            poly->coefficient = poly2->coefficient;
            poly2 = poly2->next;
        }
        else
        {
            poly->pow = poly1->pow;
            poly->coefficient = poly1->coefficient+poly2->coefficient;
            poly1 = poly1->next;
            poly2 = poly2->next;
        }

        poly->next = (struct Node *)malloc(sizeof(struct Node));
        poly = poly->next;
        poly->next = NULL;
    }
while(poly1->next || poly2->next)
    {
        if(poly1->next)
        {
            poly->pow = poly1->pow;
            poly->coefficient = poly1->coefficient;
            poly1 = poly1->next;
        }
        if(poly2->next)
        {
            poly->pow = poly2->pow;
            poly->coefficient = poly2->coefficient;
            poly2 = poly2->next;
        }
        poly->next = (struct Node *)malloc(sizeof(struct Node));
        poly = poly->next;
        poly->next = NULL;
    }
}


void show(struct Node *node)
{
while(node->next != NULL)
    {
    printf("%dx^%d", node->coefficient, node->pow);
    node = node->next;
    if(node->next != NULL)
        printf(" + ");
    }
```

```
}


int main()
{
    struct Node *poly1 = NULL, *poly2 = NULL, *poly = NULL;

    create_node(5,2,&poly1);
    create_node(4,1,&poly1);
    create_node(2,0,&poly1);
    create_node(5,1,&poly2);
    create_node(5,0,&poly2);

    printf("1st Number: ");
    show(poly1);

    printf("\n2nd Number: ");
    show(poly2);

    poly = (struct Node *)malloc(sizeof(struct Node));
    polyadd(poly1, poly2, poly);
    printf("\nAdded polynomial: ");
    show(poly);

return 0;
}
```

Output :

```
1st Number: 5x^2 + 4x^1 + 2x^0

2nd Number: 5x^1 + 5x^0

Added polynomial: 5x^2 + 9x^1 + 7x^0
```

Polynomial Multiplication with LL

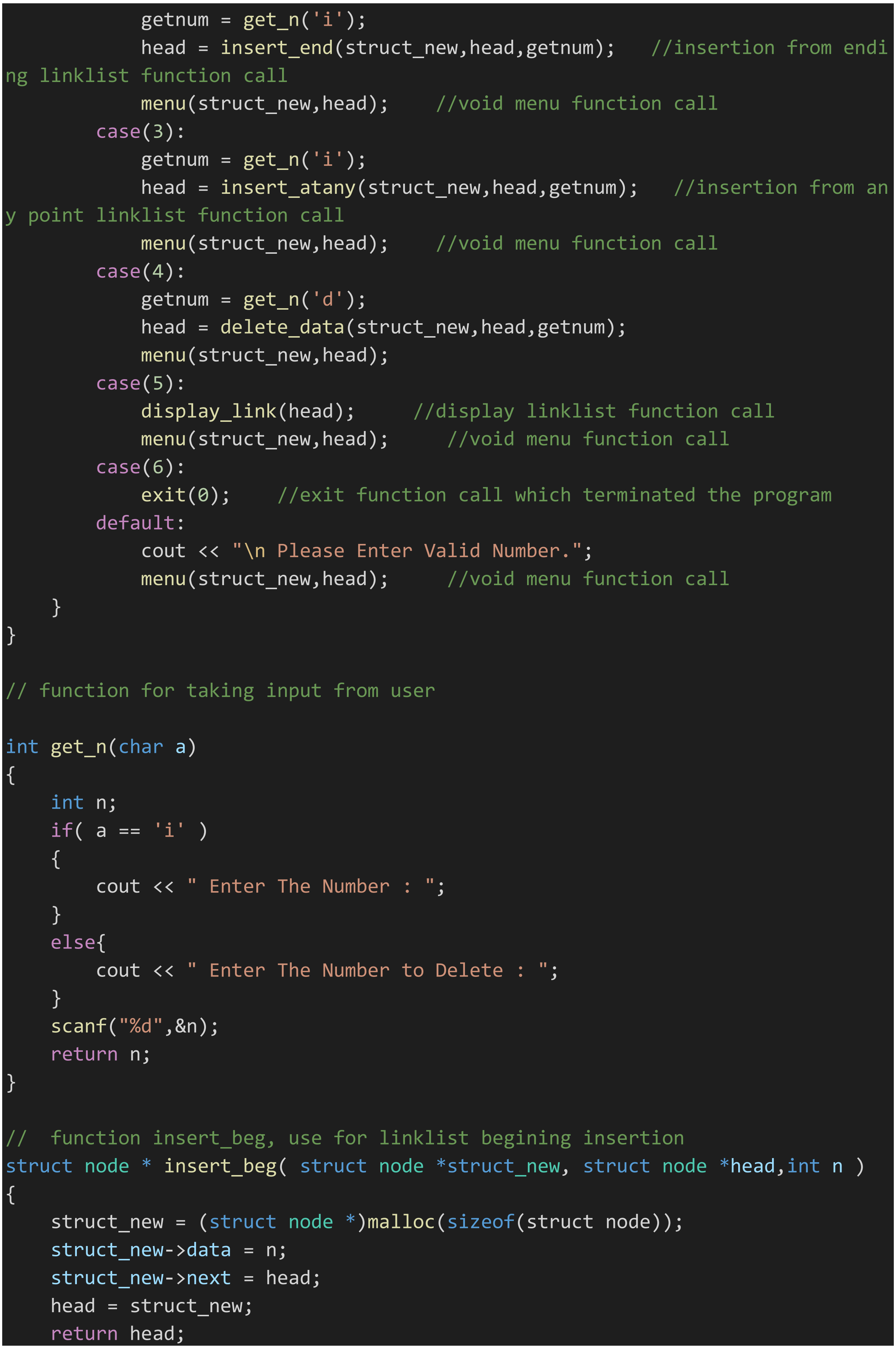
```
#include <iostream>
using namespace std;


struct Node {
    int coefficient, power;
    Node* next;
};

Node* addnode(Node* start, int coeff, int power)
{
    Node* newnode = new Node;
    newnode->coefficient = coeff;
    newnode->power = power;
    newnode->next = NULL;

    if (start == NULL)
        return newnode;

    Node* ptr = start;
    while (ptr->next != NULL)
        ptr = ptr->next;
    ptr->next = newnode;

    return start;
}

void printList(struct Node* ptr)
{
    while (ptr->next != NULL) {
        cout << ptr->coefficient << "x^" << ptr->power << " + ";

        ptr = ptr->next;
    }
    cout << ptr->coefficient << "\n";
}

void removeDuplicates(Node* start)
{
    Node *ptr1, *ptr2, *dup;
    ptr1 = start;

    while (ptr1 != NULL && ptr1->next != NULL) {
        ptr2 = ptr1;
        while (ptr2->next != NULL) {
            if (ptr1->power == ptr2->next->power) {
                ptr1->coefficient = ptr1->coefficient + ptr2->next-
>coefficient;
                dup = ptr2->next;
                ptr2->next = ptr2->next->next;
                delete (dup);
            }
            else
                ptr2 = ptr2->next;
        }
        ptr1 = ptr1->next;
    }
}


Node* multiply(Node* poly1, Node* poly2,
               Node* poly3)
{
    Node *ptr1, *ptr2;
    ptr1 = poly1;
    ptr2 = poly2;
    while (ptr1 != NULL) {
        while (ptr2 != NULL) {
            int coeff, power;
            coeff = ptr1->coefficient * ptr2->coefficient;
            power = ptr1->power + ptr2->power;
            poly3 = addnode(poly3, coeff, power);
            ptr2 = ptr2->next;
        }
        ptr2 = poly2;
        ptr1 = ptr1->next;
    }
    removeDuplicates(poly3);
    return poly3;
}

// Driver Code
int main()
{

    Node *poly1 = NULL, *poly2 = NULL, *poly3 = NULL;
    poly1 = addnode(poly1, 3, 2);
    poly1 = addnode(poly1, 5, 1);
    poly1 = addnode(poly1, 6, 0);
    poly2 = addnode(poly2, 6, 1);
    poly2 = addnode(poly2, 8, 0);

    cout << "1st Polynomial:- ";
    printList(poly1);

    cout << "2nd Polynomial:- ";
    printList(poly2);

    poly3 = multiply(poly1, poly2, poly3);

    cout << "Resultant Polynomial:- ";
    printList(poly3);

    return 0;
}
```

Output :

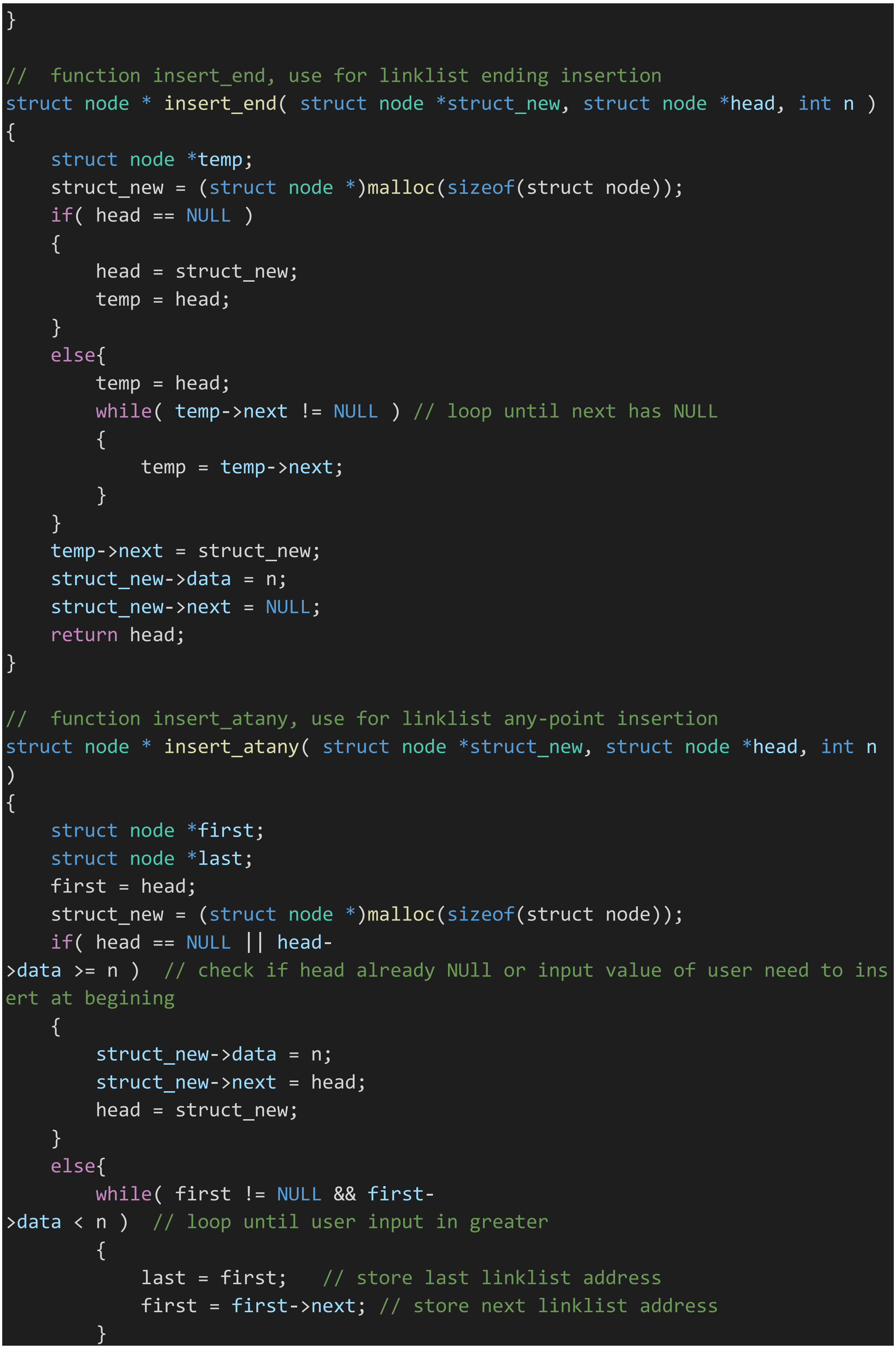
```
1st Polynomial:- 3x^2 + 5x^1 + 6

2nd Polynomial:- 6x^1 + 8

Resultant Polynomial:- 18x^3 + 54x^2 + 76x^1 + 48
```

Singly LinkList

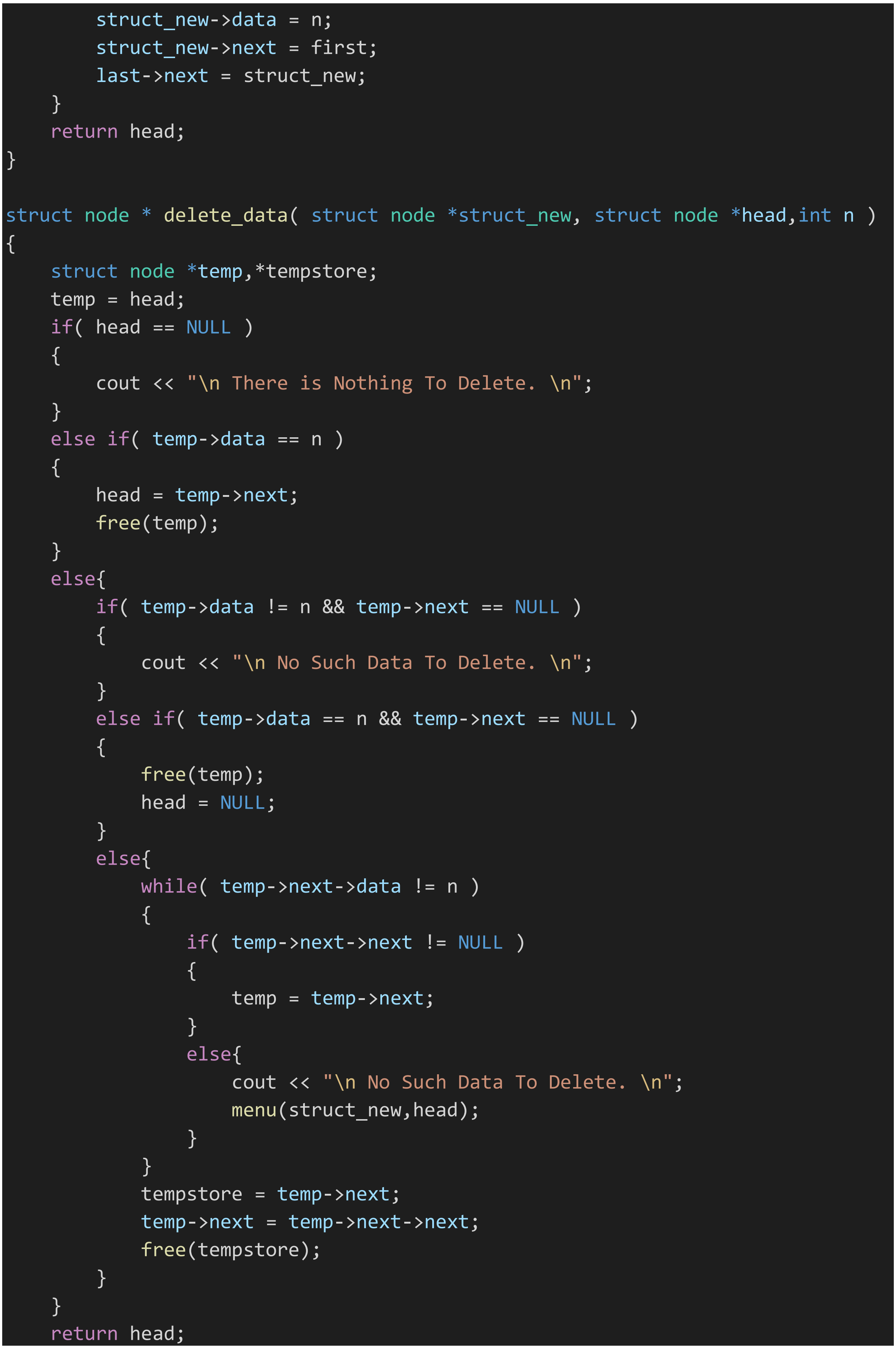
```
#include<iostream>
using namespace std;

// Structure Declaration
struct node{
    int data;
    struct node *next;
};

// Functions Declaration
void menu(struct node *,struct node *);
int get_n(char);
struct node * insert_beg(struct node *,struct node *,int);
struct node * insert_end(struct node *,struct node *, int);
struct node * insert_atany(struct node *,struct node *, int);
struct node * delete_data(struct node *,struct node *, int);
void display_link(struct node *);

// Void Main
int main()
{
    struct node *struct_new;
    struct node *head = NULL;
    menu(struct_new,head); // Calling menu funtion
    return 0;
}

// menu function
void menu( struct node *struct_new, struct node *head )
{
    int n,getnum;
    cout << "\n 1 . Add New Data To Linklist From Begining. \n 2 . Add New Data To Linklist From Ending.\n 3 . Add New Data To Linklist At Any Place. \n 4 . Delete a Number From The Link-List. \n 5 . Display LinkList Till Now. \n 6 . Exit. \n";
    cin >> n;
    // Switch case which check the user input and run specified function
    switch(n)
    {
        case(1):
            getnum = get_n('i');
            head = insert_beg(struct_new,head,getnum);   //insertion from begining linklist function call
            menu(struct_new,head);    //void menu function call
        case(2):
```



```
            getnum = get_n('i');
            head = insert_end(struct_new,head,getnum);   //insertion from ending linklist function call
            menu(struct_new,head);    //void menu function call
        case(3):
            getnum = get_n('i');
            head = insert_atany(struct_new,head,getnum);   //insertion from any point linklist function call
            menu(struct_new,head);    //void menu function call
        case(4):
            getnum = get_n('d');
            head = delete_data(struct_new,head,getnum);
            menu(struct_new,head);
        case(5):
            display_link(head);     //display linklist function call
            menu(struct_new,head);     //void menu function call
        case(6):
            exit(0);    //exit function call which terminated the program
        default:
            cout << "\n Please Enter Valid Number.";
            menu(struct_new,head);     //void menu function call
    }
}

// function for taking input from user

int get_n(char a)
{
    int n;
    if( a == 'i' )
    {
        cout << " Enter The Number : ";
    }
    else{
        cout << " Enter The Number to Delete : ";
    }
    scanf("%d",&n);
    return n;
}

//  function insert_beg, use for linklist begining insertion
struct node * insert_beg( struct node *struct_new, struct node *head,int n )
{
    struct_new = (struct node *)malloc(sizeof(struct node));
    struct_new->data = n;
    struct_new->next = head;
    head = struct_new;
    return head;
```



```
}

//  function insert_end, use for linklist ending insertion
struct node * insert_end( struct node *struct_new, struct node *head, int n )
{
    struct node *temp;
    struct_new = (struct node *)malloc(sizeof(struct node));
    if( head == NULL )
    {
        head = struct_new;
        temp = head;
    }
    else{
        temp = head;
        while( temp->next != NULL ) // loop until next has NULL
        {
            temp = temp->next;
        }
    }
    temp->next = struct_new;
    struct_new->data = n;
    struct_new->next = NULL;
    return head;
}

//  function insert_atany, use for linklist any-point insertion
struct node * insert_atany( struct node *struct_new, struct node *head, int n
)
{
    struct node *first;
    struct node *last;
    first = head;
    struct_new = (struct node *)malloc(sizeof(struct node));
    if( head == NULL || head-
>data >= n )  // check if head already NUll or input value of user need to ins
ert at begining
    {
        struct_new->data = n;
        struct_new->next = head;
        head = struct_new;
    }
    else{
        while( first != NULL && first-
>data < n )  // loop until user input in greater
        {
            last = first;   // store last linklist address
            first = first->next; // store next linklist address
        }
```



```
        struct_new->data = n;
        struct_new->next = first;
        last->next = struct_new;
    }
    return head;
}

struct node * delete_data( struct node *struct_new, struct node *head,int n )
{
    struct node *temp,*tempstore;
    temp = head;
    if( head == NULL )
    {
        cout << "\n There is Nothing To Delete. \n";
    }
    else if( temp->data == n )
    {
        head = temp->next;
        free(temp);
    }
    else{
        if( temp->data != n && temp->next == NULL )
        {
            cout << "\n No Such Data To Delete. \n";
        }
        else if( temp->data == n && temp->next == NULL )
        {
            free(temp);
            head = NULL;
        }
        else{
            while( temp->next->data != n )
            {
                if( temp->next->next != NULL )
                {
                    temp = temp->next;
                }
                else{
                    cout << "\n No Such Data To Delete. \n";
                    menu(struct_new,head);
                }
            }
            tempstore = temp->next;
            temp->next = temp->next->next;
            free(tempstore);
        }
    }
    return head;
```

```
}

//  function display_link will display the linklist elements
void display_link(struct node *head)
{
    struct node *temp;
    if(head == NULL)    // check wheater the head is null
    {
        cout << "\nThere Is Nothing To Display.\n";
    }
    else
    {
        temp = head;
        cout << "\nThe List is : \n";
        while(temp->next != NULL)   // print all the elements from the link-
list
        {
            cout << temp->data << " => ";
            temp = temp->next;
        }
       cout << temp->data;
    }
}
```

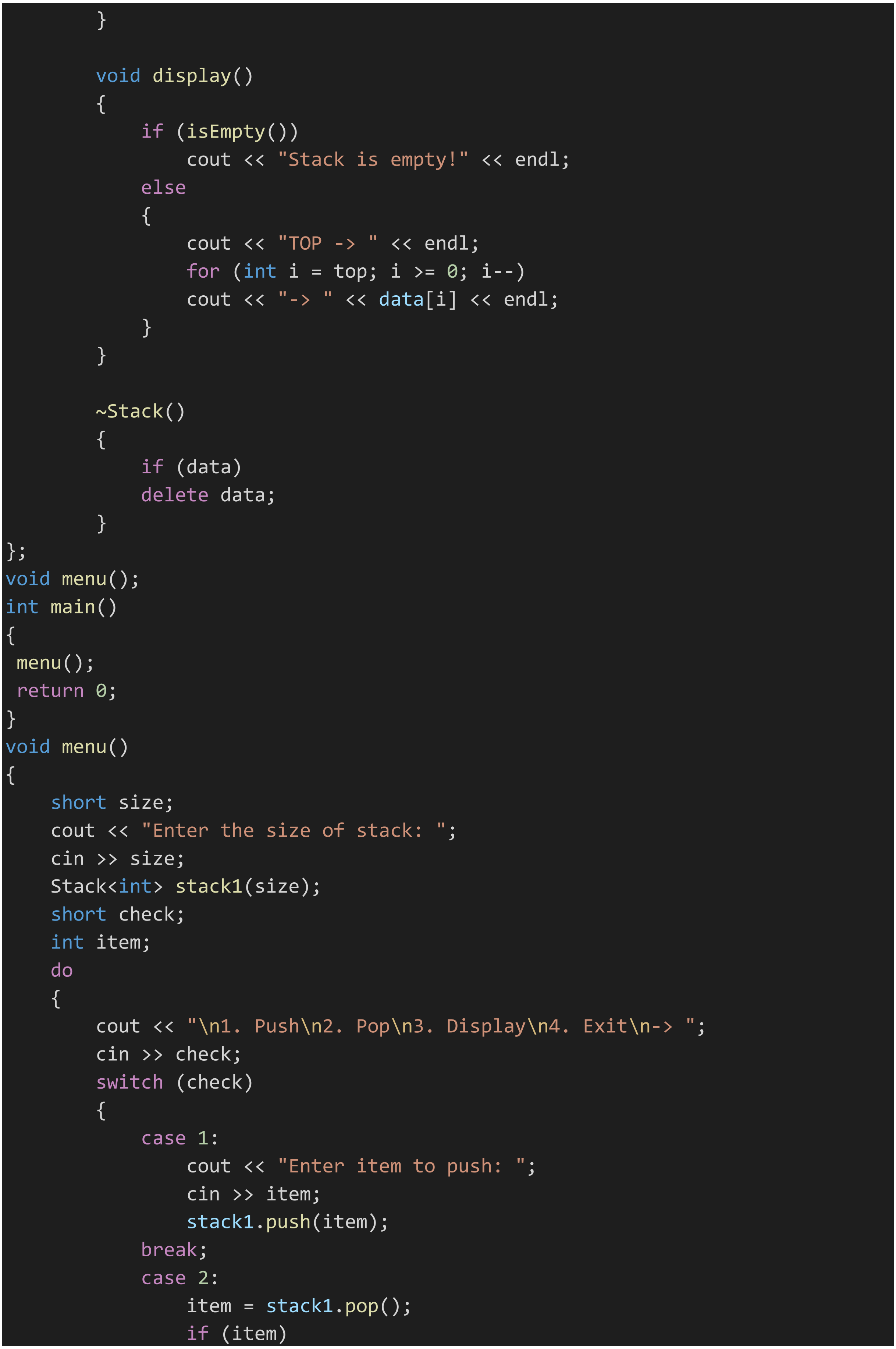
Output :

```
 1 . Add New Data To Linklist From Begining.

 2 . Add New Data To Linklist From Ending.

 3 . Add New Data To Linklist At Any Place.

 4 . Delete a Number From The Link-List.

 5 . Display LinkList Till Now.

 6 . Exit.

1

 Enter The Number : 3


 1 . Add New Data To Linklist From Begining.

 2 . Add New Data To Linklist From Ending.
 3 . Add New Data To Linklist At Any Place.
 4 . Delete a Number From The Link-List.
 5 . Display LinkList Till Now.
 6 . Exit.
1
 Enter The Number : 2

 1 . Add New Data To Linklist From Begining.
 2 . Add New Data To Linklist From Ending.
 3 . Add New Data To Linklist At Any Place.
 4 . Delete a Number From The Link-List.
 5 . Display LinkList Till Now.
 6 . Exit.
2
 Enter The Number : 6

 1 . Add New Data To Linklist From Begining.
 2 . Add New Data To Linklist From Ending.
 3 . Add New Data To Linklist At Any Place.
 4 . Delete a Number From The Link-List.
 5 . Display LinkList Till Now.
 6 . Exit.
3
 Enter The Number : 5

 1 . Add New Data To Linklist From Begining.
 2 . Add New Data To Linklist From Ending.

 3 . Add New Data To Linklist At Any Place.

 4 . Delete a Number From The Link-List.

 5 . Display LinkList Till Now.

 6 . Exit.

5


The List is :

2 => 3 => 5 => 6 %d

 1 . Add New Data To Linklist From Begining.

 2 . Add New Data To Linklist From Ending.

 3 . Add New Data To Linklist At Any Place.

 4 . Delete a Number From The Link-List.

 5 . Display LinkList Till Now.

 6 . Exit.

4

 Enter The Number to Delete : 5


 1 . Add New Data To Linklist From Begining.

 2 . Add New Data To Linklist From Ending.

 3 . Add New Data To Linklist At Any Place.

 4 . Delete a Number From The Link-List.

 5 . Display LinkList Till Now.

 6 . Exit.

5
```

```
The List is :
2 => 3 => 6
 1 . Add New Data To Linklist From Begining.
 2 . Add New Data To Linklist From Ending.
 3 . Add New Data To Linklist At Any Place.
 4 . Delete a Number From The Link-List.
 5 . Display LinkList Till Now.
 6 . Exit.
```

Stack With Array

```
#include <iostream>
using namespace std;
template <typename T>

class Stack
{
    T *data;
    short top, size;

    public:
        Stack(int size)
        {
            if (size < 1)
                size = 5;
                this->size = size;
                top = -1;
                data = new T[this->size];
        }

        bool isFull()
        {
            return (top > size - 1);
        }

        bool isEmpty()
        {
            return (top < 0);
        }

        void push(T item)
        {
            if (isFull())
            {
                cout << "Stack Overflow" << endl;
                return;
            }
            data[++top] = item;
        }

        T pop()
        {
            if (isEmpty())
            {
                cout << "Stack is empty!" << endl;
                return NULL;
            }
            return data[top--];
```



```
        }

        void display()
        {
            if (isEmpty())
                cout << "Stack is empty!" << endl;
            else
            {
                cout << "TOP -> " << endl;
                for (int i = top; i >= 0; i--)
                cout << "-> " << data[i] << endl;
            }
        }

        ~Stack()
        {
            if (data)
            delete data;
        }
};
void menu();
int main()
{
 menu();
 return 0;
}
void menu()
{
    short size;
    cout << "Enter the size of stack: ";
    cin >> size;
    Stack<int> stack1(size);
    short check;
    int item;
    do
    {
        cout << "\n1. Push\n2. Pop\n3. Display\n4. Exit\n-> ";
        cin >> check;
        switch (check)
        {
            case 1:
                cout << "Enter item to push: ";
                cin >> item;
                stack1.push(item);
            break;
            case 2:
                item = stack1.pop();
                if (item)
                cout << "Deleted Item: " << item << endl;
            break;
            case 3:
                stack1.display();
            break;
            case 4:
            break;
            default:
                cout << "Select Proper Selection." << endl;
        }
    } while (check);
}
```

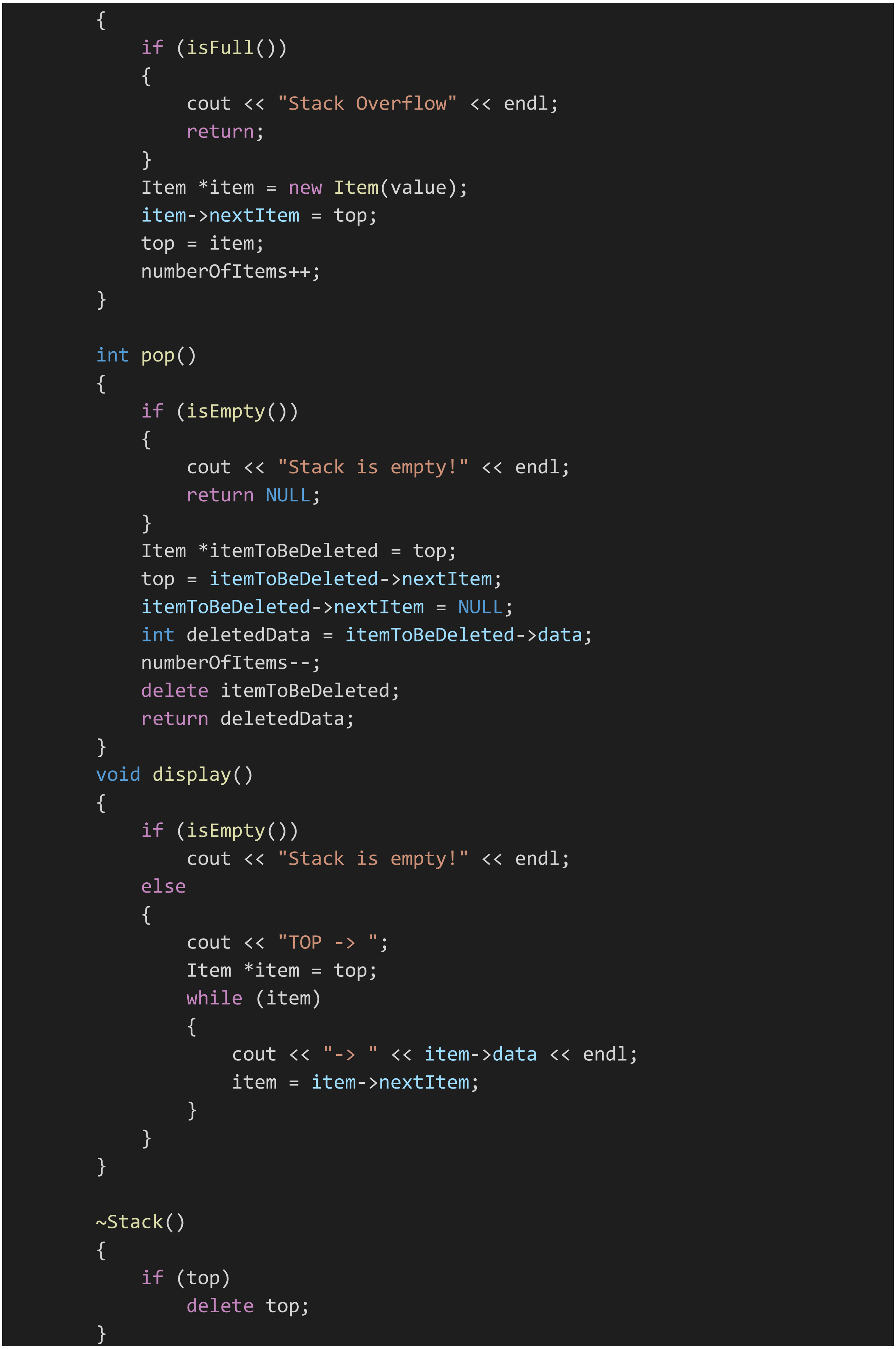
Output :

```
Enter the size of stack: 4

1. Push

2. Pop

3. Display

4. Exit

-> 1

Enter item to push: 5


1. Push

2. Pop

3. Display

4. Exit

-> 1

Enter item to push: 3


1. Push

2. Pop

3. Display
```

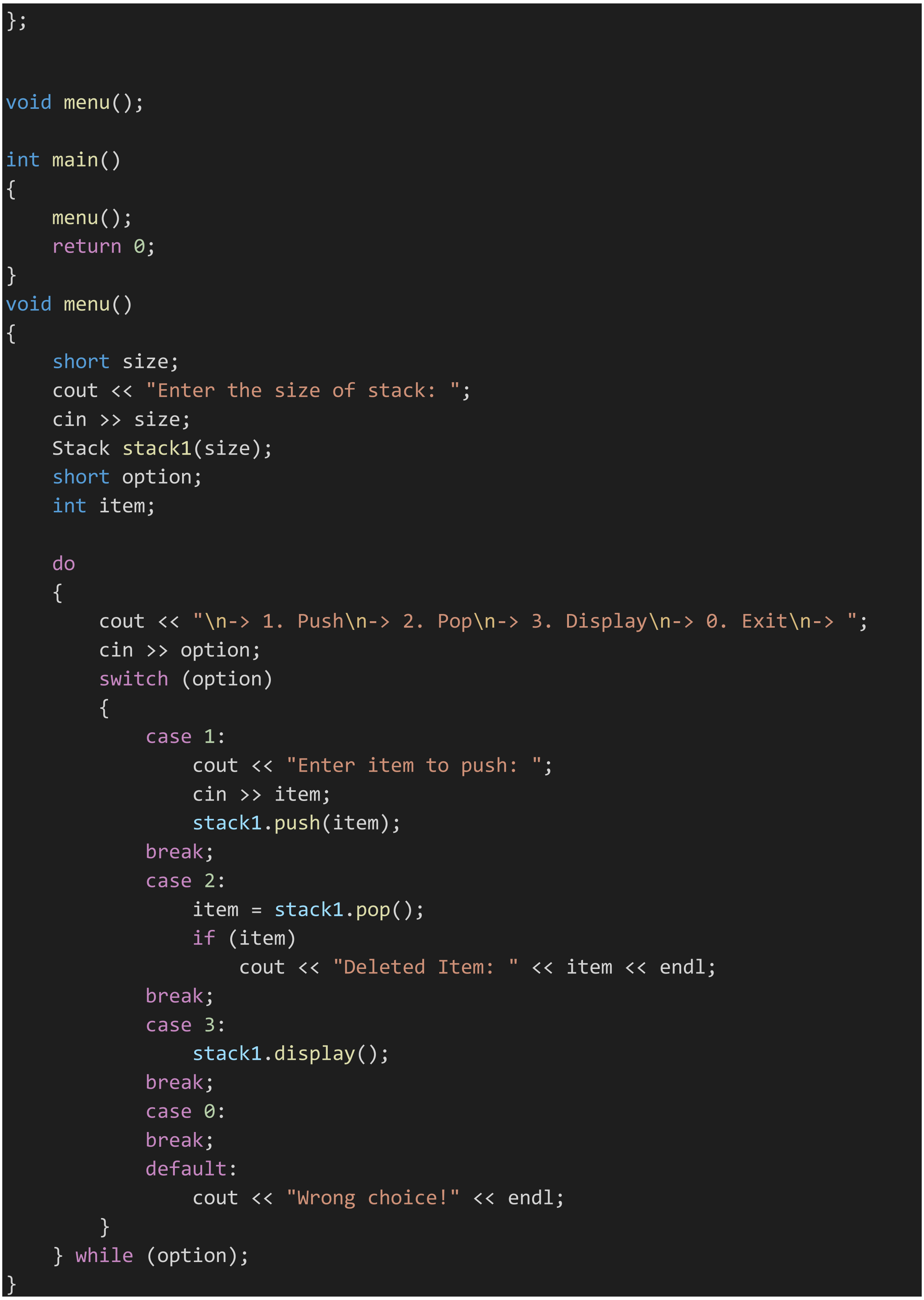
```
4. Exit
-> 3
TOP ->
-> 3
-> 5

1. Push
2. Pop
3. Display
4. Exit
-> 2
Deleted Item: 3

1. Push
2. Pop
3. Display
4. Exit
-> 3
TOP ->
-> 5

1. Push
2. Pop
3. Display
4. Exit
-> 4
```

Stack With LL

```
#include <iostream>
using namespace std;
class Stack
{
    class Item
    {
        public:
        int data;
        Item *nextItem;

        Item(int value)
        {
            data = value;
            nextItem = NULL;
        }

        ~Item()
        {
            if (nextItem)
            delete nextItem;
        }
    };
    Item *top;
    short numberOfItems, size;
    public:
        Stack(int size)
        {
            if (size < 1)
                size = 5;
            this->size = size;
            numberOfItems = 0;
            top = NULL;
        }

        bool isFull()
        {
            return (numberOfItems >= size);
        }

        bool isEmpty()
        {
            return !top;
        }

        void push(int value)
```



```
    {
        if (isFull())
        {
            cout << "Stack Overflow" << endl;
            return;
        }
        Item *item = new Item(value);
        item->nextItem = top;
        top = item;
        numberOfItems++;
    }

    int pop()
    {
        if (isEmpty())
        {
            cout << "Stack is empty!" << endl;
            return NULL;
        }
        Item *itemToBeDeleted = top;
        top = itemToBeDeleted->nextItem;
        itemToBeDeleted->nextItem = NULL;
        int deletedData = itemToBeDeleted->data;
        numberOfItems--;
        delete itemToBeDeleted;
        return deletedData;
    }
    void display()
    {
        if (isEmpty())
            cout << "Stack is empty!" << endl;
        else
        {
            cout << "TOP -> ";
            Item *item = top;
            while (item)
            {
                cout << "-> " << item->data << endl;
                item = item->nextItem;
            }
        }
    }

    ~Stack()
    {
        if (top)
            delete top;
    }
```



```cpp
};


void menu();

int main()
{
    menu();
    return 0;
}
void menu()
{
    short size;
    cout << "Enter the size of stack: ";
    cin >> size;
    Stack stack1(size);
    short option;
    int item;

    do
    {
        cout << "\n-> 1. Push\n-> 2. Pop\n-> 3. Display\n-> 0. Exit\n-> ";
        cin >> option;
        switch (option)
        {
            case 1:
                cout << "Enter item to push: ";
                cin >> item;
                stack1.push(item);
            break;
            case 2:
                item = stack1.pop();
                if (item)
                    cout << "Deleted Item: " << item << endl;
            break;
            case 3:
                stack1.display();
            break;
            case 0:
            break;
            default:
                cout << "Wrong choice!" << endl;
        }
    } while (option);
}
```

Output :

```
Enter the size of stack: 5

-> 1. Push
-> 2. Pop
-> 3. Display
-> 0. Exit
-> 1
Enter item to push: 6

-> 1. Push
-> 2. Pop
-> 3. Display
-> 0. Exit
-> 2
Deleted Item: 6

-> 1. Push
-> 2. Pop
-> 3. Display
-> 0. Exit
-> 3
Stack is empty!

-> 1. Push
-> 2. Pop
-> 3. Display
-> 0. Exit
-> 1
Enter item to push: 8

-> 1. Push
-> 2. Pop
-> 3. Display
-> 0. Exit
-> 1
Enter item to push: 5

-> 1. Push
-> 2. Pop
-> 3. Display
-> 0. Exit
-> 1
Enter item to push: 2

-> 1. Push
-> 2. Pop
-> 3. Display
-> 0. Exit
-> 3
TOP -> -> 2
-> 5
-> 8


-> 1. Push

-> 2. Pop

-> 3. Display

-> 0. Exit
```

Factorial using Recursion

```cpp
#include <iostream>
using namespace std;

// Get User Input
int input()
{
    int n;
    cout << "Enter The Number";
    cin >> n;
    return n;
}


// factorial function return all the factorial value until the limit
int factorial(int n)
{
   if (n < 2)
   {
       return 1;
   }
   else
   {
       return n*factorial(n-1);
   }
}


int main() {
   int n = input();
   cout << "Factorial of " << n << " is " << factorial(n);
   return 0;
}
```

Output :

Factorial of 7 is 5040

Fibonacci using Recursion

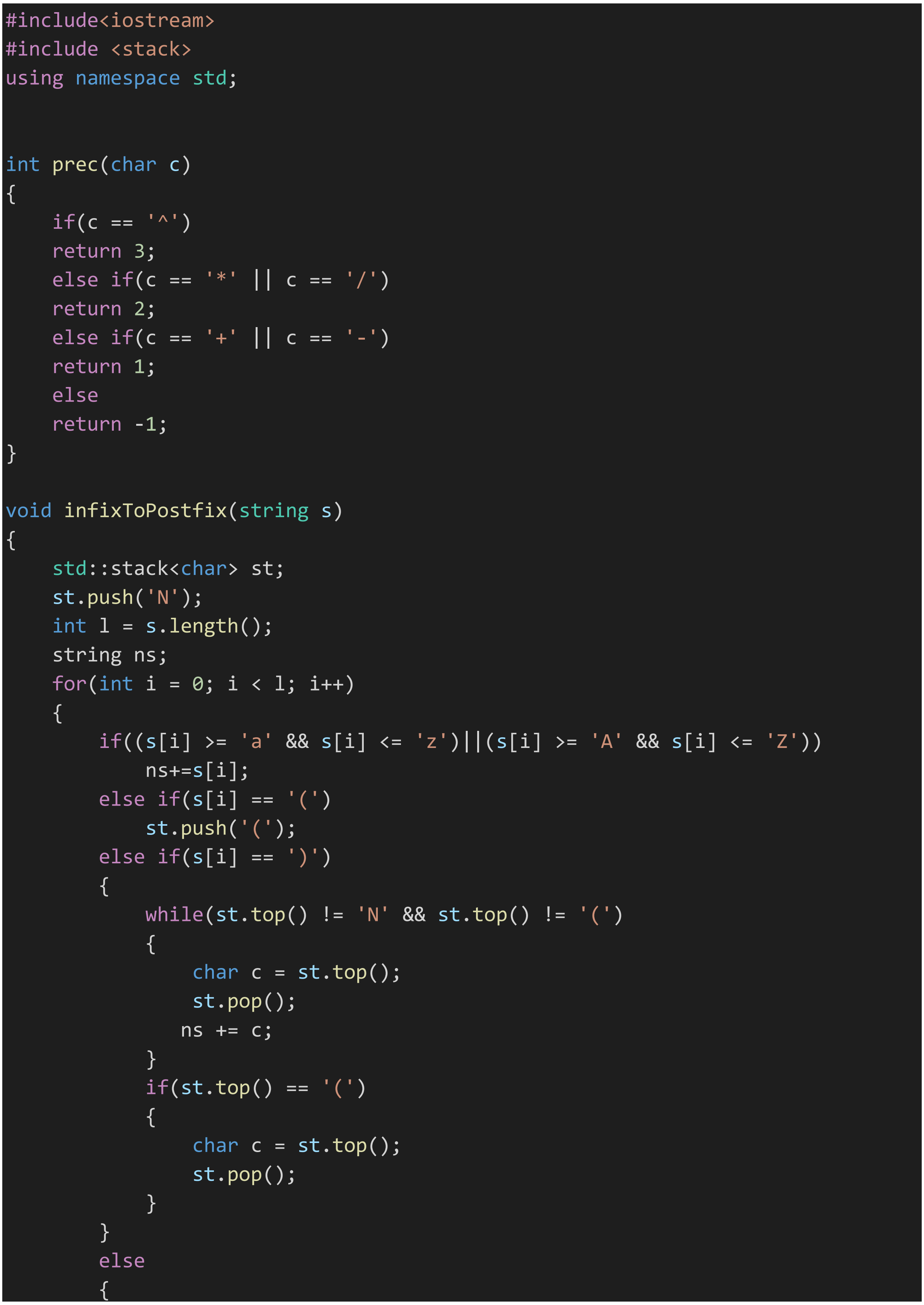
```
#include <iostream>
using namespace std;


int fibonacci(int num)
{
   if((num==1)||(num==0))
   {
      return(num);
   }
   else
   {
      return(fibonacci(num-1)+fibonacci(num-2));
   }
}
int main()
{
   int limit, i = 0;
   cout << "Enter the limit for Fibonacci : ";
   cin >> limit;
   cout << "\nFibonnaci Series : ";
   while( i < limit ) {
      cout << " " << fibonacci(i);
      i++;
   }
   return 0;
}
```

Output :

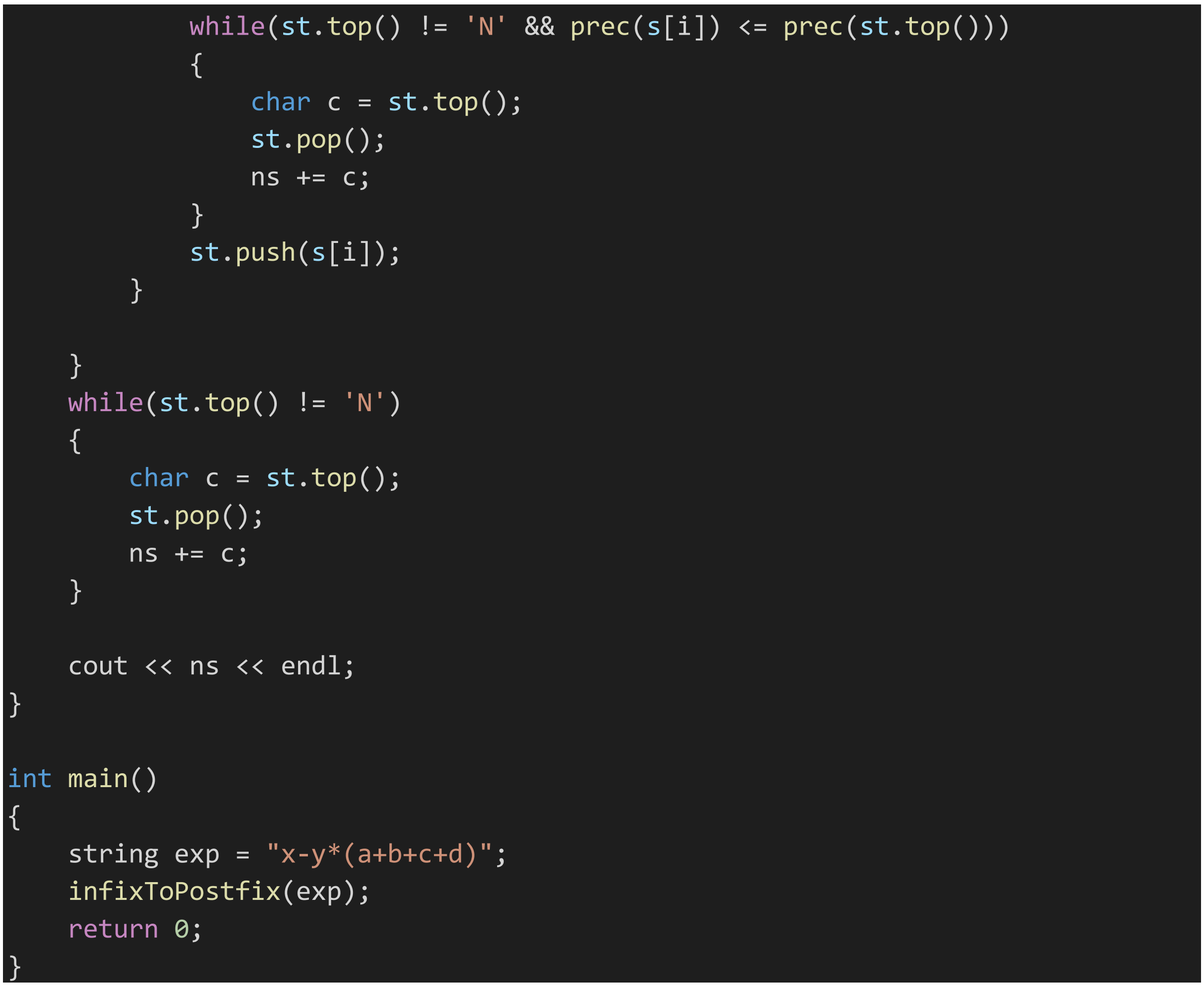
Enter the limit for Fibonacci : 6

Fibonnaci Series : 0 1 1 2 3 5

Infix To Postfix



```
#include<iostream>
#include <stack>
using namespace std;


int prec(char c)
{
    if(c == '^')
    return 3;
    else if(c == '*' || c == '/')
    return 2;
    else if(c == '+' || c == '-')
    return 1;
    else
    return -1;
}

void infixToPostfix(string s)
{
    std::stack<char> st;
    st.push('N');
    int l = s.length();
    string ns;
    for(int i = 0; i < l; i++)
    {
        if((s[i] >= 'a' && s[i] <= 'z')||(s[i] >= 'A' && s[i] <= 'Z'))
            ns+=s[i];
        else if(s[i] == '(')
            st.push('(');
        else if(s[i] == ')')
        {
            while(st.top() != 'N' && st.top() != '(')
            {
                char c = st.top();
                st.pop();
               ns += c;
            }
            if(st.top() == '(')
            {
                char c = st.top();
                st.pop();
            }
        }
        else
        {
```

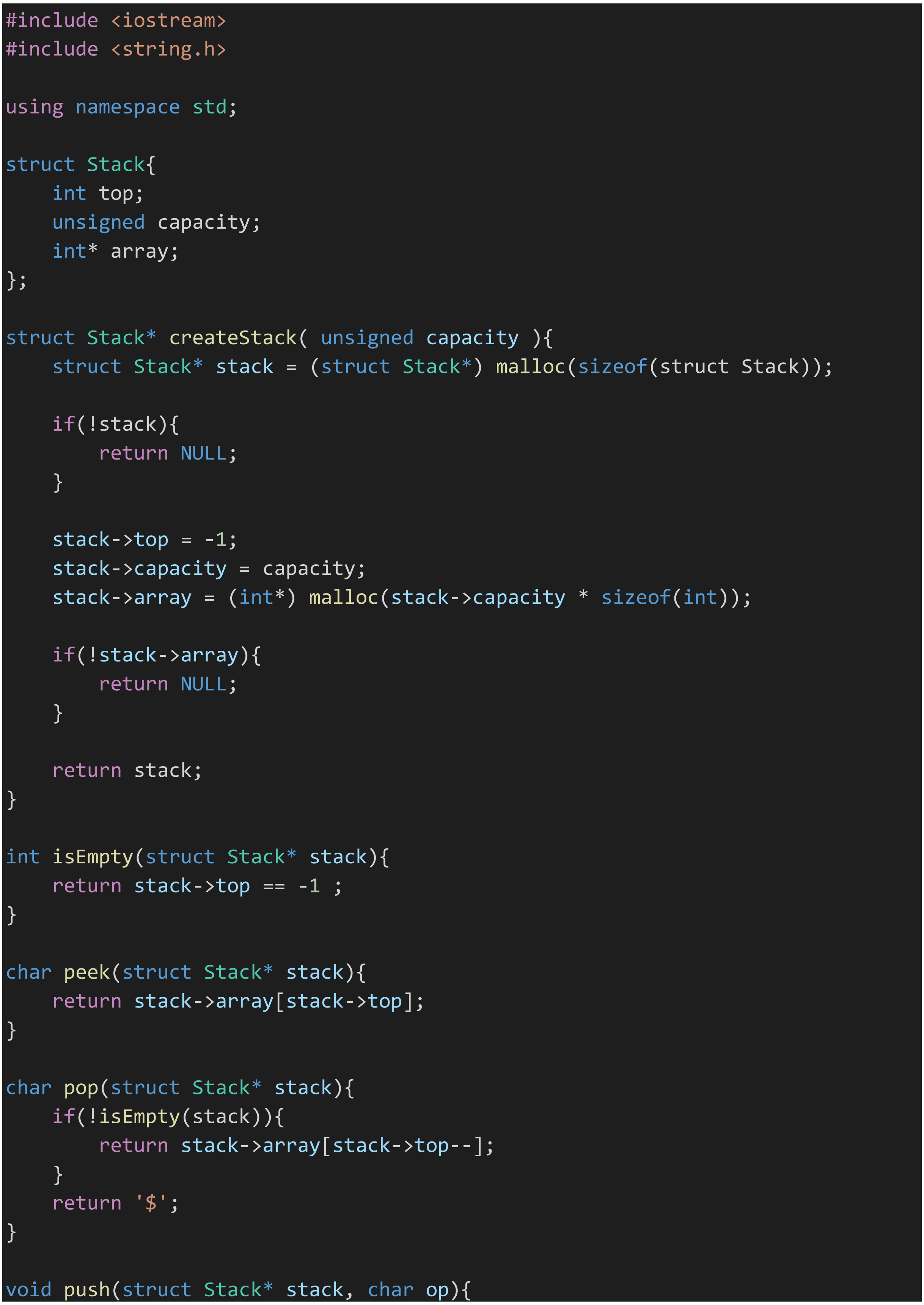


```
            while(st.top() != 'N' && prec(s[i]) <= prec(st.top()))
            {
                char c = st.top();
                st.pop();
                ns += c;
            }
            st.push(s[i]);
        }

    }
    while(st.top() != 'N')
    {
        char c = st.top();
        st.pop();
        ns += c;
    }

    cout << ns << endl;
}

int main()
{
    string exp = "x-y*(a+b+c+d)";
    infixToPostfix(exp);
    return 0;
}
```

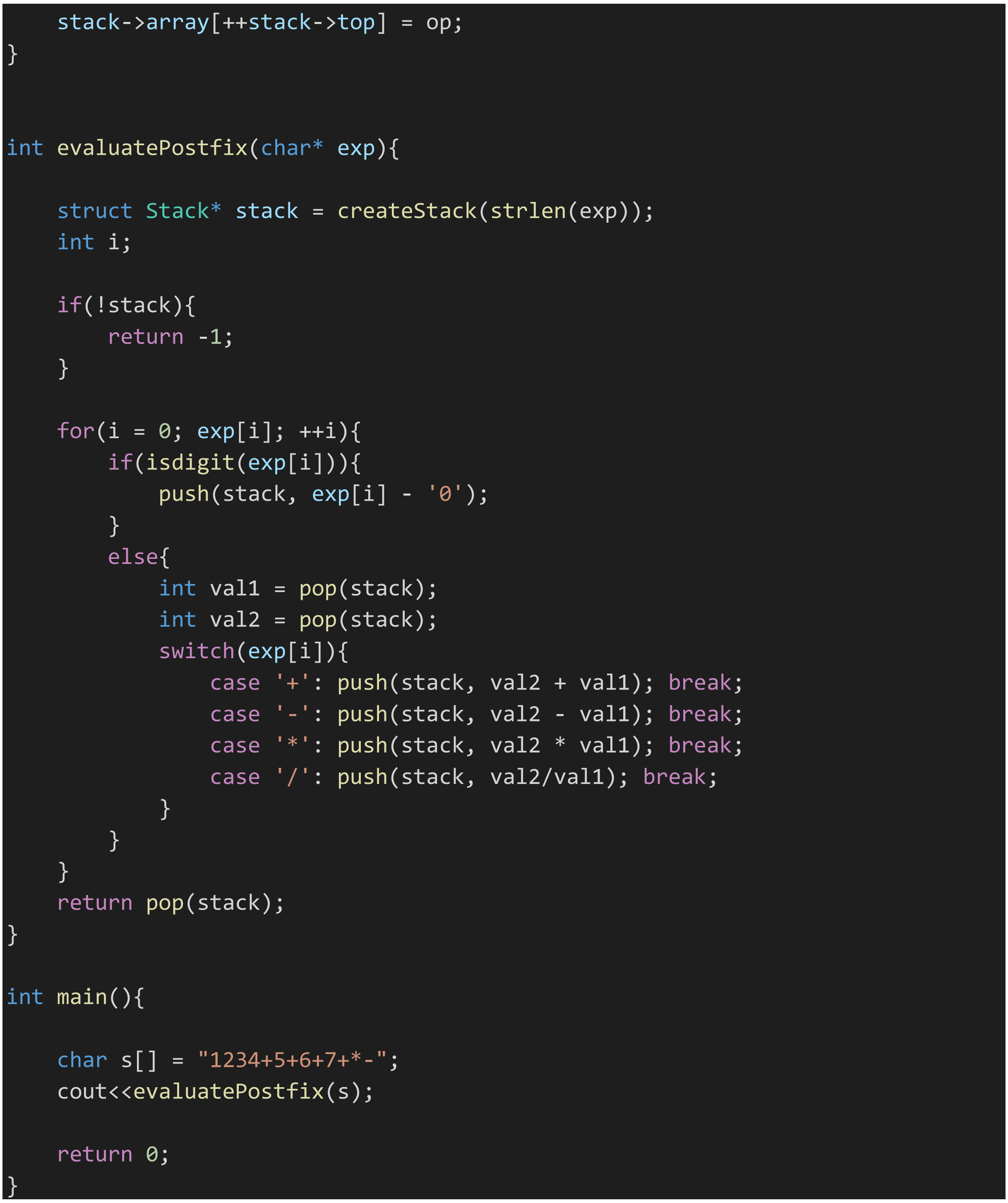
Output :

xyab+c+d+*-

Postfix Evaluation



```
#include <iostream>
#include <string.h>

using namespace std;

struct Stack{
    int top;
    unsigned capacity;
    int* array;
};

struct Stack* createStack( unsigned capacity ){
    struct Stack* stack = (struct Stack*) malloc(sizeof(struct Stack));

    if(!stack){
        return NULL;
    }

    stack->top = -1;
    stack->capacity = capacity;
    stack->array = (int*) malloc(stack->capacity * sizeof(int));

    if(!stack->array){
        return NULL;
    }

    return stack;
}

int isEmpty(struct Stack* stack){
    return stack->top == -1 ;
}

char peek(struct Stack* stack){
    return stack->array[stack->top];
}

char pop(struct Stack* stack){
    if(!isEmpty(stack)){
        return stack->array[stack->top--];
    }
    return '$';
}

void push(struct Stack* stack, char op){
```



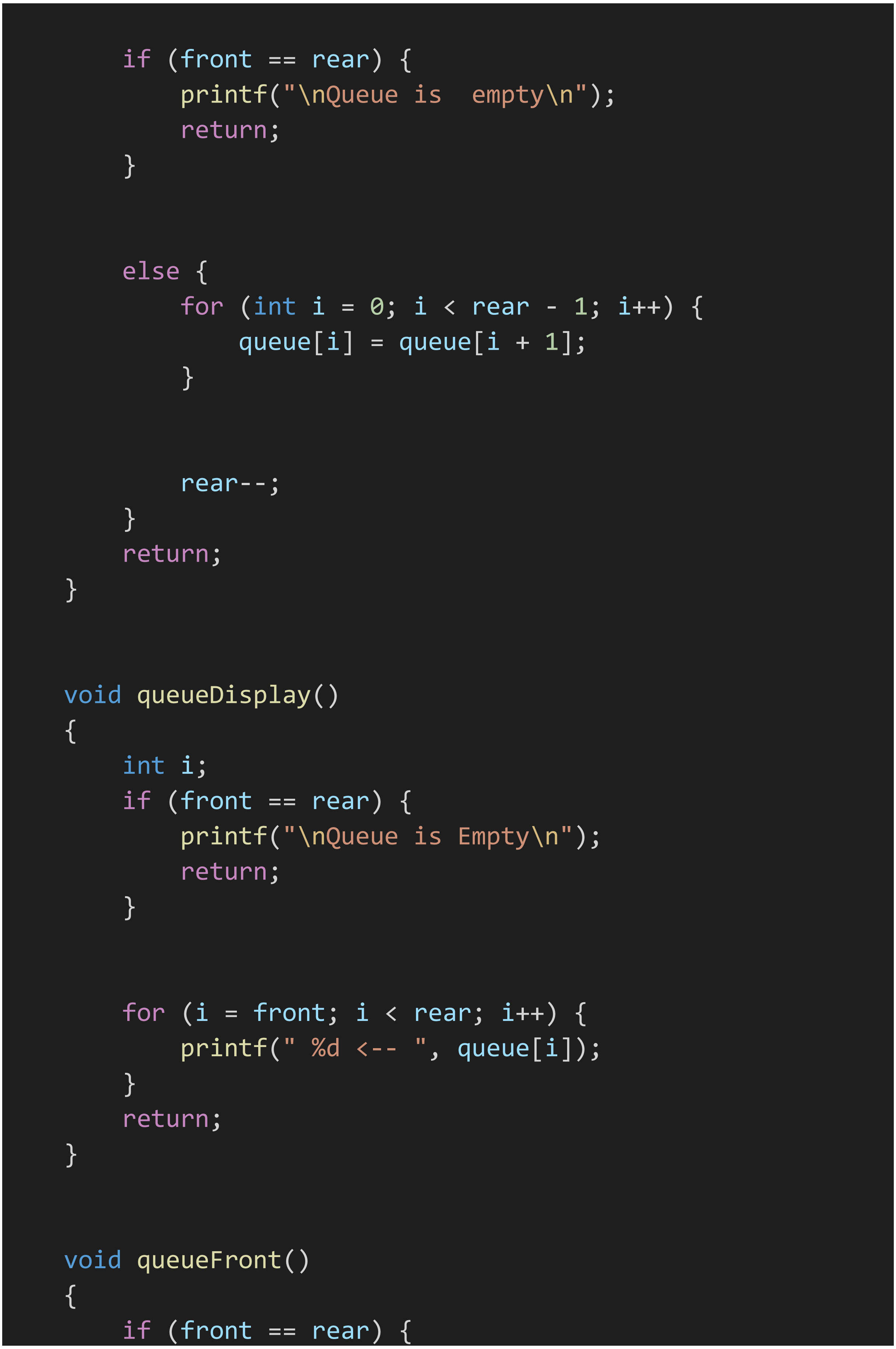
```
    stack->array[++stack->top] = op;
}


int evaluatePostfix(char* exp){

    struct Stack* stack = createStack(strlen(exp));
    int i;

    if(!stack){
        return -1;
    }

    for(i = 0; exp[i]; ++i){
        if(isdigit(exp[i])){
            push(stack, exp[i] - '0');
        }
        else{
            int val1 = pop(stack);
            int val2 = pop(stack);
            switch(exp[i]){
                case '+': push(stack, val2 + val1); break;
                case '-': push(stack, val2 - val1); break;
                case '*': push(stack, val2 * val1); break;
                case '/': push(stack, val2/val1); break;
            }
        }
    }
    return pop(stack);
}

int main(){

    char s[] = "1234+5+6+7+*-";
    cout<<evaluatePostfix(s);

    return 0;
}
```

Output :

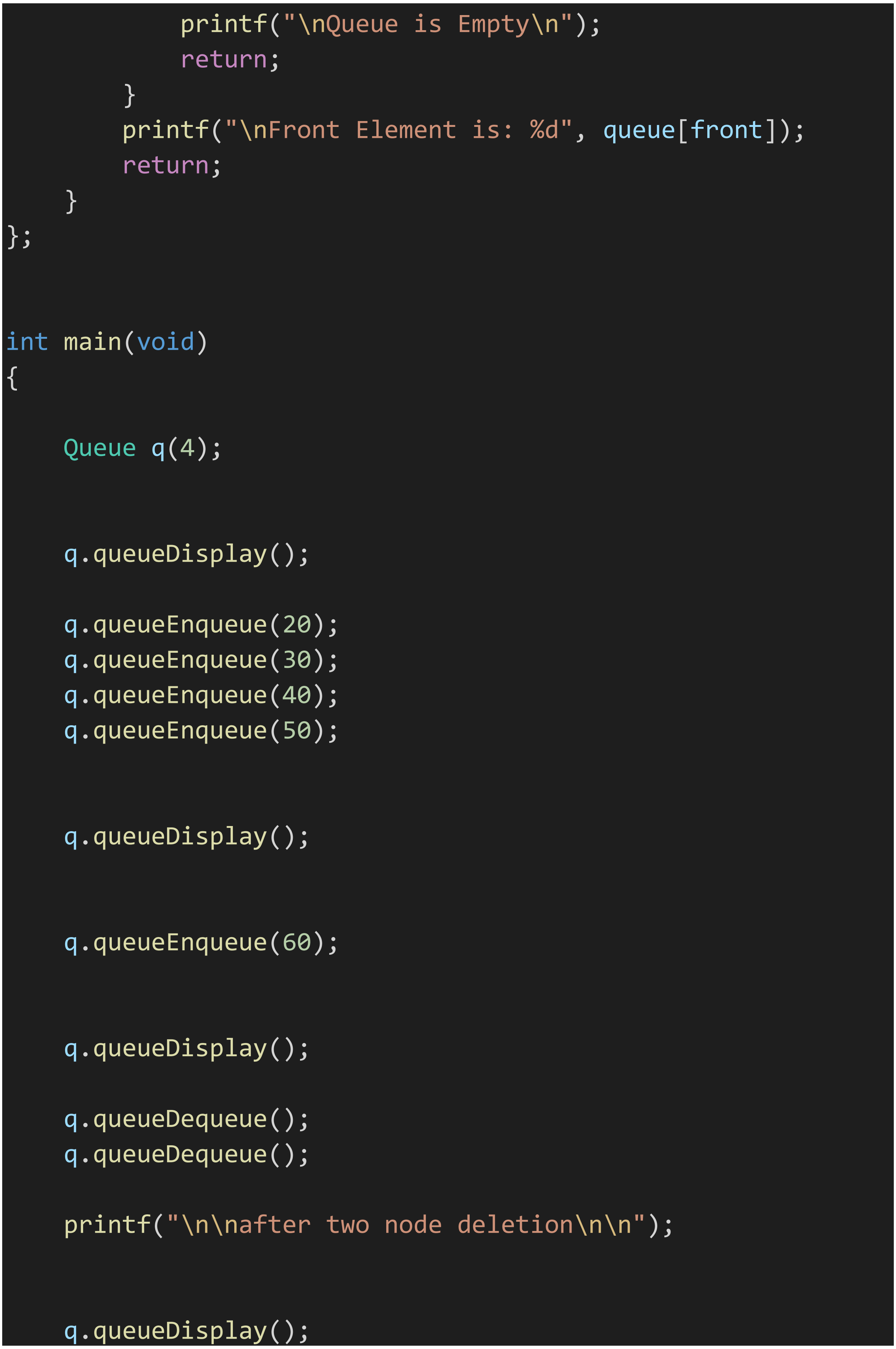
-49

Queue with Array

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

struct Queue {
    int front, rear, capacity;
    int* queue;
    Queue(int c)
    {
        front = rear = 0;
        capacity = c;
        queue = new int;
    }

    ~Queue() { delete[] queue; }


    void queueEnqueue(int data)
    {

        if (capacity == rear) {
            printf("\nQueue is full\n");
            return;
        }


        else {
            queue[rear] = data;
            rear++;
        }
        return;
    }


    void queueDequeue()
    {
```



```
    if (front == rear) {
        printf("\nQueue is  empty\n");
        return;
    }


    else {
        for (int i = 0; i < rear - 1; i++) {
            queue[i] = queue[i + 1];
        }


        rear--;
    }
    return;
}


void queueDisplay()
{
    int i;
    if (front == rear) {
        printf("\nQueue is Empty\n");
        return;
    }


    for (i = front; i < rear; i++) {
        printf(" %d <-- ", queue[i]);
    }
    return;
}


void queueFront()
{
    if (front == rear) {
```



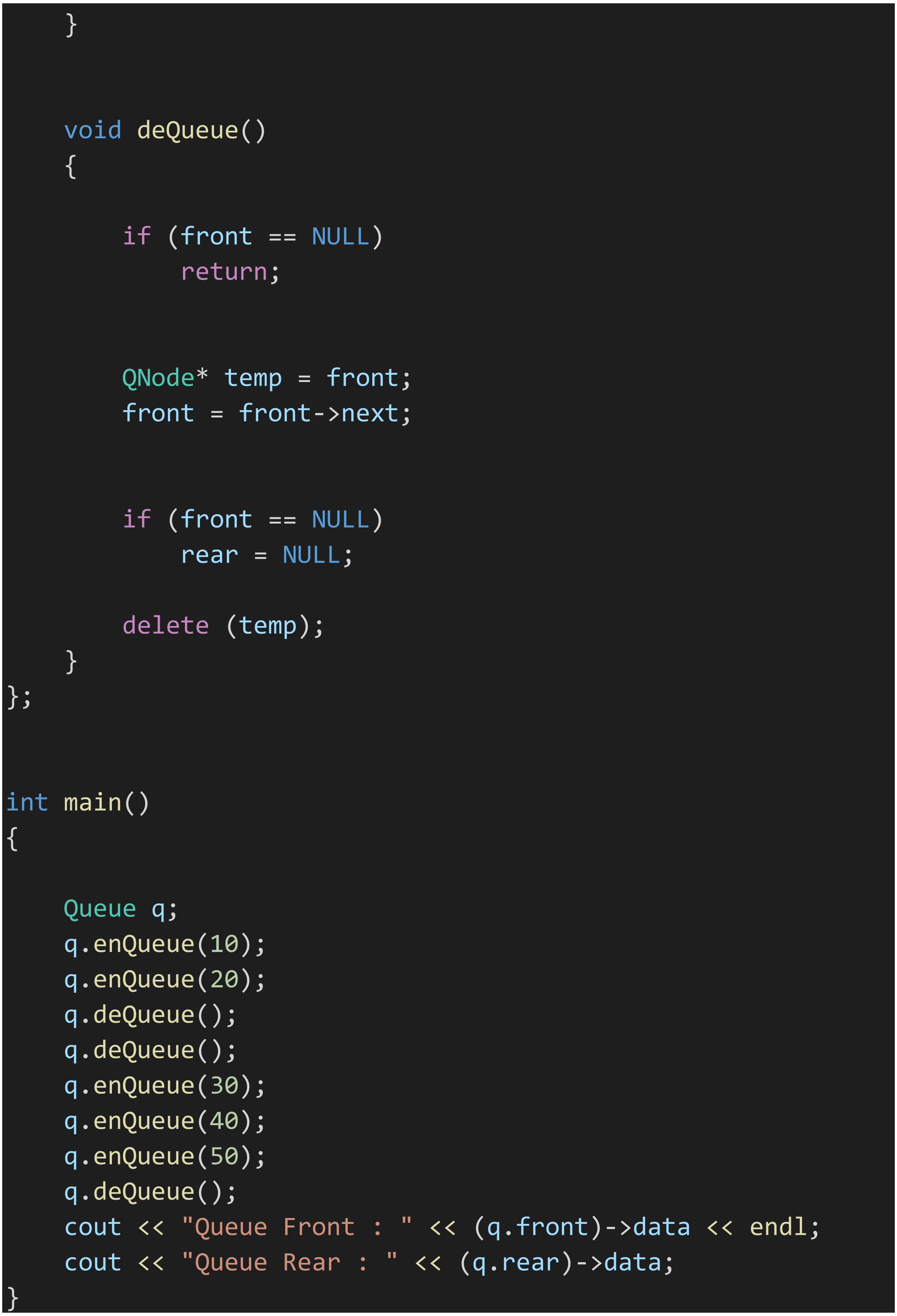
```
            printf("\nQueue is Empty\n");
            return;
        }
        printf("\nFront Element is: %d", queue[front]);
        return;
    }
};


int main(void)
{

    Queue q(4);


    q.queueDisplay();

    q.queueEnqueue(20);
    q.queueEnqueue(30);
    q.queueEnqueue(40);
    q.queueEnqueue(50);


    q.queueDisplay();


    q.queueEnqueue(60);


    q.queueDisplay();

    q.queueDequeue();
    q.queueDequeue();

    printf("\n\nafter two node deletion\n\n");


    q.queueDisplay();
    q.queueFront();

    return 0;
}
```

Output:

```
Queue is Empty
 20 <--  30 <--  40 <--  50 <--
Queue is full
 20 <--  30 <--  40 <--  50 <--

after two node deletion

 40 <--  50 <--
Front Element is: 40
```

Queue with Linked List

```cpp
#include <iostream>
using namespace std;

struct QNode {
    int data;
    QNode* next;
    QNode(int d)
    {
        data = d;
        next = NULL;
    }
};

struct Queue {
    QNode *front, *rear;
    Queue()
    {
        front = rear = NULL;
    }

    void enQueue(int x)
    {


        QNode* temp = new QNode(x);


        if (rear == NULL) {
            front = rear = temp;
            return;
        }


        rear->next = temp;
        rear = temp;
```



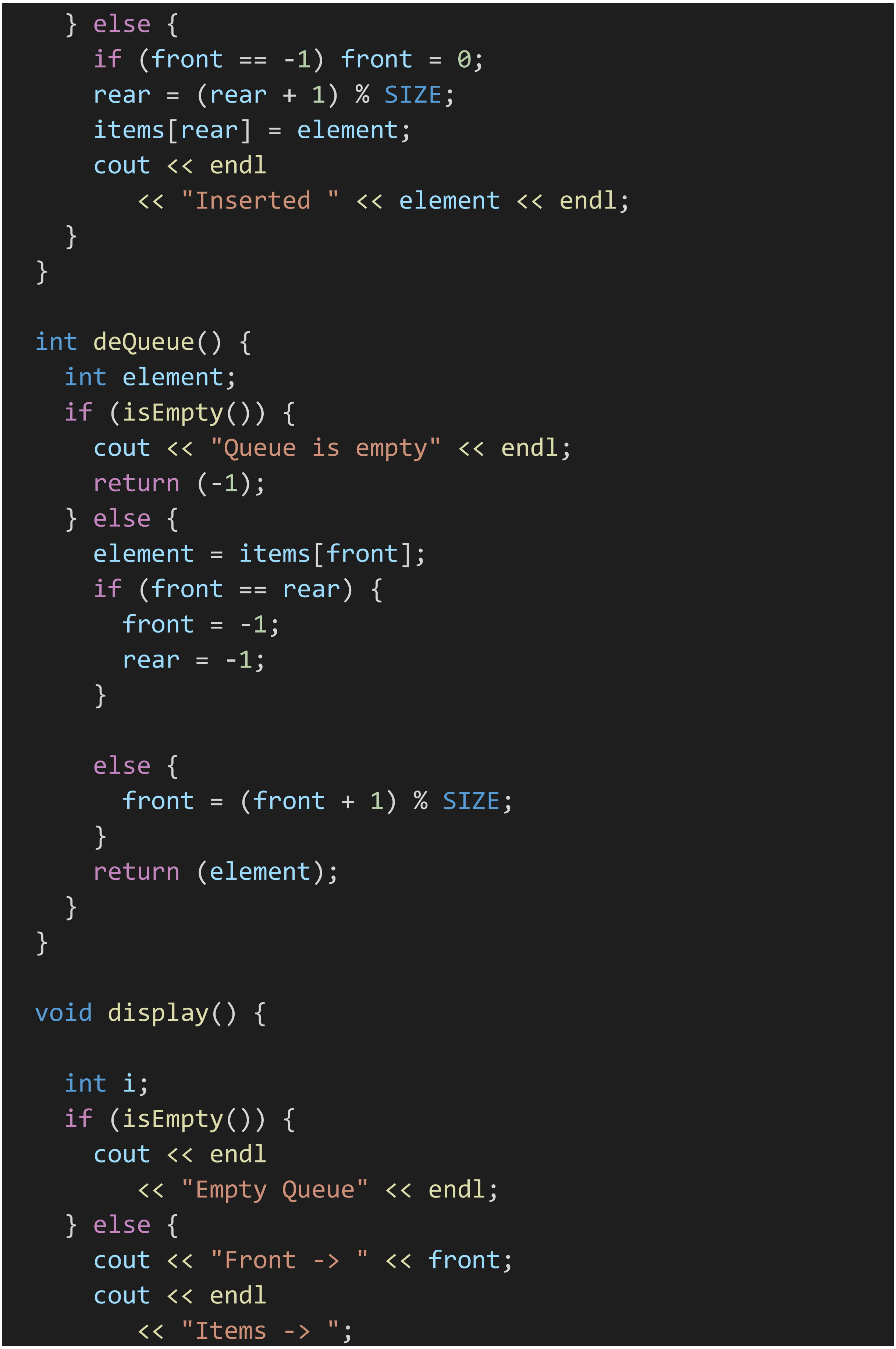
```cpp
    }


    void deQueue()
    {

        if (front == NULL)
            return;


        QNode* temp = front;
        front = front->next;


        if (front == NULL)
            rear = NULL;

        delete (temp);
    }
};


int main()
{

    Queue q;
    q.enQueue(10);
    q.enQueue(20);
    q.deQueue();
    q.deQueue();
    q.enQueue(30);
    q.enQueue(40);
    q.enQueue(50);
    q.deQueue();
    cout << "Queue Front : " << (q.front)->data << endl;
    cout << "Queue Rear : " << (q.rear)->data;
}
```

Output:

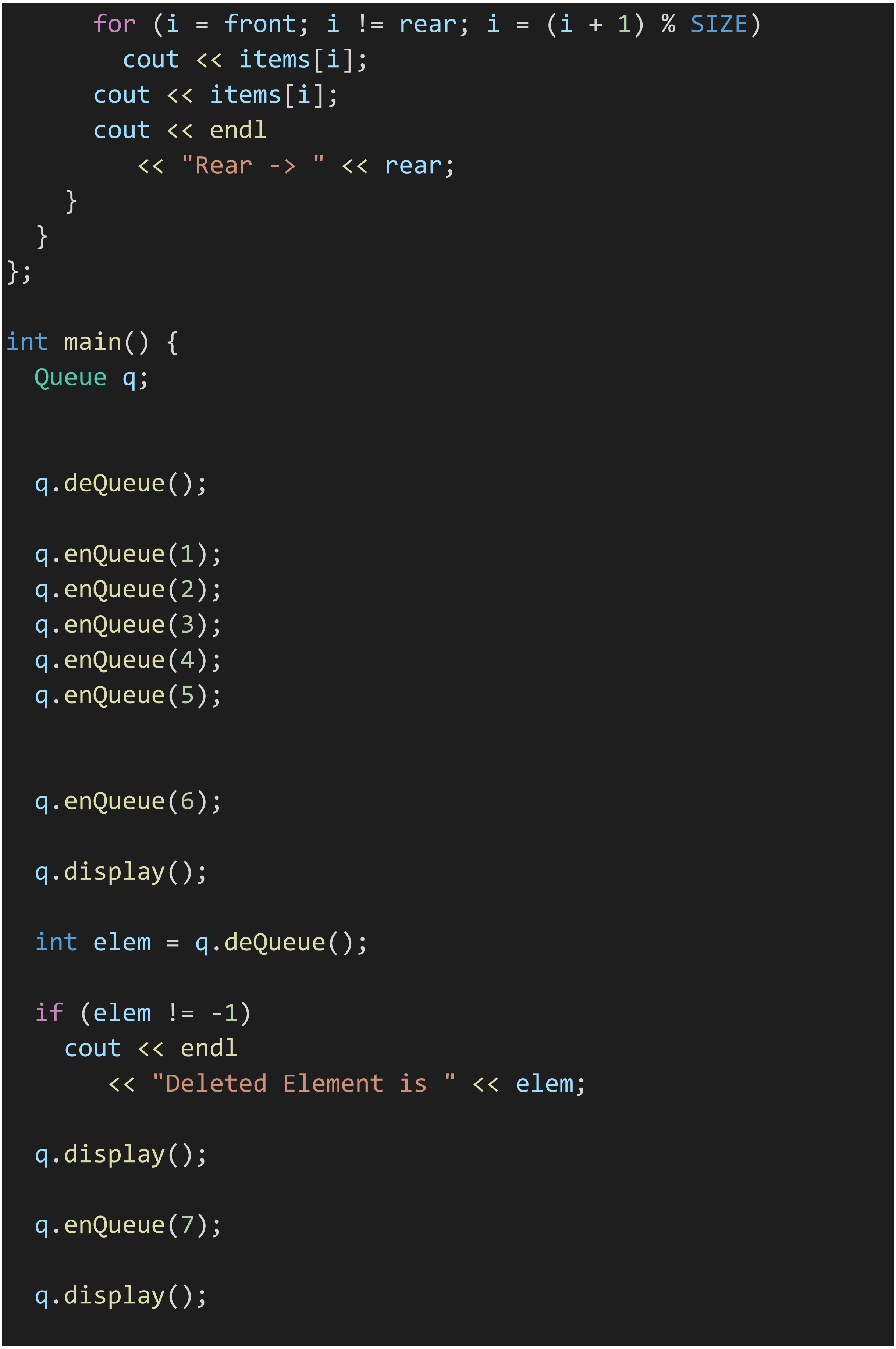
```
Queue Front : 40

Queue Rear : 50
```

Circular Queue with Array

```
#include <iostream>
#define SIZE 5

using namespace std;

class Queue {
   private:
  int items[SIZE], front, rear;

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

  bool isFull() {
    if (front == 0 && rear == SIZE - 1) {
      return true;
    }
    if (front == rear + 1) {
      return true;
    }
    return false;
  }

  bool isEmpty() {
    if (front == -1)
      return true;
    else
      return false;
  }

  void enQueue(int element) {
    if (isFull()) {
      cout << "Queue is full";
```



```
  } else {
    if (front == -1) front = 0;
    rear = (rear + 1) % SIZE;
    items[rear] = element;
    cout << endl
       << "Inserted " << element << endl;
  }
}

int deQueue() {
  int element;
  if (isEmpty()) {
    cout << "Queue is empty" << endl;
    return (-1);
  } else {
    element = items[front];
    if (front == rear) {
      front = -1;
      rear = -1;
    }

    else {
      front = (front + 1) % SIZE;
    }
    return (element);
  }
}

void display() {

  int i;
  if (isEmpty()) {
    cout << endl
       << "Empty Queue" << endl;
  } else {
    cout << "Front -> " << front;
    cout << endl
       << "Items -> ";
```



```
      for (i = front; i != rear; i = (i + 1) % SIZE)
        cout << items[i];
      cout << items[i];
      cout << endl
         << "Rear -> " << rear;
    }
  }
};

int main() {
  Queue q;


  q.deQueue();

  q.enQueue(1);
  q.enQueue(2);
  q.enQueue(3);
  q.enQueue(4);
  q.enQueue(5);


  q.enQueue(6);

  q.display();

  int elem = q.deQueue();

  if (elem != -1)
    cout << endl
       << "Deleted Element is " << elem;

  q.display();

  q.enQueue(7);

  q.display();
  q.enQueue(8);

  return 0;
}
```

Output:

Queue is empty

Inserted 1

Inserted 2

Inserted 3

Inserted 4

Inserted 5

Queue is fullFront -> 0

Items -> 12345

Rear -> 4

Deleted Element is 1Front -> 1

Items -> 2345

Rear -> 4

Inserted 7

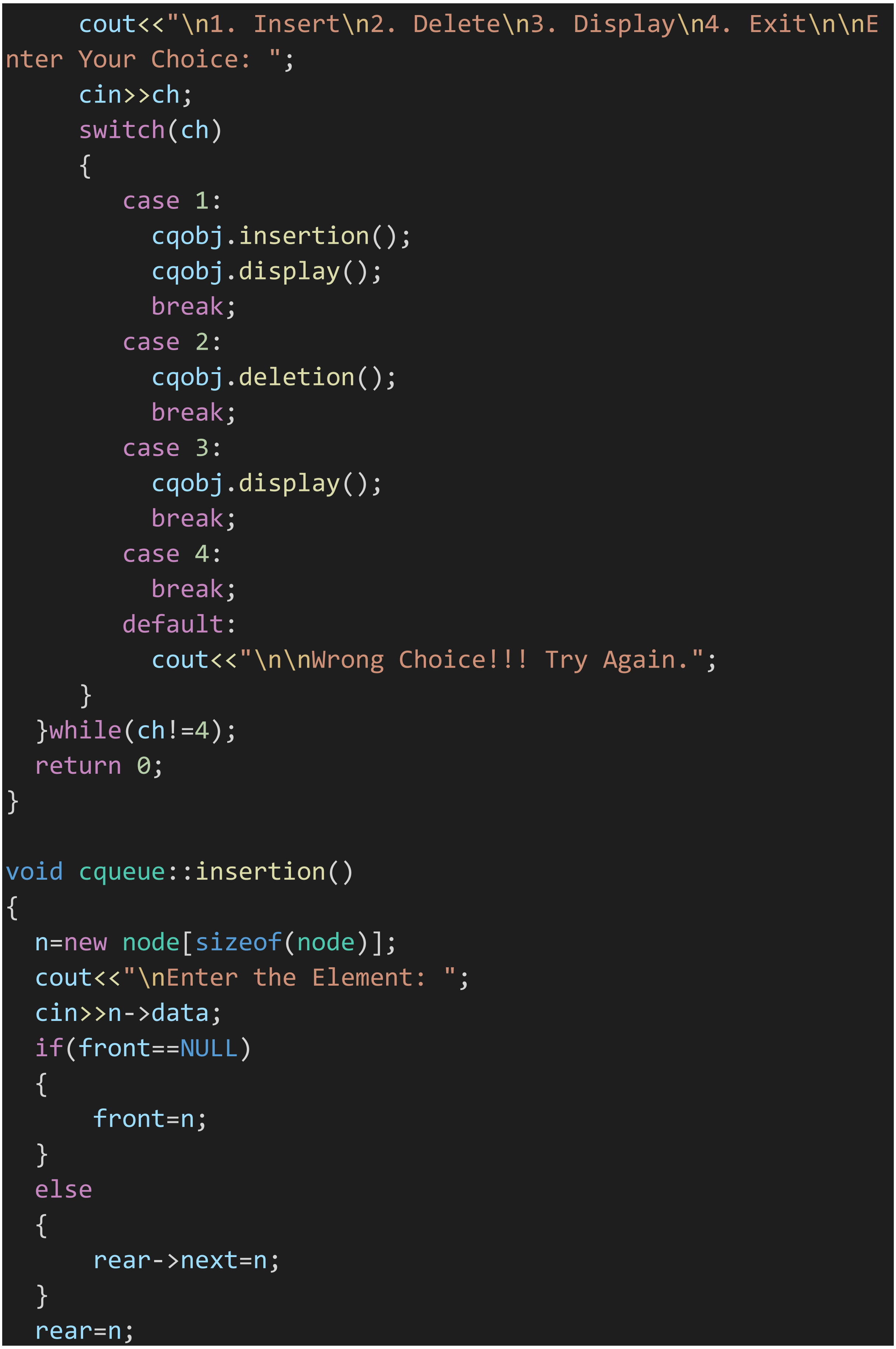
Front -> 1

Items -> 23457

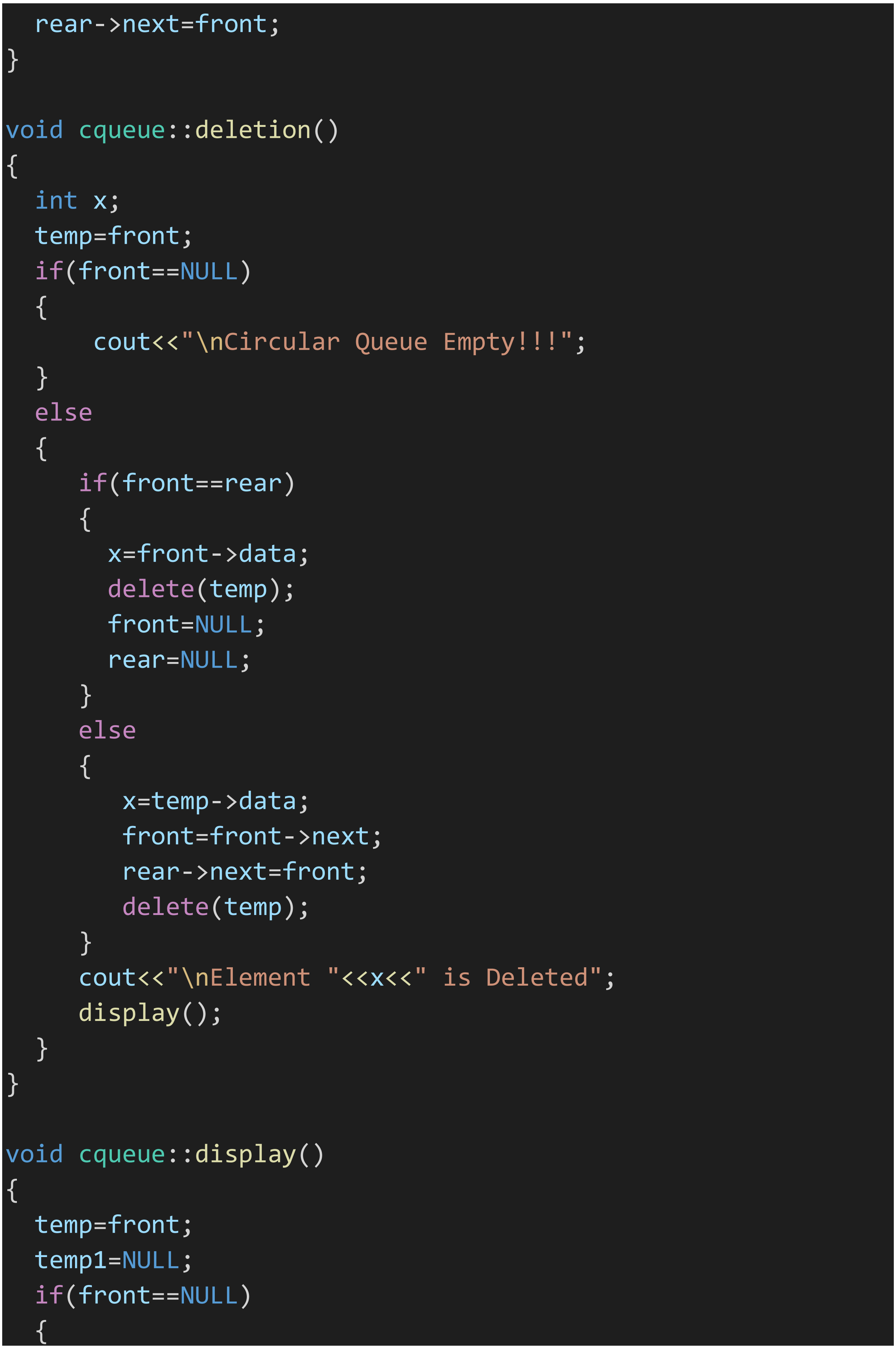
Rear -> 0Queue is full

Circular Queue with Linked List

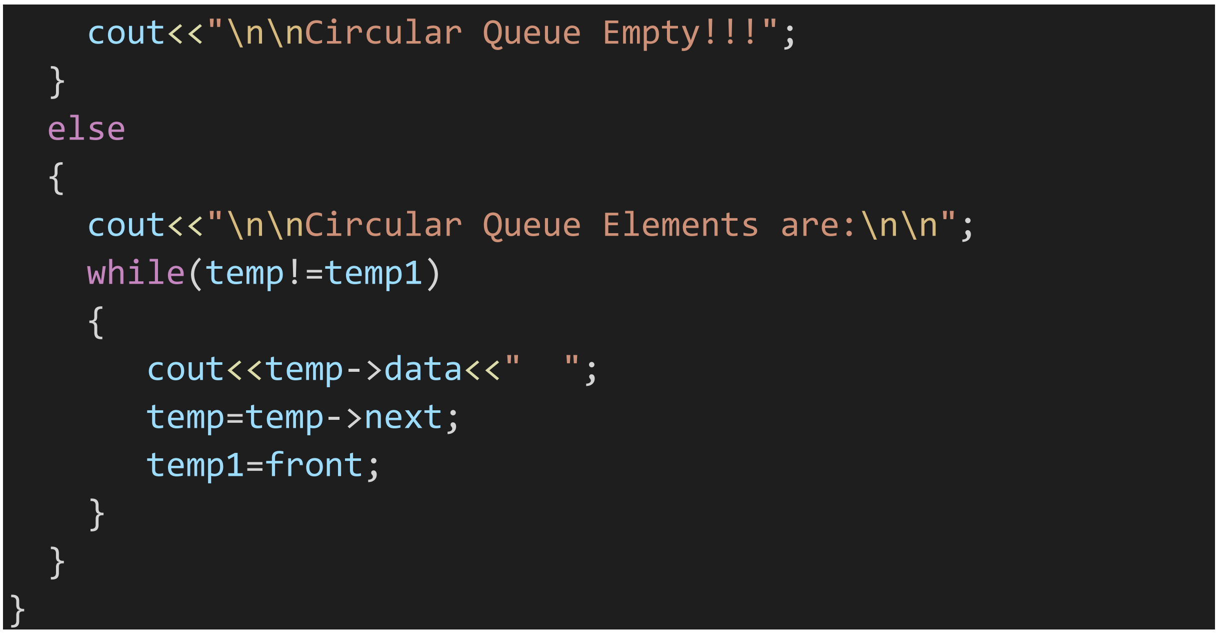
```
#include<iostream>

#define SIZE 100

using namespace std;

class node
{
public:
    node()
    {
        next = NULL;
    }
  int data;
  node *next;
}*front=NULL,*rear=NULL,*n,*temp,*temp1;

class cqueue
{
public:
    void insertion();
    void deletion();
    void display();
};

int main()
{
    cqueue cqobj;
  int ch;
  do
  {
     cout<<"\n\n\tMain Menu";
     cout<<"\n###########################";
```



```
     cout<<"\n1. Insert\n2. Delete\n3. Display\n4. Exit\n\nEnter Your Choice: ";
     cin>>ch;
     switch(ch)
     {
        case 1:
          cqobj.insertion();
          cqobj.display();
          break;
        case 2:
          cqobj.deletion();
          break;
        case 3:
          cqobj.display();
          break;
        case 4:
          break;
        default:
          cout<<"\n\nWrong Choice!!! Try Again.";
     }
  }while(ch!=4);
  return 0;
}

void cqueue::insertion()
{
  n=new node[sizeof(node)];
  cout<<"\nEnter the Element: ";
  cin>>n->data;
  if(front==NULL)
  {
      front=n;
  }
  else
  {
      rear->next=n;
  }
  rear=n;
```



```
  rear->next=front;
}

void cqueue::deletion()
{
  int x;
  temp=front;
  if(front==NULL)
  {
      cout<<"\nCircular Queue Empty!!!";
  }
  else
  {
     if(front==rear)
     {
       x=front->data;
       delete(temp);
       front=NULL;
       rear=NULL;
     }
     else
     {
        x=temp->data;
        front=front->next;
        rear->next=front;
        delete(temp);
     }
     cout<<"\nElement "<<x<<" is Deleted";
     display();
  }
}

void cqueue::display()
{
  temp=front;
  temp1=NULL;
  if(front==NULL)
  {
```



```
    cout<<"\n\nCircular Queue Empty!!!";
  }
  else
  {
    cout<<"\n\nCircular Queue Elements are:\n\n";
    while(temp!=temp1)
    {
       cout<<temp->data<<"  ";
       temp=temp->next;
       temp1=front;
    }
  }
}
```

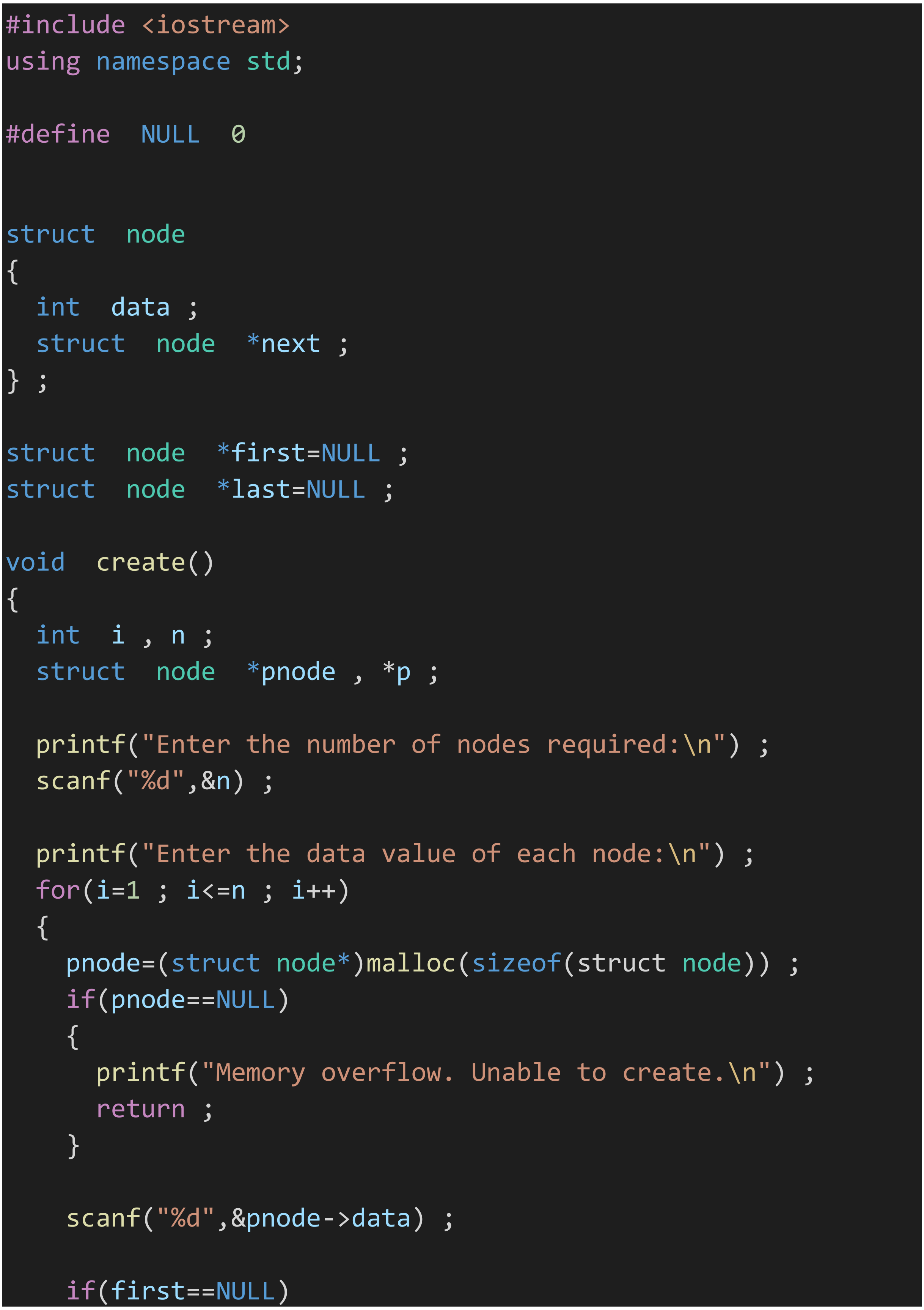
Output:

```
  Main Menu
##########################
1. Insert
2. Delete
3. Display
4. Exit

Enter Your Choice: 1

Enter the Element: 25



Circular Queue Elements are:


25
```

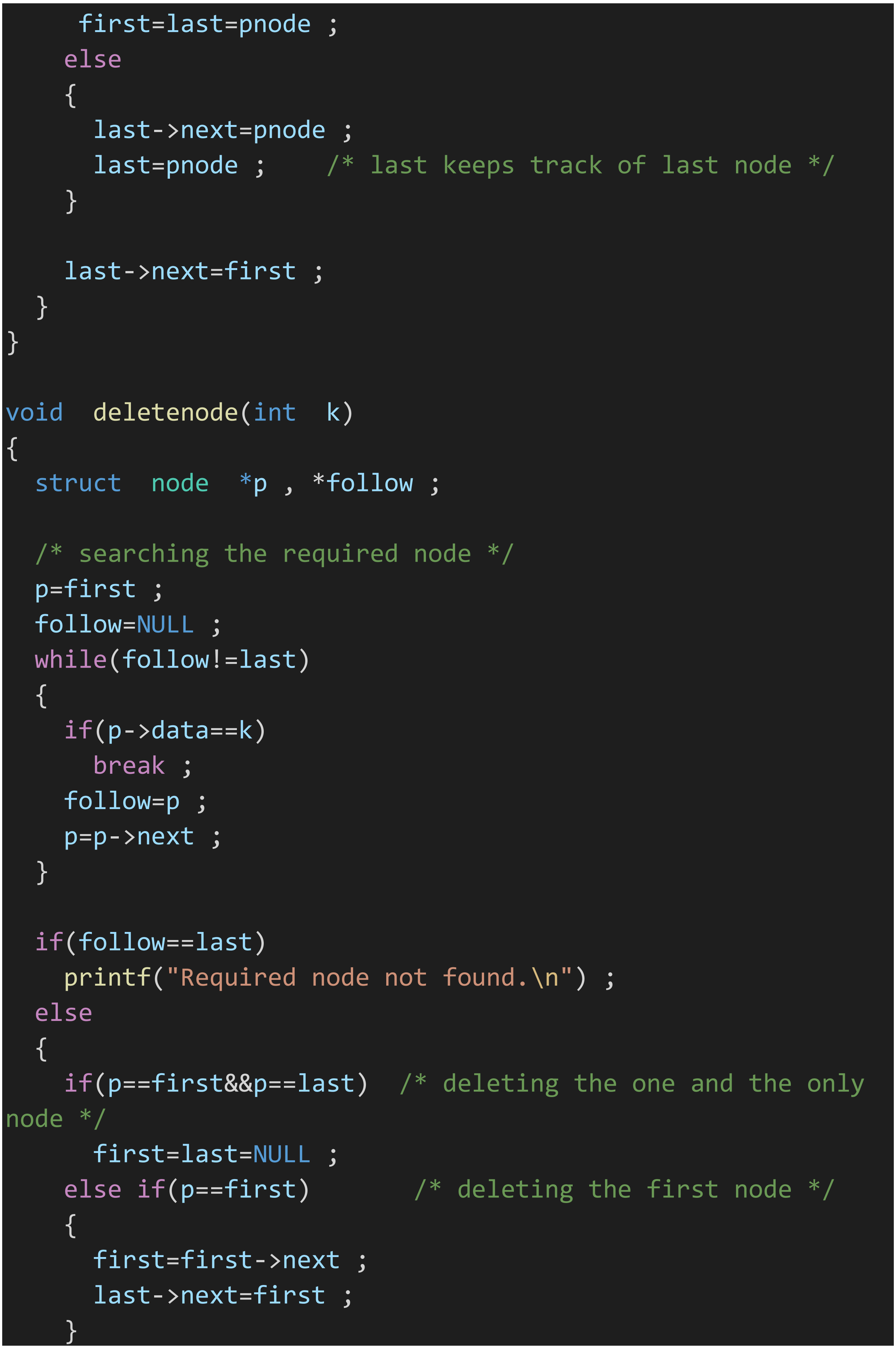
```
        Main Menu
##########################
1. Insert
2. Delete
3. Display
4. Exit

Enter Your Choice: 1

Enter the Element: 70


Circular Queue Elements are:

25  70

        Main Menu
##########################
1. Insert
2. Delete
3. Display
4. Exit

Enter Your Choice: 2
```

```
Element 25 is Deleted

Circular Queue Elements are:

70

        Main Menu
##########################
1. Insert
2. Delete
3. Display
4. Exit

Enter Your Choice: 4
```

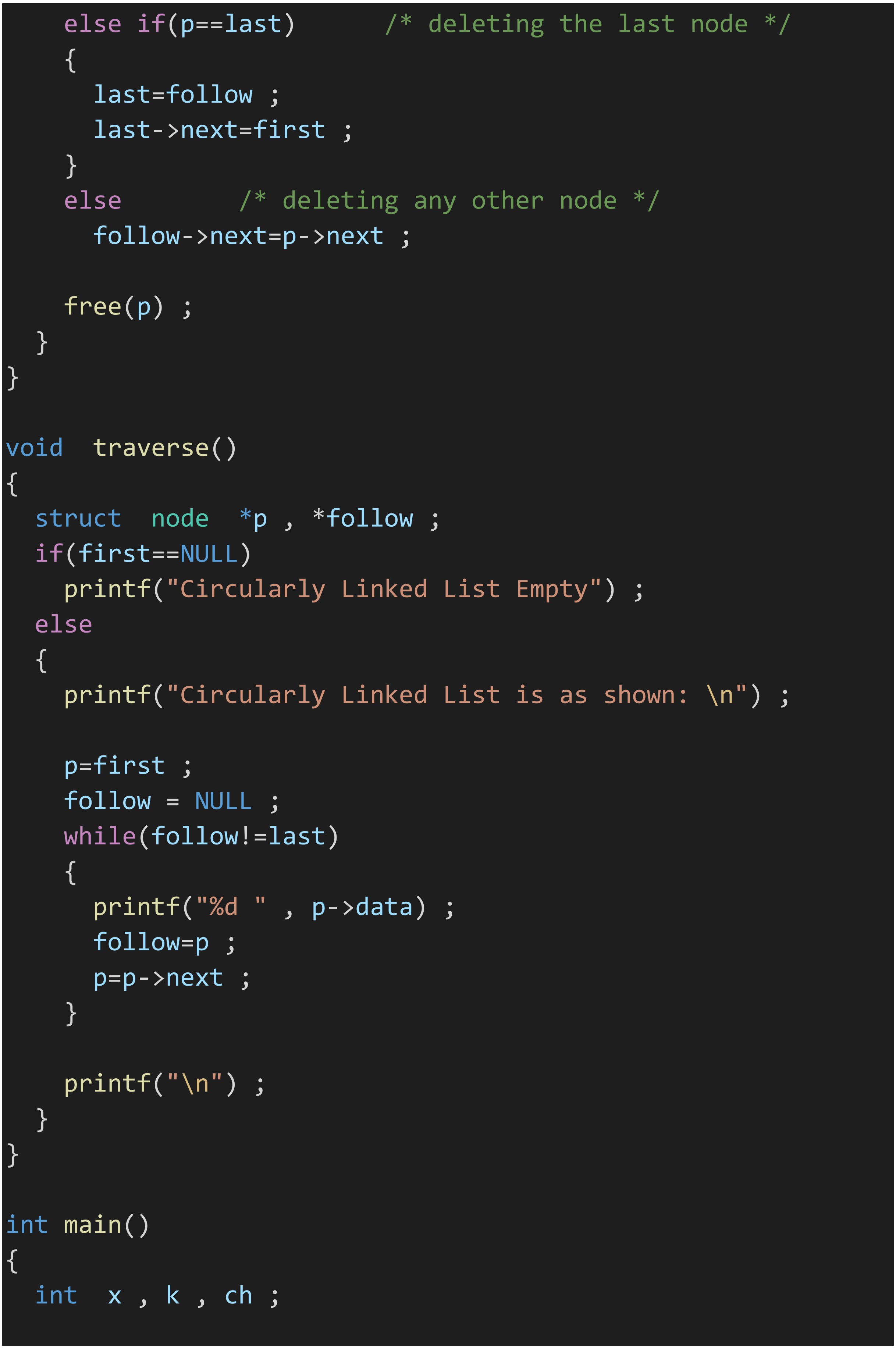
Circular Linked List(All Insertions, All Deletions & Display)



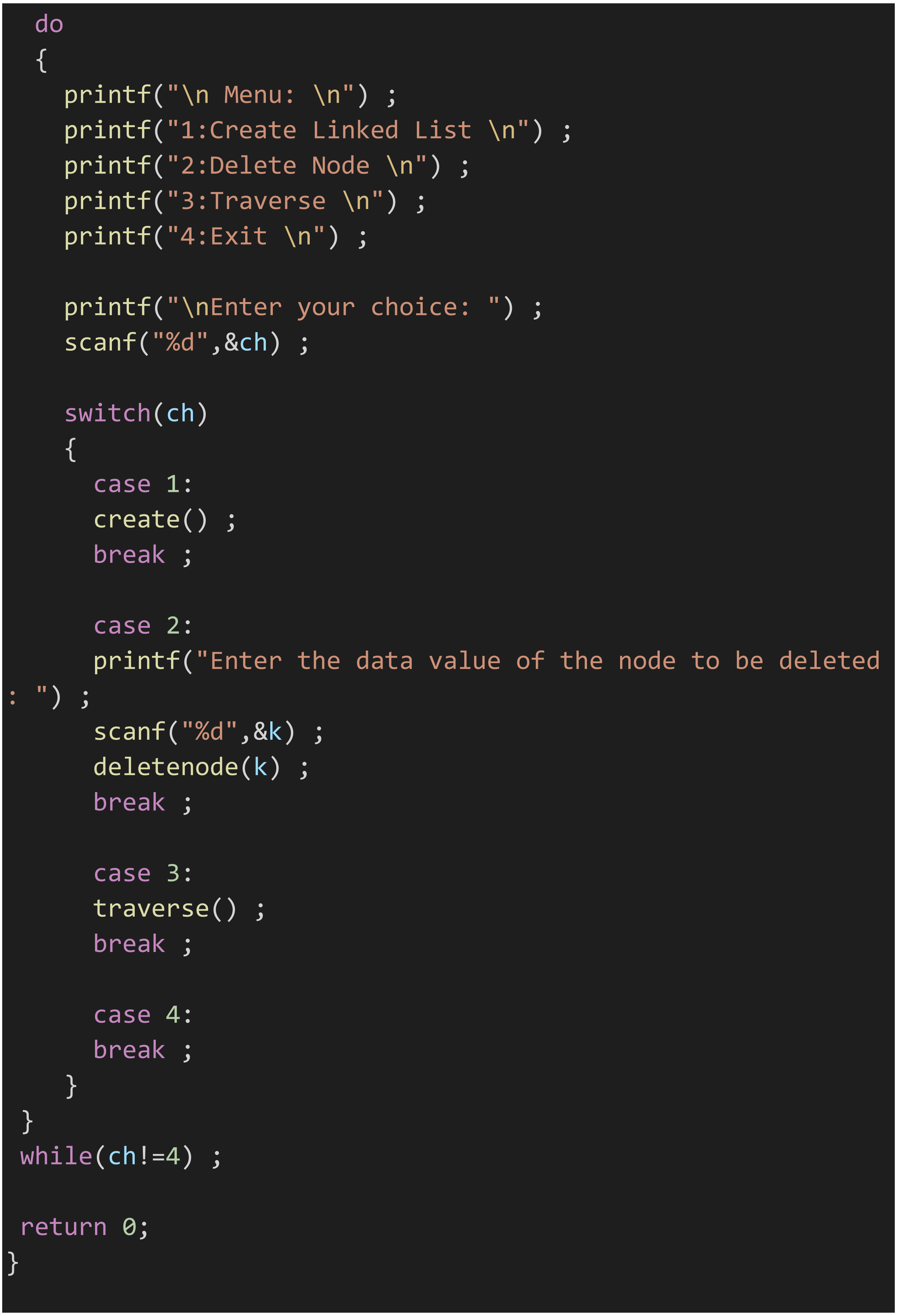
```
#include <iostream>
using namespace std;

#define  NULL  0


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

struct  node  *first=NULL ;
struct  node  *last=NULL ;

void  create()
{
  int  i , n ;
  struct  node  *pnode , *p ;

  printf("Enter the number of nodes required:\n") ;
  scanf("%d",&n) ;

  printf("Enter the data value of each node:\n") ;
  for(i=1 ; i<=n ; i++)
  {
    pnode=(struct node*)malloc(sizeof(struct node)) ;
    if(pnode==NULL)
    {
      printf("Memory overflow. Unable to create.\n") ;
      return ;
    }

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

    if(first==NULL)
```



```
     first=last=pnode ;
    else
    {
      last->next=pnode ;
      last=pnode ;    /* last keeps track of last node */
    }

    last->next=first ;
  }
}

void  deletenode(int  k)
{
  struct  node  *p , *follow ;

  /* searching the required node */
  p=first ;
  follow=NULL ;
  while(follow!=last)
  {
    if(p->data==k)
      break ;
    follow=p ;
    p=p->next ;
  }

  if(follow==last)
    printf("Required node not found.\n") ;
  else
  {
    if(p==first&&p==last)  /* deleting the one and the only
node */
      first=last=NULL ;
    else if(p==first)       /* deleting the first node */
    {
      first=first->next ;
      last->next=first ;
    }
```



```
    else if(p==last)      /* deleting the last node */
    {
      last=follow ;
      last->next=first ;
    }
    else        /* deleting any other node */
      follow->next=p->next ;

    free(p) ;
  }
}

void  traverse()
{
  struct  node  *p , *follow ;
  if(first==NULL)
    printf("Circularly Linked List Empty") ;
  else
  {
    printf("Circularly Linked List is as shown: \n") ;

    p=first ;
    follow = NULL ;
    while(follow!=last)
    {
      printf("%d " , p->data) ;
      follow=p ;
      p=p->next ;
    }

    printf("\n") ;
  }
}

int main()
{
  int  x , k , ch ;
```



```
  do
  {
    printf("\n Menu: \n") ;
    printf("1:Create Linked List \n") ;
    printf("2:Delete Node \n") ;
    printf("3:Traverse \n") ;
    printf("4:Exit \n") ;

    printf("\nEnter your choice: ") ;
    scanf("%d",&ch) ;

    switch(ch)
    {
      case 1:
      create() ;
      break ;

      case 2:
      printf("Enter the data value of the node to be deleted
: ") ;
      scanf("%d",&k) ;
      deletenode(k) ;
      break ;

      case 3:
      traverse() ;
      break ;

      case 4:
      break ;
    }
 }
 while(ch!=4) ;

 return 0;
}
```

Output:

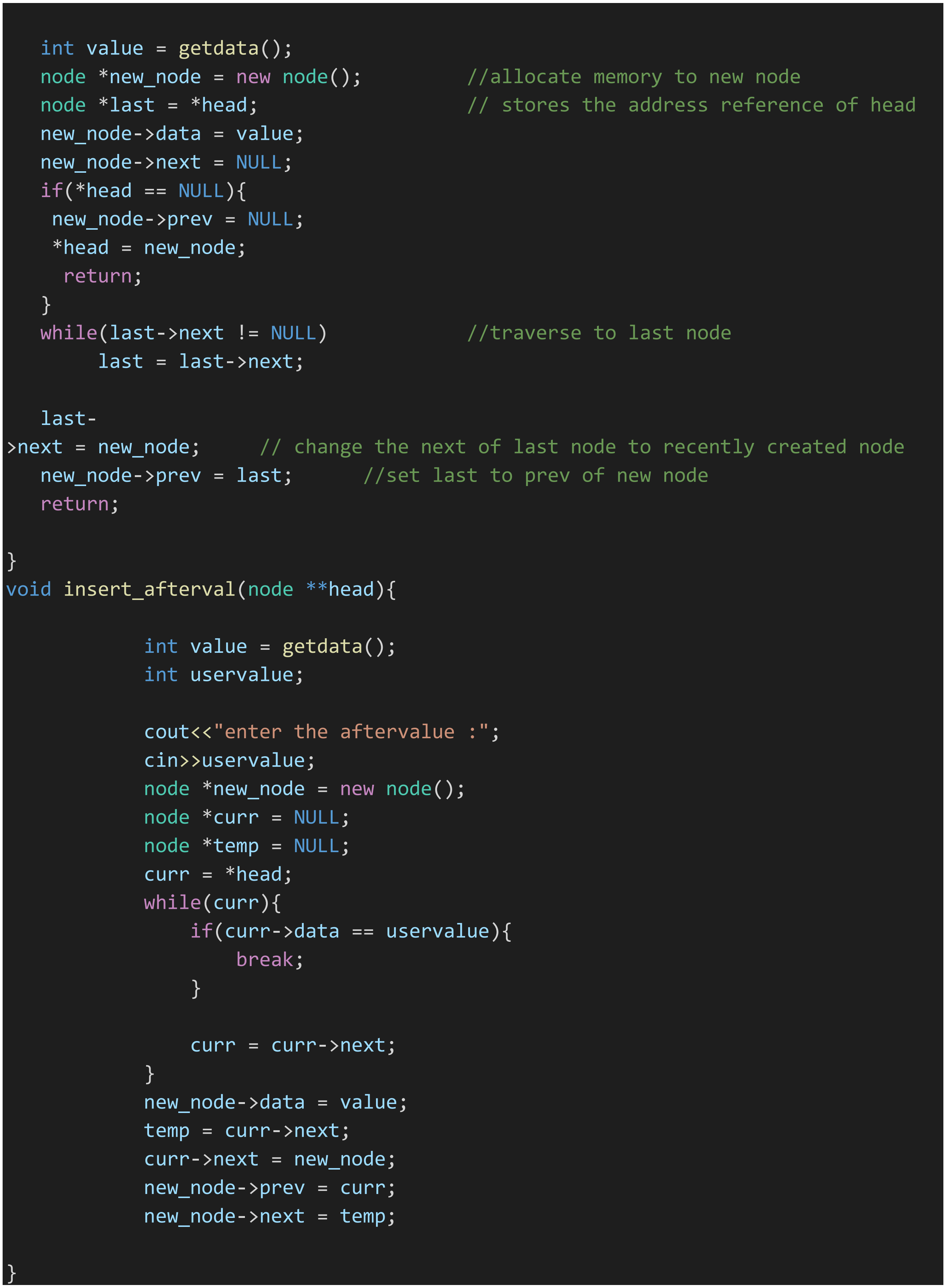
```
Menu:
1:Create Linked List
2:Delete Node
3:Traverse
4:Exit

Enter your choice: 1
Enter the number of nodes required:
6
Enter the data value of each node:
34
2
67
12
99
77

 Menu:
1:Create Linked List
2:Delete Node
3:Traverse
4:Exit

Enter your choice: 3
Circularly Linked List is as shown:
```

```
34 2 67 12 99 77

 Menu:
1:Create Linked List
2:Delete Node
3:Traverse
4:Exit

Enter your choice: 2
Enter the data value of the node to be deleted: 34

 Menu:
1:Create Linked List
2:Delete Node
3:Traverse
4:Exit

Enter your choice: 3
Circularly Linked List is as shown:
2 67 12 99 77

 Menu:
1:Create Linked List
2:Delete Node
3:Traverse
4:Exit
```

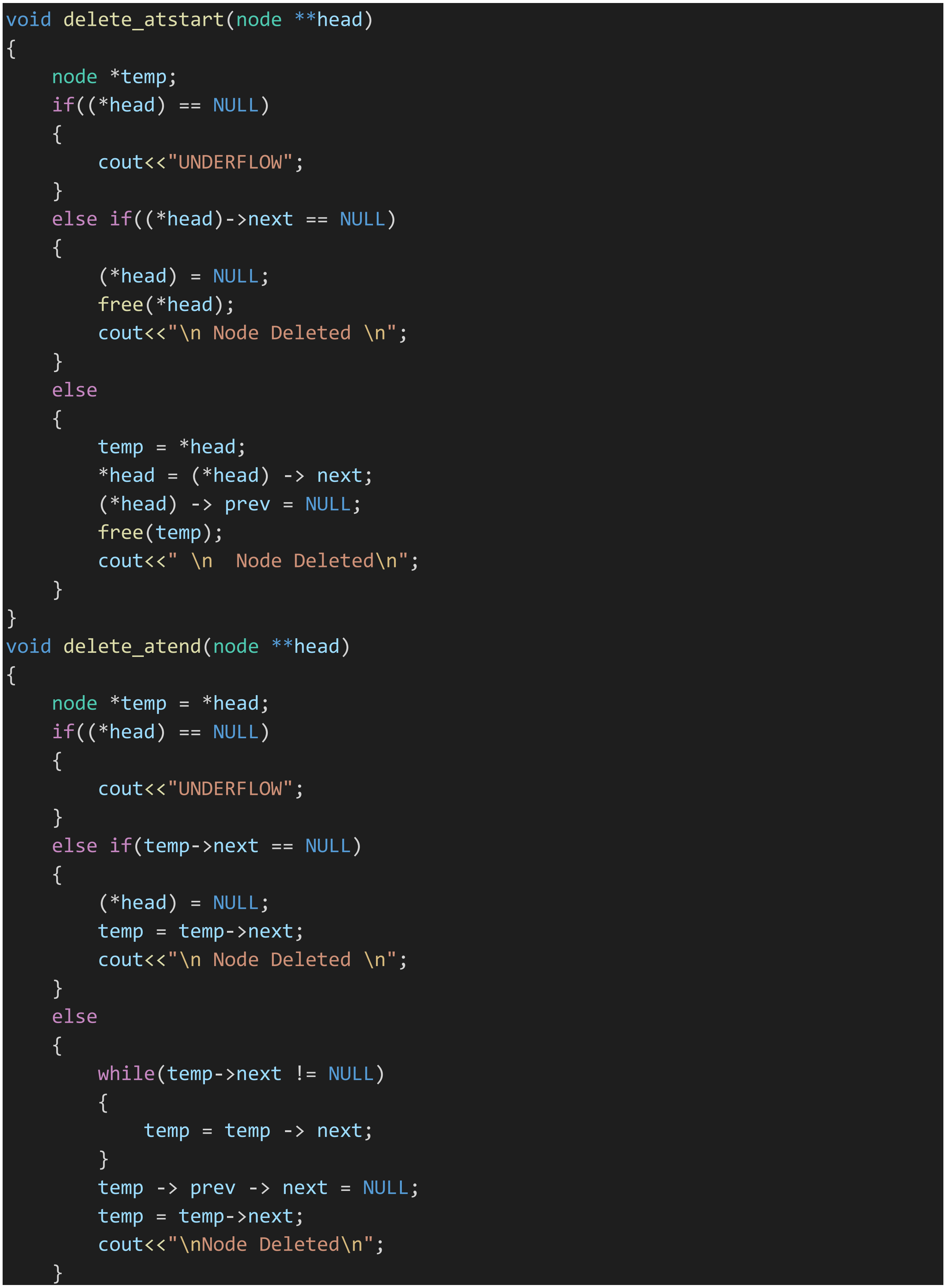
```
Enter your choice: 2
Enter the data value of the node to be deleted: 99

 Menu:
1:Create Linked List
2:Delete Node
3:Traverse
4:Exit

Enter your choice: 3
Circularly Linked List is as shown:
2 67 12 77

 Menu:
1:Create Linked List
2:Delete Node
3:Traverse
4:Exit

Enter your choice: 4
```

Doubly Link-list

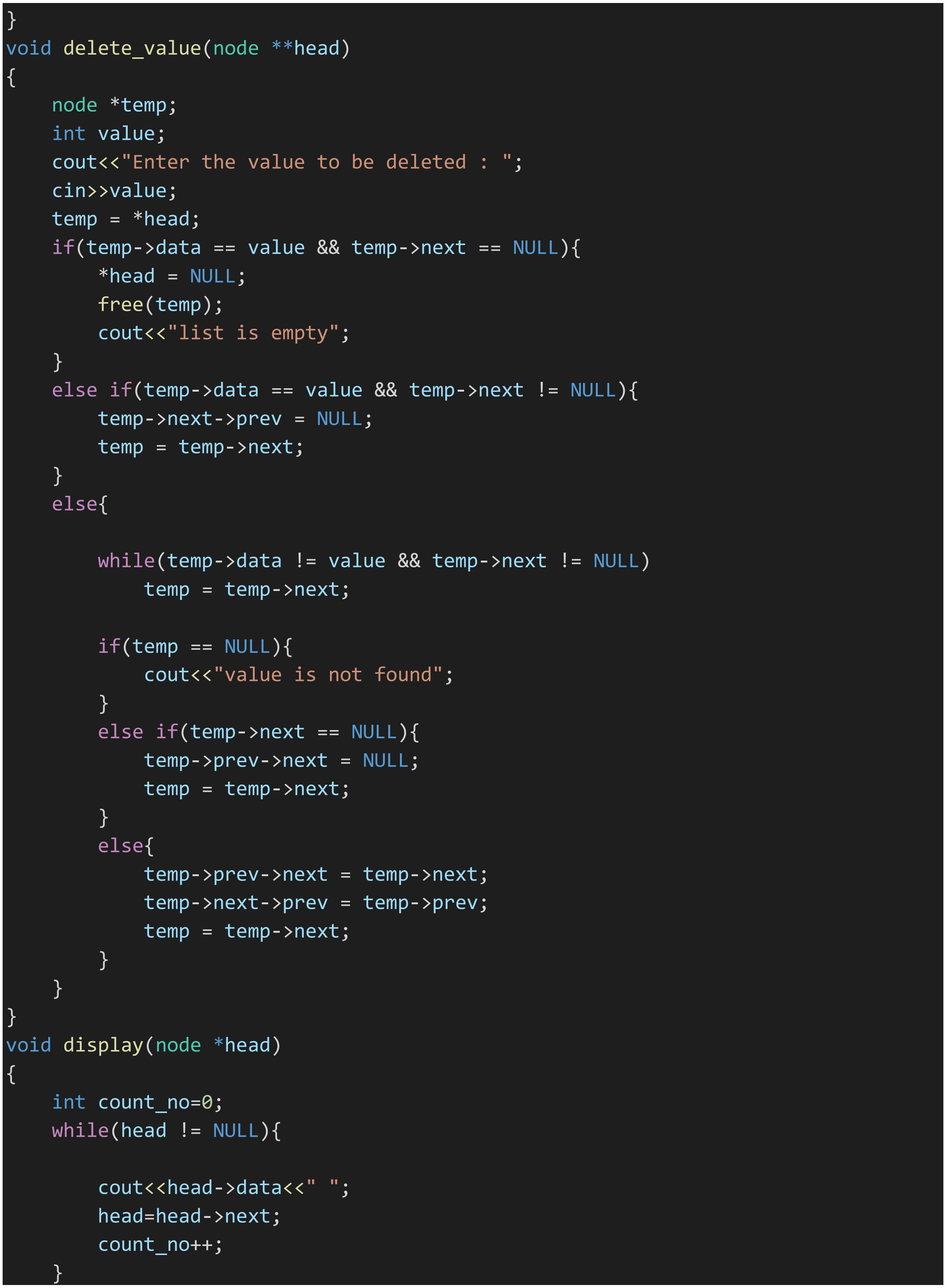
```
#include<iostream>
#include<stdlib.h>
#include<new>

using namespace std;

class node{

public:

    int data;
    node *next;
    node *prev;
};
 int getdata(){

     int value;
     cout<<"enter the value : ";
     cin>>value;
     return value;
 }
 void insert_atstart(node **head){

    int value = getdata();   //gets the value
    node *new_node=new node(); //allocates the memory to new node
    if(*head == NULL){
        new_node->next = NULL;
        new_node->prev = NULL;
        new_node->data = value;
        (*head)=new_node;
    }
    else{
         new_node->data = value;
        new_node->next = (*head);
        new_node->prev = NULL;
        (*head)->prev = new_node;
        (*head)=new_node;

    }


}
void insert_atend(node **head){
```



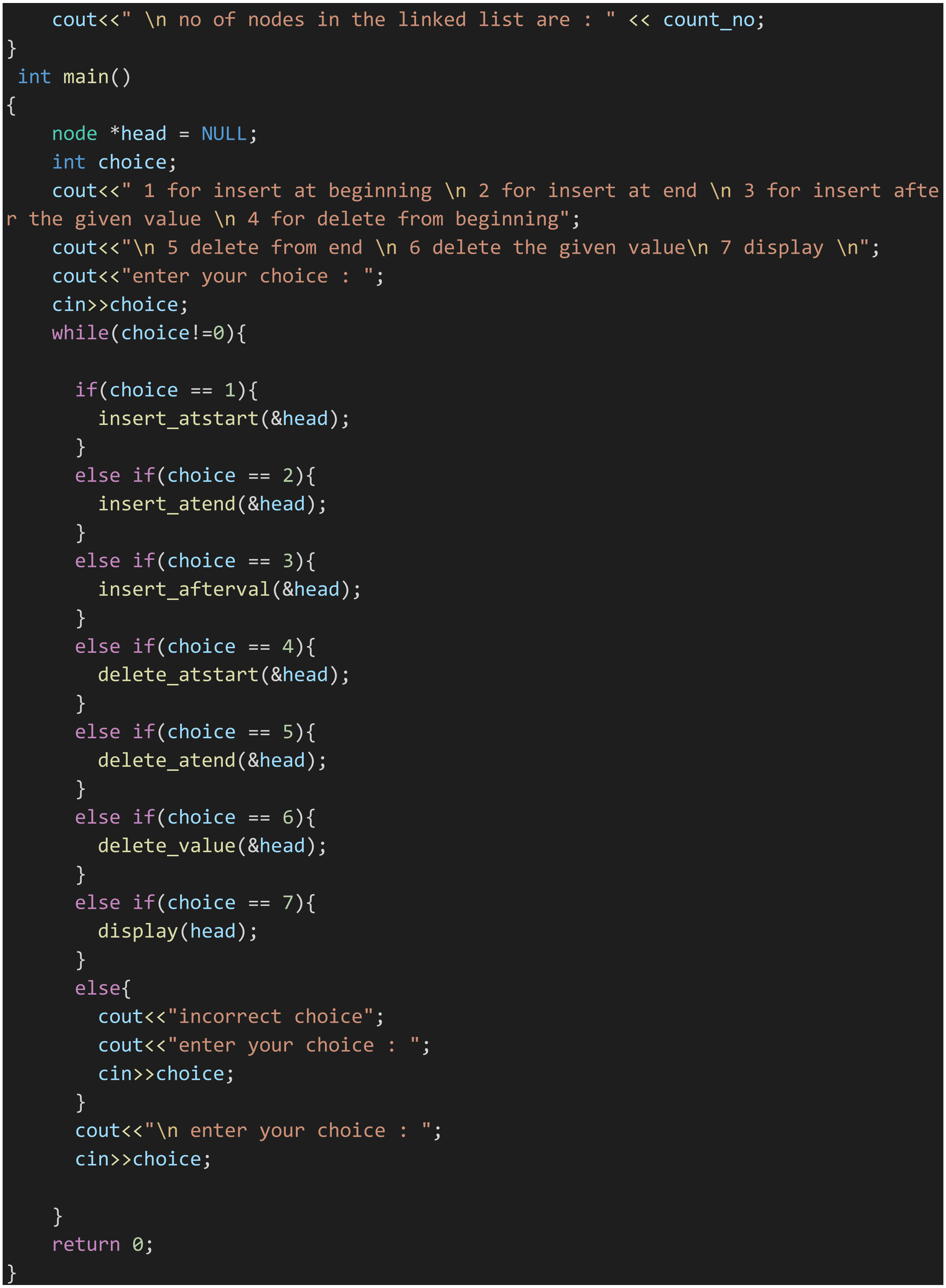
```
   int value = getdata();
   node *new_node = new node();         //allocate memory to new node
   node *last = *head;                  // stores the address reference of head
   new_node->data = value;
   new_node->next = NULL;
   if(*head == NULL){
    new_node->prev = NULL;
    *head = new_node;
     return;
   }
   while(last->next != NULL)            //traverse to last node
        last = last->next;

   last-
>next = new_node;     // change the next of last node to recently created node
   new_node->prev = last;      //set last to prev of new node
   return;

}
void insert_afterval(node **head){

            int value = getdata();
            int uservalue;

            cout<<"enter the aftervalue :";
            cin>>uservalue;
            node *new_node = new node();
            node *curr = NULL;
            node *temp = NULL;
            curr = *head;
            while(curr){
                if(curr->data == uservalue){
                    break;
                }

                curr = curr->next;
            }
            new_node->data = value;
            temp = curr->next;
            curr->next = new_node;
            new_node->prev = curr;
            new_node->next = temp;

}
```



```cpp
void delete_atstart(node **head)
{
    node *temp;
    if((*head) == NULL)
    {
        cout<<"UNDERFLOW";
    }
    else if((*head)->next == NULL)
    {
        (*head) = NULL;
        free(*head);
        cout<<"\n Node Deleted \n";
    }
    else
    {
        temp = *head;
        *head = (*head) -> next;
        (*head) -> prev = NULL;
        free(temp);
        cout<<" \n  Node Deleted\n";
    }
}
void delete_atend(node **head)
{
    node *temp = *head;
    if((*head) == NULL)
    {
        cout<<"UNDERFLOW";
    }
    else if(temp->next == NULL)
    {
        (*head) = NULL;
        temp = temp->next;
        cout<<"\n Node Deleted \n";
    }
    else
    {
        while(temp->next != NULL)
        {
            temp = temp -> next;
        }
        temp -> prev -> next = NULL;
        temp = temp->next;
        cout<<"\nNode Deleted\n";
    }
```



```
}
void delete_value(node **head)
{
    node *temp;
    int value;
    cout<<"Enter the value to be deleted : ";
    cin>>value;
    temp = *head;
    if(temp->data == value && temp->next == NULL){
        *head = NULL;
        free(temp);
        cout<<"list is empty";
    }
    else if(temp->data == value && temp->next != NULL){
        temp->next->prev = NULL;
        temp = temp->next;
    }
    else{

        while(temp->data != value && temp->next != NULL)
            temp = temp->next;

        if(temp == NULL){
            cout<<"value is not found";
        }
        else if(temp->next == NULL){
            temp->prev->next = NULL;
            temp = temp->next;
        }
        else{
            temp->prev->next = temp->next;
            temp->next->prev = temp->prev;
            temp = temp->next;
        }
    }
}
void display(node *head)
{
    int count_no=0;
    while(head != NULL){

        cout<<head->data<<" ";
        head=head->next;
        count_no++;
    }
```



```
    cout<<" \n no of nodes in the linked list are : " << count_no;
}
 int main()
{
    node *head = NULL;
    int choice;
    cout<<" 1 for insert at beginning \n 2 for insert at end \n 3 for insert after the given value \n 4 for delete from beginning";
    cout<<"\n 5 delete from end \n 6 delete the given value\n 7 display \n";
    cout<<"enter your choice : ";
    cin>>choice;
    while(choice!=0){

      if(choice == 1){
        insert_atstart(&head);
      }
      else if(choice == 2){
        insert_atend(&head);
      }
      else if(choice == 3){
        insert_afterval(&head);
      }
      else if(choice == 4){
        delete_atstart(&head);
      }
      else if(choice == 5){
        delete_atend(&head);
      }
      else if(choice == 6){
        delete_value(&head);
      }
      else if(choice == 7){
        display(head);
      }
      else{
        cout<<"incorrect choice";
        cout<<"enter your choice : ";
        cin>>choice;
      }
      cout<<"\n enter your choice : ";
      cin>>choice;

    }
    return 0;
}
```

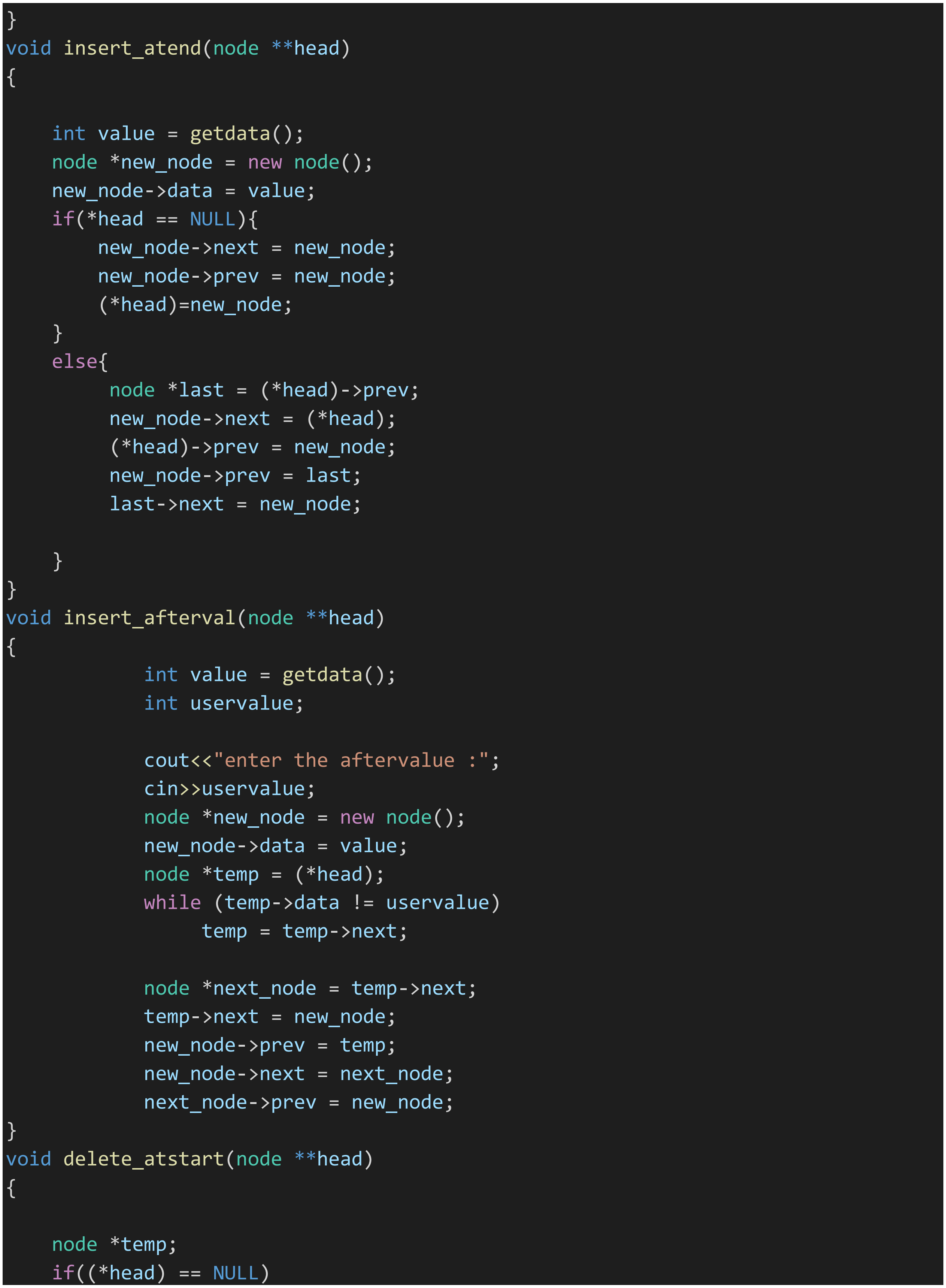
Output :

```
1 for insert at beginning
 2 for insert at end
 3 for insert after the given value
 4 for delete from beginning
 5 delete from end
 6 delete the given value
 7 display
enter your choice : 1
enter the value : 5

 enter your choice : 2
enter the value : 4

 enter your choice : 7
5 4
 no of nodes in the linked list are : 2
 enter your choice : 1
enter the value : 3

 enter your choice : 7
3 5 4
 no of nodes in the linked list are : 3
 enter your choice : 6
```

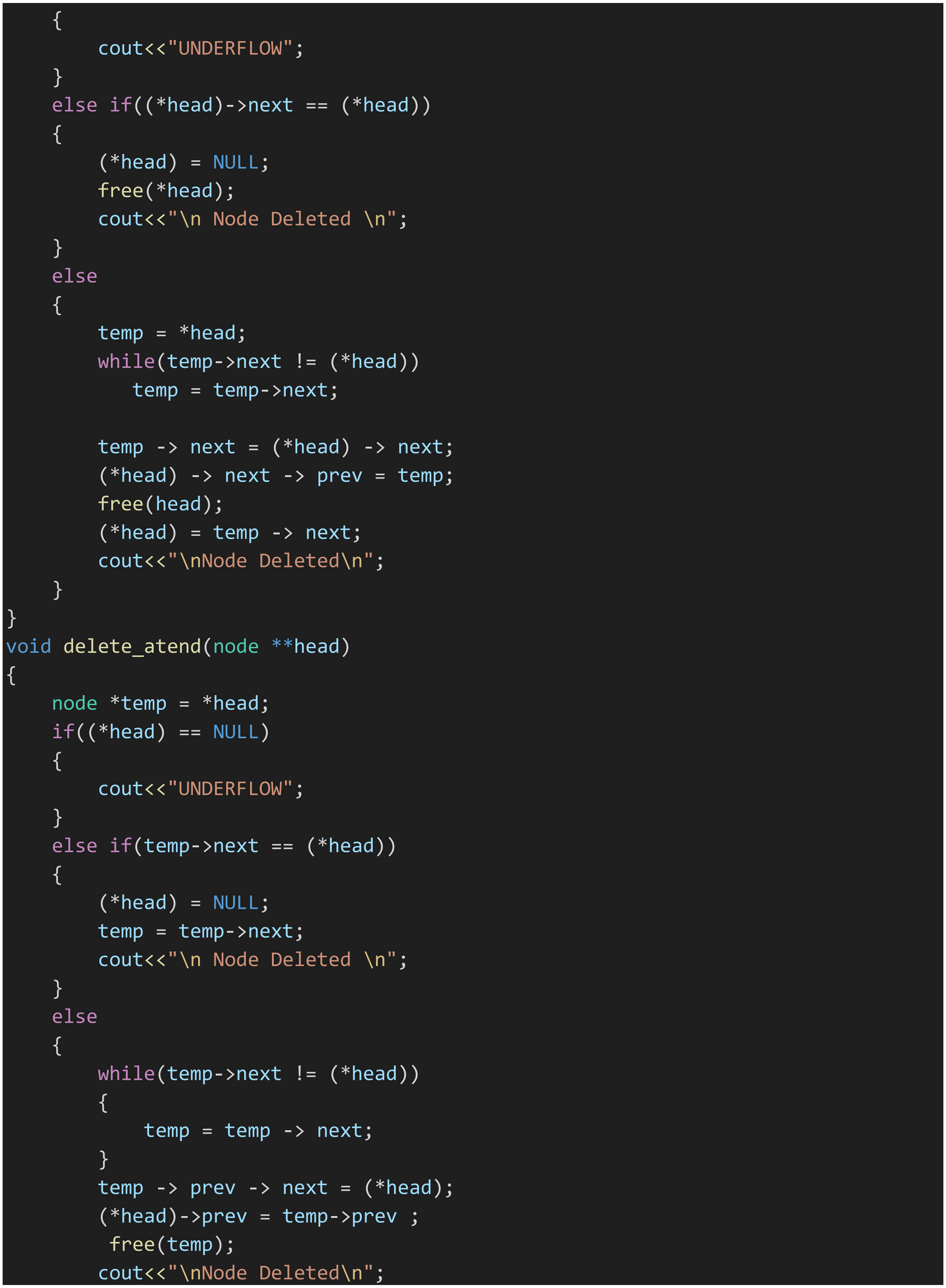
```
Enter the value to be deleted : 5


 enter your choice : 7

3 4

 no of nodes in the linked list are : 2

 enter your choice : 4


  Node Deleted


 enter your choice : 7

4

 no of nodes in the linked list are : 1

 enter your choice : 5


 Node Deleted


 enter your choice : 7


 no of nodes in the linked list are : 0

 enter your choice :
```

Doubly Circular Linklist

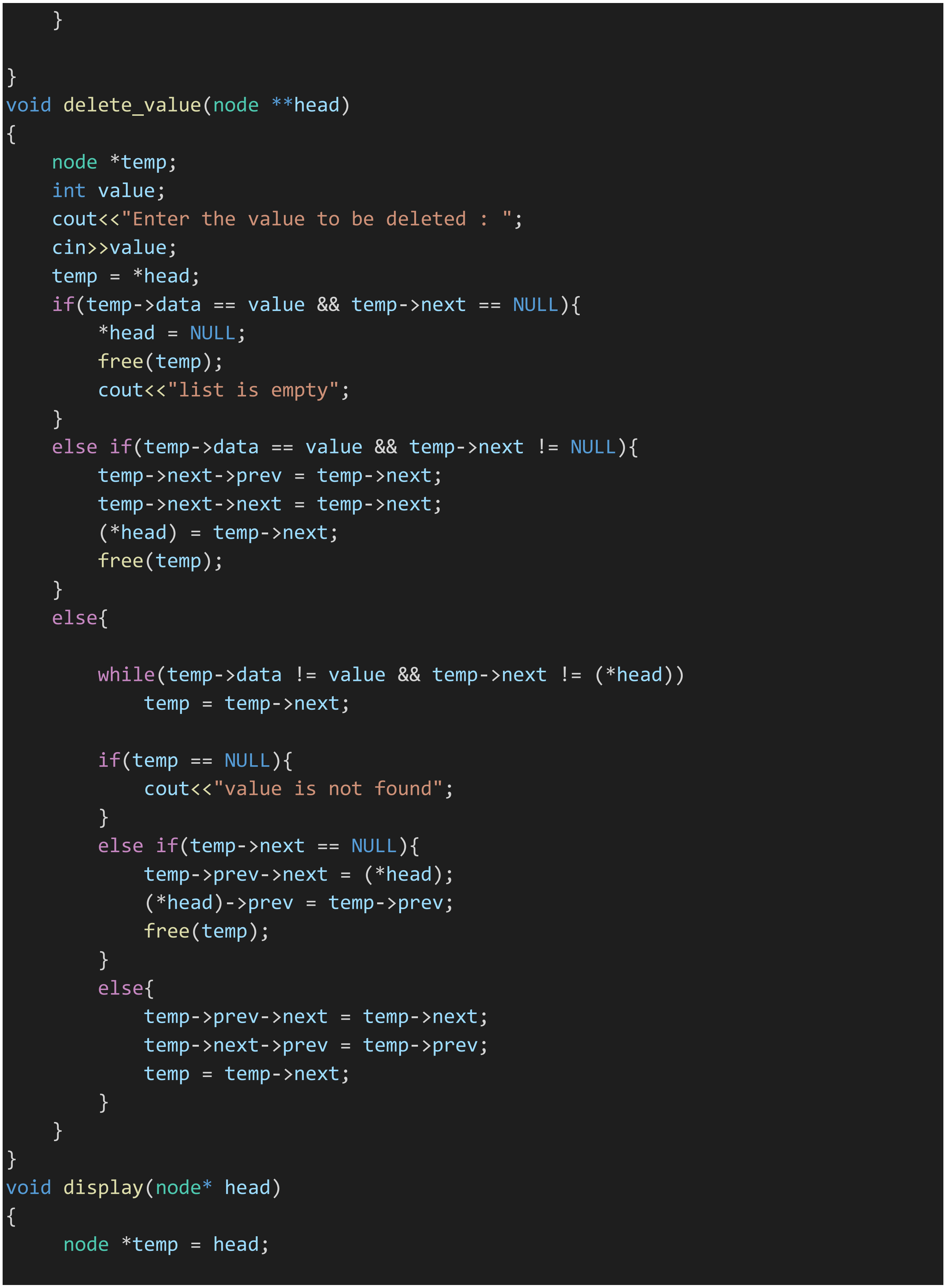
```
#include<iostream>
#include<stdlib.h>
#include<new>

using namespace std;

class node{

public:

    int data;
    node *next;
    node *prev;
};
int getdata(){

     int value;
     cout<<"enter the value : ";
     cin>>value;
     return value;
 }
void insert_atstart(node **head){

       int value = getdata();
        node *new_node = new node();
        new_node->data = value;
        if(*head == NULL){
        new_node->next = new_node;
        new_node->prev = new_node;
        (*head)=new_node;
        }
        else{
       node *last = (*head)->prev;
       new_node->data = value;
       new_node->next = (*head);
       new_node->prev = last;
       last->next = (*head)->prev = new_node;
       (*head) = new_node;
        }
```



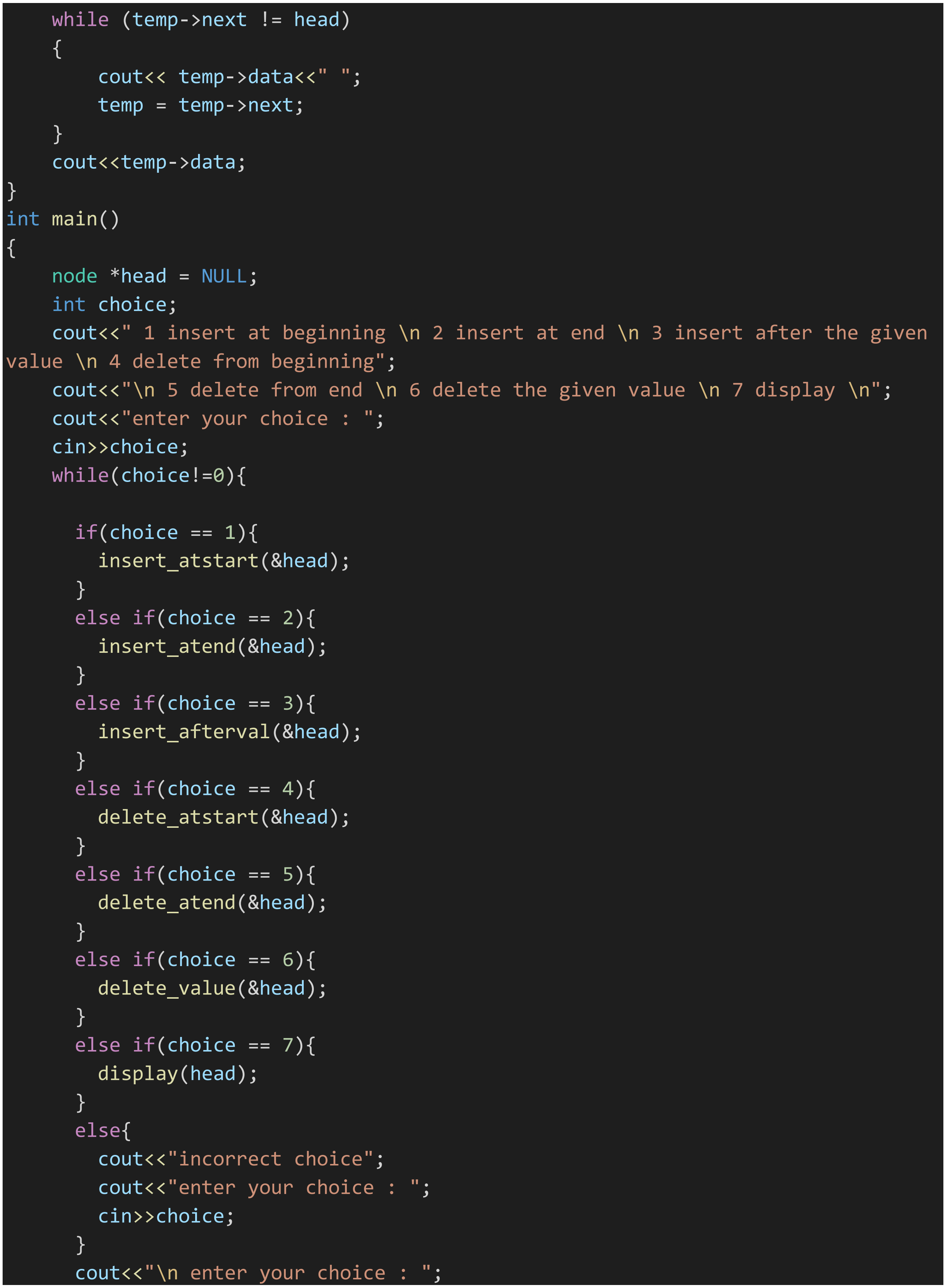
```
}
void insert_atend(node **head)
{

    int value = getdata();
    node *new_node = new node();
    new_node->data = value;
    if(*head == NULL){
        new_node->next = new_node;
        new_node->prev = new_node;
        (*head)=new_node;
    }
    else{
         node *last = (*head)->prev;
         new_node->next = (*head);
         (*head)->prev = new_node;
         new_node->prev = last;
         last->next = new_node;

    }
}
void insert_afterval(node **head)
{
            int value = getdata();
            int uservalue;

            cout<<"enter the aftervalue :";
            cin>>uservalue;
            node *new_node = new node();
            new_node->data = value;
            node *temp = (*head);
            while (temp->data != uservalue)
                 temp = temp->next;

            node *next_node = temp->next;
            temp->next = new_node;
            new_node->prev = temp;
            new_node->next = next_node;
            next_node->prev = new_node;
}
void delete_atstart(node **head)
{

    node *temp;
    if((*head) == NULL)
```



```
    {
        cout<<"UNDERFLOW";
    }
    else if((*head)->next == (*head))
    {
        (*head) = NULL;
        free(*head);
        cout<<"\n Node Deleted \n";
    }
    else
    {
        temp = *head;
        while(temp->next != (*head))
           temp = temp->next;

        temp -> next = (*head) -> next;
        (*head) -> next -> prev = temp;
        free(head);
        (*head) = temp -> next;
        cout<<"\nNode Deleted\n";
    }
}
void delete_atend(node **head)
{
    node *temp = *head;
    if((*head) == NULL)
    {
        cout<<"UNDERFLOW";
    }
    else if(temp->next == (*head))
    {
        (*head) = NULL;
        temp = temp->next;
        cout<<"\n Node Deleted \n";
    }
    else
    {
        while(temp->next != (*head))
        {
            temp = temp -> next;
        }
        temp -> prev -> next = (*head);
        (*head)->prev = temp->prev ;
         free(temp);
        cout<<"\nNode Deleted\n";
```



```
    }

}
void delete_value(node **head)
{
    node *temp;
    int value;
    cout<<"Enter the value to be deleted : ";
    cin>>value;
    temp = *head;
    if(temp->data == value && temp->next == NULL){
        *head = NULL;
        free(temp);
        cout<<"list is empty";
    }
    else if(temp->data == value && temp->next != NULL){
        temp->next->prev = temp->next;
        temp->next->next = temp->next;
        (*head) = temp->next;
        free(temp);
    }
    else{

        while(temp->data != value && temp->next != (*head))
            temp = temp->next;

        if(temp == NULL){
            cout<<"value is not found";
        }
        else if(temp->next == NULL){
            temp->prev->next = (*head);
            (*head)->prev = temp->prev;
            free(temp);
        }
        else{
            temp->prev->next = temp->next;
            temp->next->prev = temp->prev;
            temp = temp->next;
        }
    }
}
void display(node* head)
{
     node *temp = head;
```



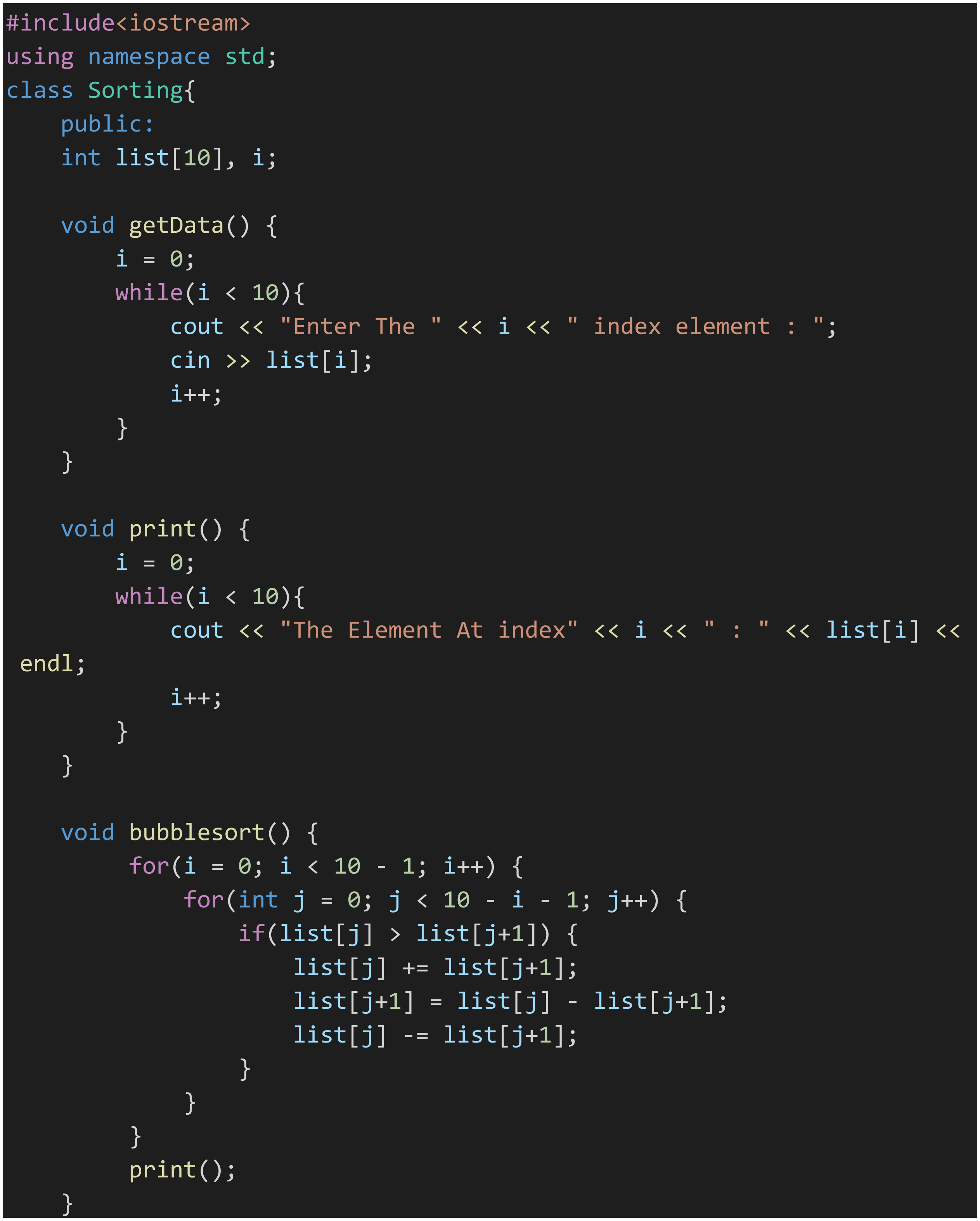
```
    while (temp->next != head)
    {
        cout<< temp->data<<" ";
        temp = temp->next;
    }
    cout<<temp->data;
}
int main()
{
    node *head = NULL;
    int choice;
    cout<<" 1 insert at beginning \n 2 insert at end \n 3 insert after the given value \n 4 delete from beginning";
    cout<<"\n 5 delete from end \n 6 delete the given value \n 7 display \n";
    cout<<"enter your choice : ";
    cin>>choice;
    while(choice!=0){

      if(choice == 1){
        insert_atstart(&head);
      }
      else if(choice == 2){
        insert_atend(&head);
      }
      else if(choice == 3){
        insert_afterval(&head);
      }
      else if(choice == 4){
        delete_atstart(&head);
      }
      else if(choice == 5){
        delete_atend(&head);
      }
      else if(choice == 6){
        delete_value(&head);
      }
      else if(choice == 7){
        display(head);
      }
      else{
        cout<<"incorrect choice";
        cout<<"enter your choice : ";
        cin>>choice;
      }
      cout<<"\n enter your choice : ";
      cin>>choice;

    }
    return 0;
}
```

Output :

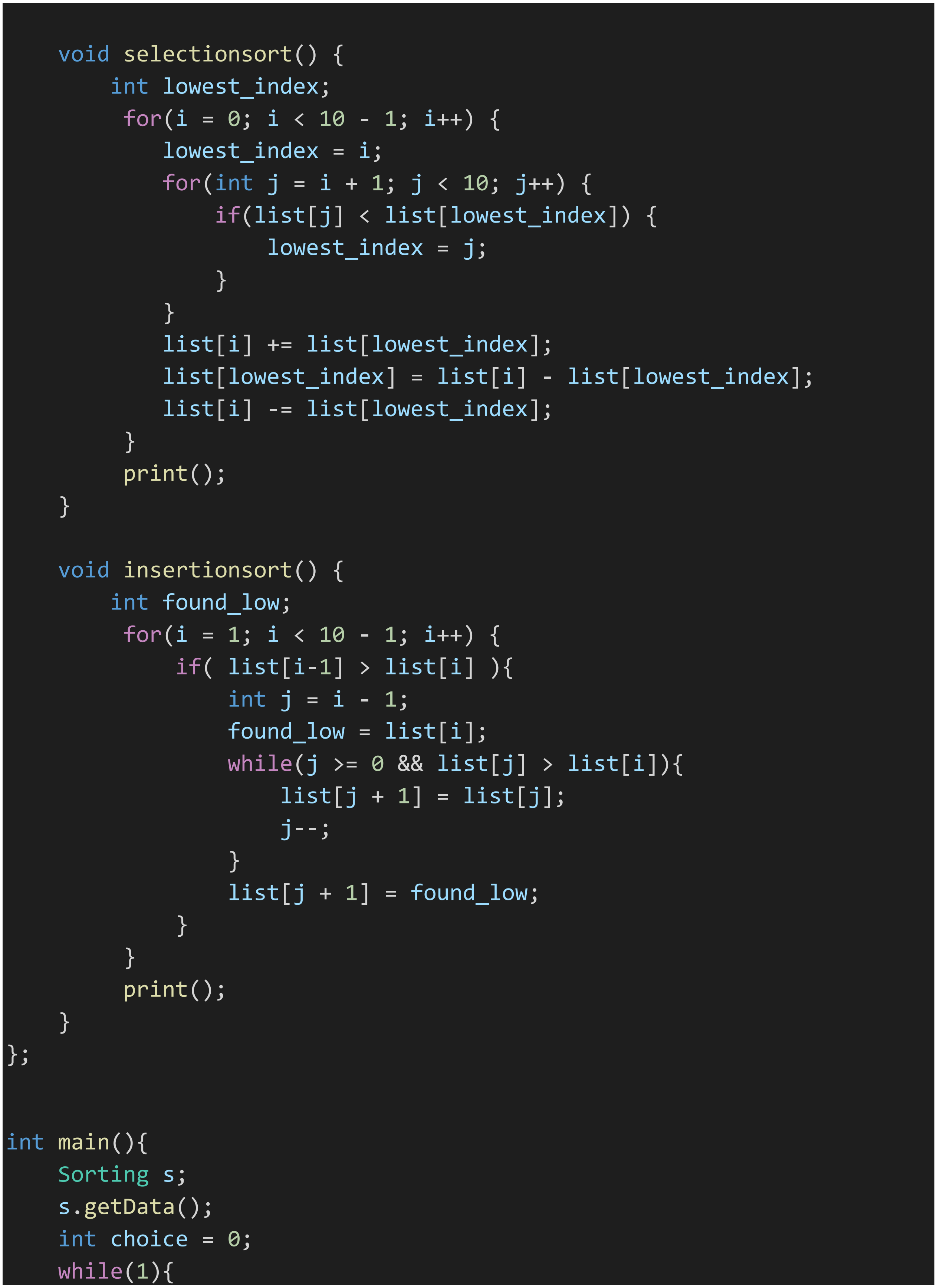
```
1 for insert at beginning

 2 for insert at end

 3 for insert after the given value

 4 for delete from beginning

 5 delete from end

 6 delete the given value

 7 display

enter your choice : 1

enter the value : 3


 enter your choice : 2

enter the value : 9


 enter your choice : 7

3 9

 no of nodes in the linked list are : 2

 enter your choice : 1

enter the value : 5


 enter your choice : 7
```

```
5 3 9

 no of nodes in the linked list are : 3

 enter your choice : 6

Enter the value to be deleted : 5



 enter your choice : 7

3 9

 no of nodes in the linked list are : 2

 enter your choice : 4



  Node Deleted



 enter your choice : 7

9

 no of nodes in the linked list are : 1

 enter your choice : 5



 Node Deleted



 enter your choice : 7



 no of nodes in the linked list are : 0

 enter your choice :
```

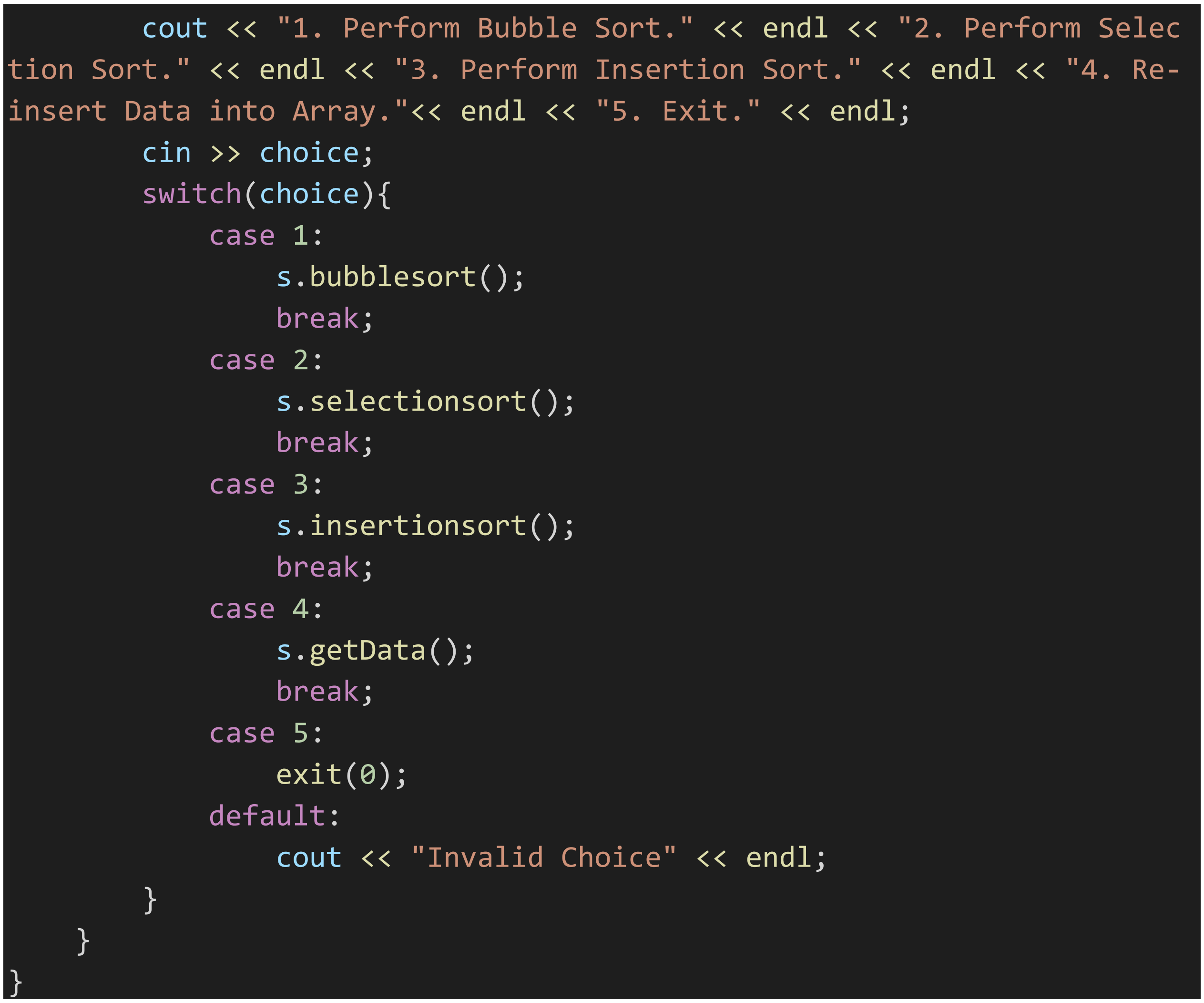
Perform Bubble, selection, insertion sort



```
#include<iostream>
using namespace std;
class Sorting{
    public:
    int list[10], i;

    void getData() {
        i = 0;
        while(i < 10){
            cout << "Enter The " << i << " index element : ";
            cin >> list[i];
            i++;
        }
    }

    void print() {
        i = 0;
        while(i < 10){
            cout << "The Element At index" << i << " : " << list[i] <<
 endl;
            i++;
        }
    }

    void bubblesort() {
         for(i = 0; i < 10 - 1; i++) {
             for(int j = 0; j < 10 - i - 1; j++) {
                 if(list[j] > list[j+1]) {
                     list[j] += list[j+1];
                     list[j+1] = list[j] - list[j+1];
                     list[j] -= list[j+1];
                 }
             }
         }
         print();
    }
```



```
    void selectionsort() {
        int lowest_index;
         for(i = 0; i < 10 - 1; i++) {
            lowest_index = i;
            for(int j = i + 1; j < 10; j++) {
                if(list[j] < list[lowest_index]) {
                    lowest_index = j;
                }
            }
            list[i] += list[lowest_index];
            list[lowest_index] = list[i] - list[lowest_index];
            list[i] -= list[lowest_index];
         }
         print();
    }

    void insertionsort() {
        int found_low;
         for(i = 1; i < 10 - 1; i++) {
             if( list[i-1] > list[i] ){
                 int j = i - 1;
                 found_low = list[i];
                 while(j >= 0 && list[j] > list[i]){
                     list[j + 1] = list[j];
                     j--;
                 }
                 list[j + 1] = found_low;
             }
         }
         print();
    }
};


int main(){
    Sorting s;
    s.getData();
    int choice = 0;
    while(1){
```



```
        cout << "1. Perform Bubble Sort." << endl << "2. Perform Selec
tion Sort." << endl << "3. Perform Insertion Sort." << endl << "4. Re-
insert Data into Array."<< endl << "5. Exit." << endl;
        cin >> choice;
        switch(choice){
            case 1:
                s.bubblesort();
                break;
            case 2:
                s.selectionsort();
                break;
            case 3:
                s.insertionsort();
                break;
            case 4:
                s.getData();
                break;
            case 5:
                exit(0);
            default:
                cout << "Invalid Choice" << endl;
        }
    }
}
```

<u>Ouput :</u>

```
PS E:\MCA\MCA SEM 3\DS> .\bubblesort.exe

Enter The 0 index element : 45

Enter The 1 index element : 12

Enter The 2 index element : 34

Enter The 3 index element : 87

Enter The 4 index element : 3

Enter The 5 index element : 6

Enter The 6 index element : 9
Enter The 7 index element : 10
Enter The 8 index element : 17
Enter The 9 index element : 23
1. Perform Bubble Sort.
2. Perform Selection Sort.
3. Perform Insertion Sort.
4. Re-insert Data into Array.
5. Exit.
1
The Element At index0 : 3
The Element At index1 : 6
The Element At index2 : 9
The Element At index3 : 10
The Element At index4 : 12
The Element At index5 : 17
The Element At index6 : 23
The Element At index7 : 34
The Element At index8 : 45
The Element At index9 : 87
1. Perform Bubble Sort.
2. Perform Selection Sort.
3. Perform Insertion Sort.
4. Re-insert Data into Array.
5. Exit.
5
```

Linear Search

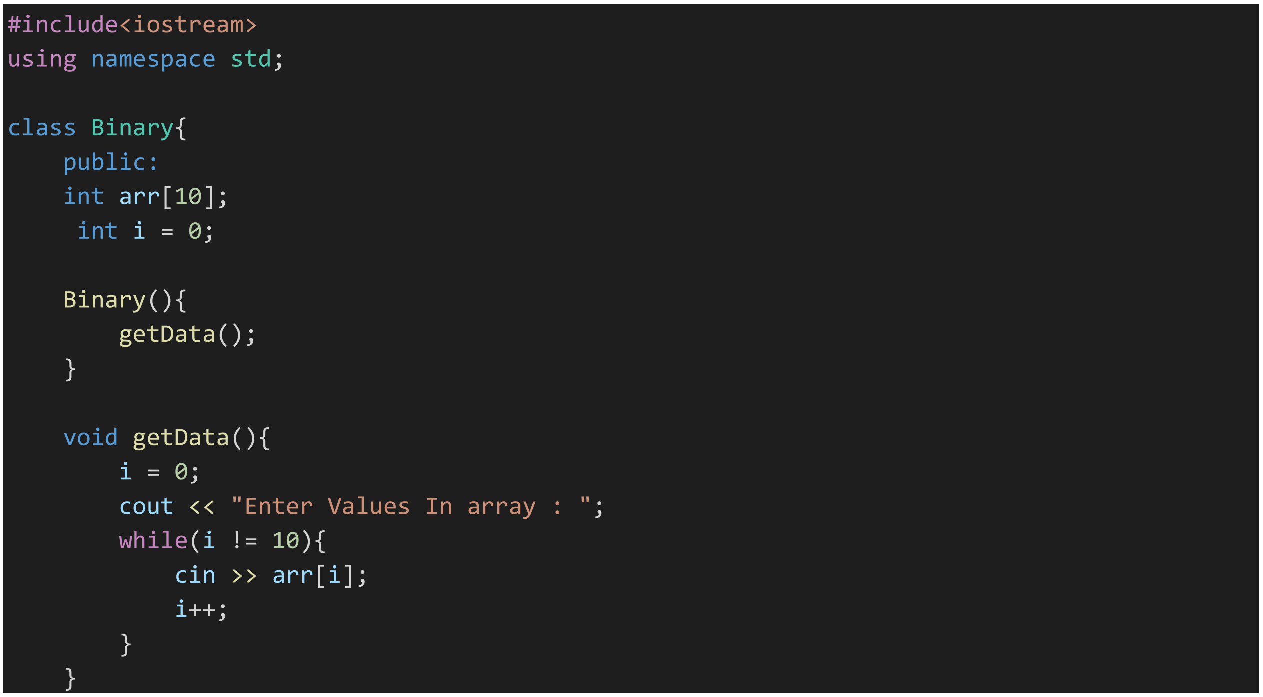
```
#include<iostream>
using namespace std;

class Linear{
    public:
    int arr[10];
     int i = 0;

    Linear(){
        getData();
    }

    void getData(){
        i = 0;
        cout << "Enter Values In array : ";
        while(i != 10){
            cin >> arr[i];
            i++;
        }
    }

    int getNumber(){
        int n;
        cout << "Enter Values You want to Search : ";
        cin >> n;
        return n;
    }

    int linearsearch(int n) {
        i = 0;
        while(i != 10){
            if(arr[i] == n) {
                return i;
            }
        i++;
        }
        return -1;
    }
};

int main(){
    Linear l;
    char ch;
    while(1){

        int search = l.getNumber();
        int result = l.linearsearch(search);
        if(result < 0) cout << "Value Not Present in Array List." << endl;
        else cout << "Search Value Located At " << result << " index of the array" << end
l;
        cout << "Press /'c/' for continue search (Any other character than /'c/' will exi
t the Program) : ";
        cin >> ch;
        if(ch != 'c')
            exit(0);
    }
}
```

Output :

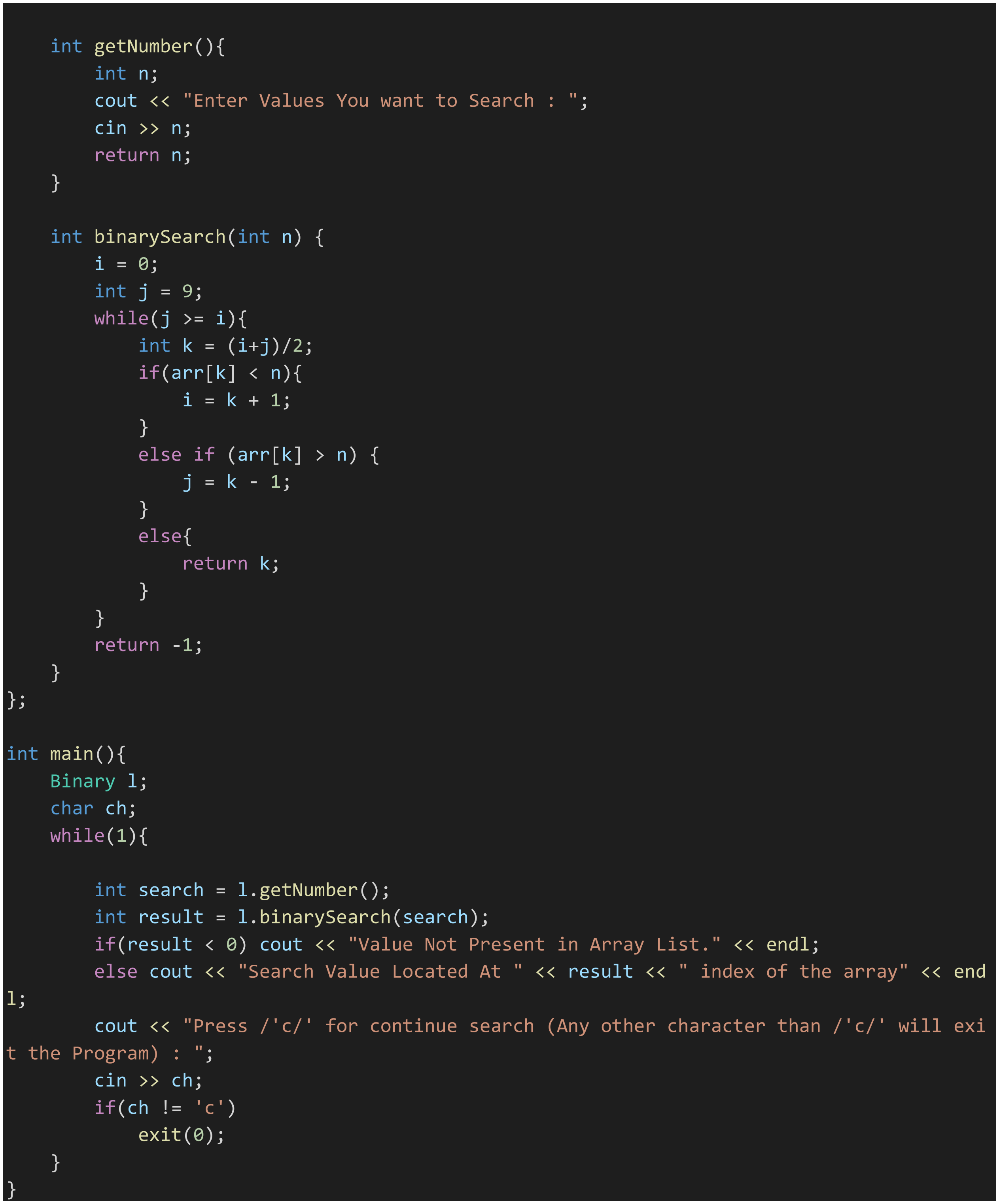
```
PS E:\MCA\MCA SEM 3\DS\40%> .\linear.exe

Enter Values In array : 21

12

34

8

5

40

39

20

11

1

Enter Values You want to Search : 12

Search Value Located At 1 index of the array

Press /'c/' for continue search (Any other character than /'c/' will exit the Program) : c

Enter Values You want to Search : 34

Search Value Located At 2 index of the array

Press /'c/' for continue search (Any other character than /'c/' will exit the Program) : c

Enter Values You want to Search : 21
```

```
Search Value Located At 0 index of the array

Press /'c/' for continue search (Any other character than /'c/' will exit the Program) : c

Enter Values You want to Search : 1

Search Value Located At 9 index of the array

Press /'c/' for continue search (Any other character than /'c/' will exit the Program) : c

Enter Values You want to Search : 5

Search Value Located At 4 index of the array

Press /'c/' for continue search (Any other character than /'c/' will exit the Program) : z
```

Binary Search



```cpp
#include<iostream>
using namespace std;

class Binary{
    public:
    int arr[10];
     int i = 0;

    Binary(){
        getData();
    }

    void getData(){
        i = 0;
        cout << "Enter Values In array : ";
        while(i != 10){
            cin >> arr[i];
            i++;
        }
    }
```



```
    int getNumber(){
        int n;
        cout << "Enter Values You want to Search : ";
        cin >> n;
        return n;
    }

    int binarySearch(int n) {
        i = 0;
        int j = 9;
        while(j >= i){
            int k = (i+j)/2;
            if(arr[k] < n){
                i = k + 1;
            }
            else if (arr[k] > n) {
                j = k - 1;
            }
            else{
                return k;
            }
        }
        return -1;
    }
};

int main(){
    Binary l;
    char ch;
    while(1){

        int search = l.getNumber();
        int result = l.binarySearch(search);
        if(result < 0) cout << "Value Not Present in Array List." << endl;
        else cout << "Search Value Located At " << result << " index of the array" << end
l;
        cout << "Press /'c/' for continue search (Any other character than /'c/' will exi
t the Program) : ";
        cin >> ch;
        if(ch != 'c')
            exit(0);
    }
}
```

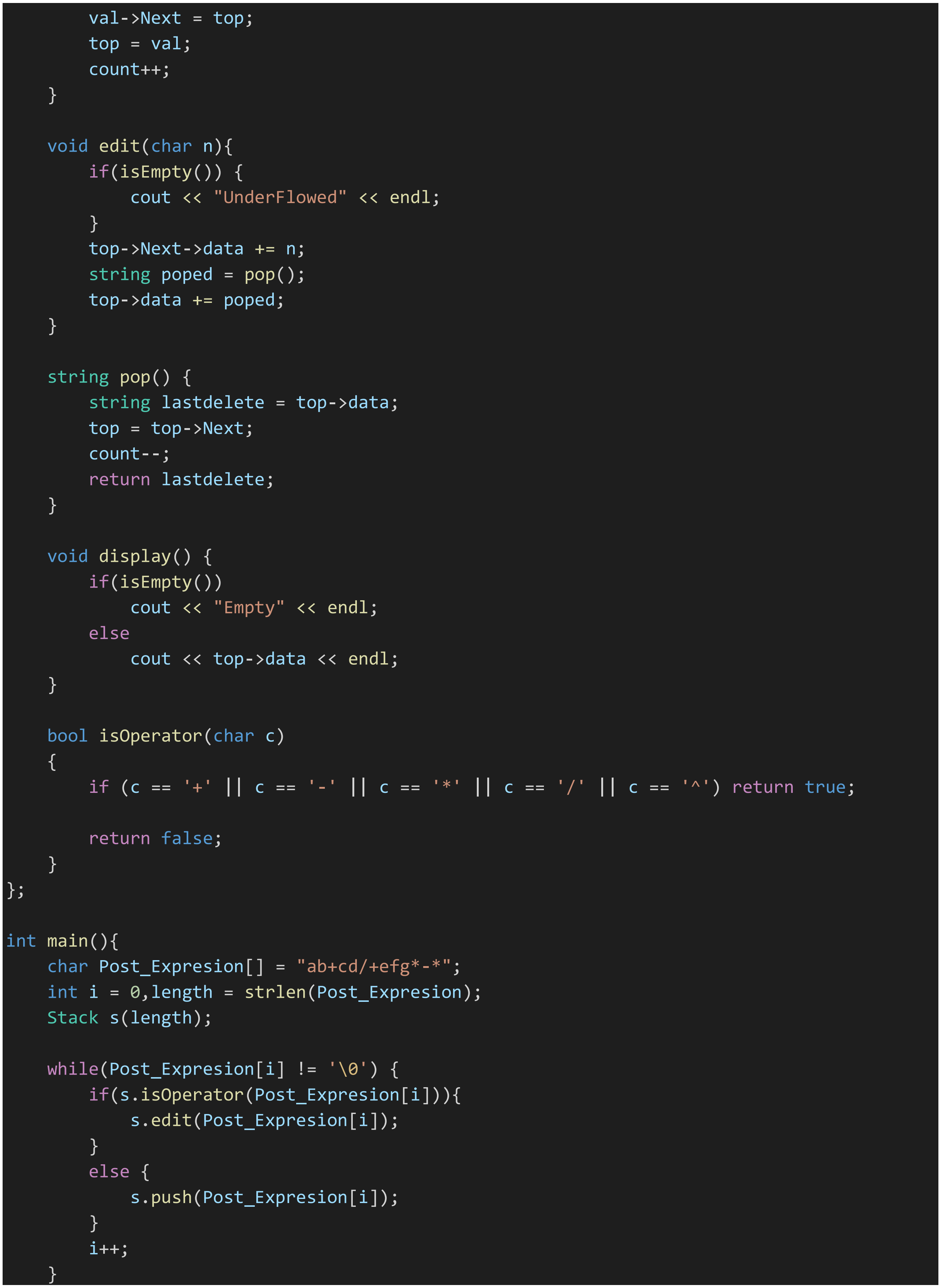
Output :

```
PS E:\MCA\MCA SEM 3\DS\40%> .\Binary.exe
```

```
Enter Values In array : 10

12

14

18

19

22

47

49

62

81

Enter Values You want to Search : 12

Search Value Located At 1 index of the array

Press /'c/' for continue search (Any other character than /'c/' will exit the Program) : c

Enter Values You want to Search : 31

Value Not Present in Array List.

Press /'c/' for continue search (Any other character than /'c/' will exit the Program) : c

Enter Values You want to Search : 62

Search Value Located At 8 index of the array

Press /'c/' for continue search (Any other character than /'c/' will exit the Program) : m
```

Expression Tree

```
#include<iostream>
#include<cstring>
using namespace std;

class Stack {

    class Value {
        public:
        string data;
        Value *Next;

        Value(char n) {
            data = n;
            Next = NULL;
        }
    };

    Value *top;
    int size, count;

    public:

    Stack(int size)
    {
        if (size < 1)
            size = 5;
        this->size = size;
        count = 0;
        top = NULL;
    }

    bool isFull(){
        return (count >= size);
    }

    bool isEmpty(){
        return !top;
    }

    void push(char n){
        if(isFull()){
            cout << "Overflowed" << endl;
            return;
        }
        Value *val = new Value(n);
```



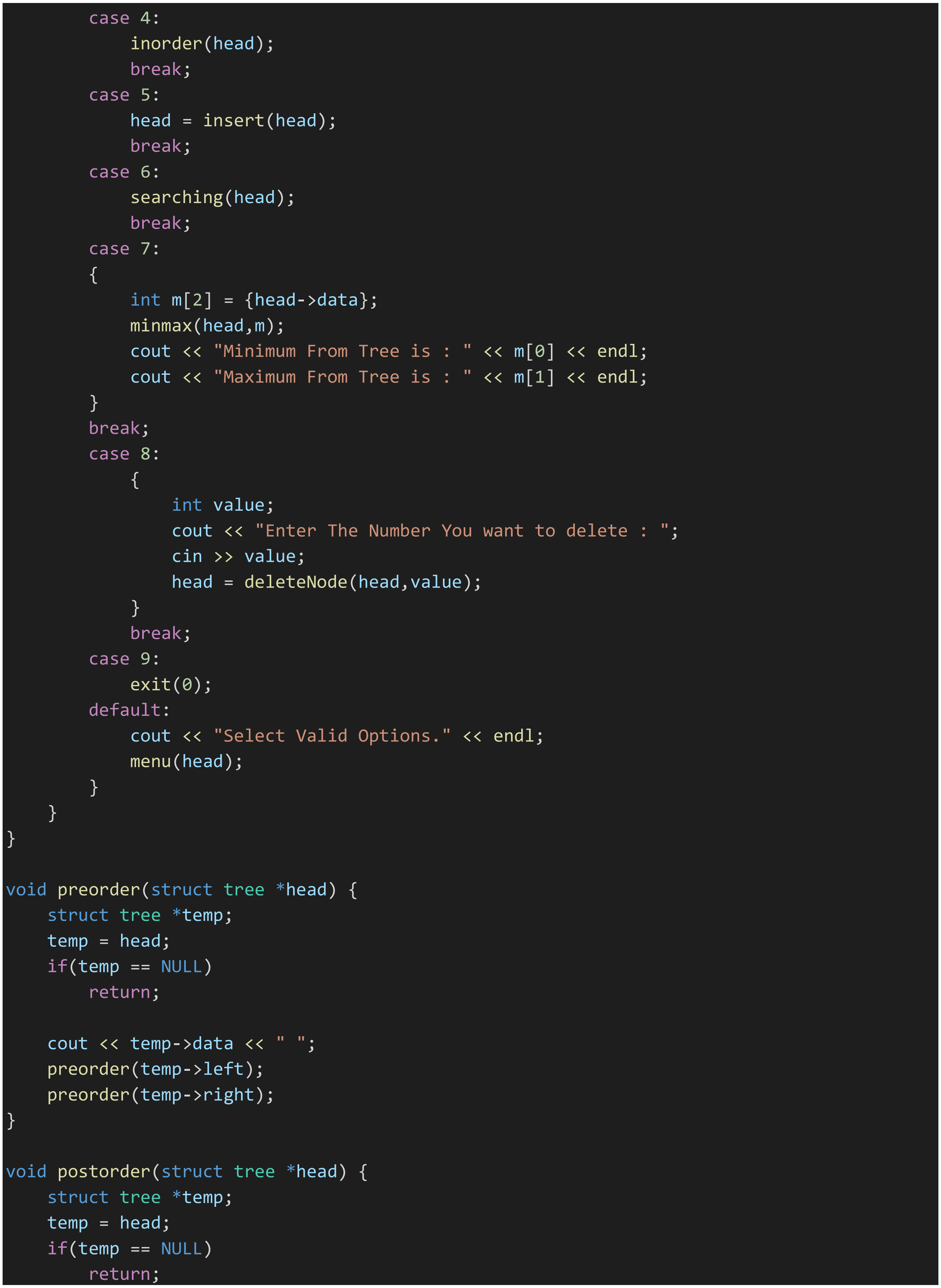
```
        val->Next = top;
        top = val;
        count++;
    }

    void edit(char n){
        if(isEmpty()) {
            cout << "UnderFlowed" << endl;
        }
        top->Next->data += n;
        string poped = pop();
        top->data += poped;
    }

    string pop() {
        string lastdelete = top->data;
        top = top->Next;
        count--;
        return lastdelete;
    }

    void display() {
        if(isEmpty())
            cout << "Empty" << endl;
        else
            cout << top->data << endl;
    }

    bool isOperator(char c)
    {
        if (c == '+' || c == '-' || c == '*' || c == '/' || c == '^') return true;

        return false;
    }
};

int main(){
    char Post_Expresion[] = "ab+cd/+efg*-*";
    int i = 0,length = strlen(Post_Expresion);
    Stack s(length);

    while(Post_Expresion[i] != '\0') {
        if(s.isOperator(Post_Expresion[i])){
            s.edit(Post_Expresion[i]);
        }
        else {
            s.push(Post_Expresion[i]);
        }
        i++;
    }
    s.display();
    return 0;
}
```

Output :

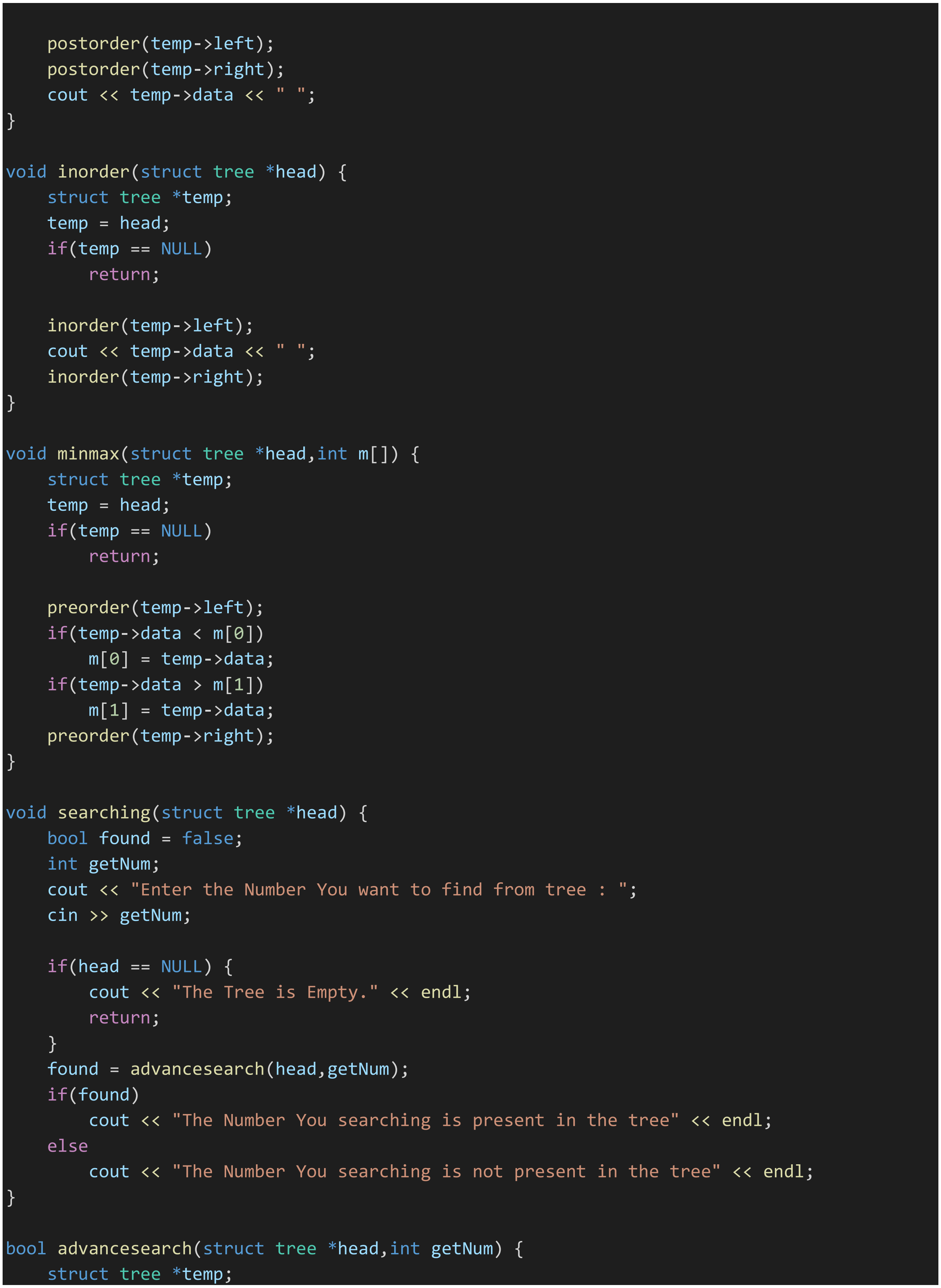
```
PS E:\MCA\MCA SEM 3\DS\40%> .\Expression.exe

a+b+c/d*e-f*g
```

Binary Search Tree

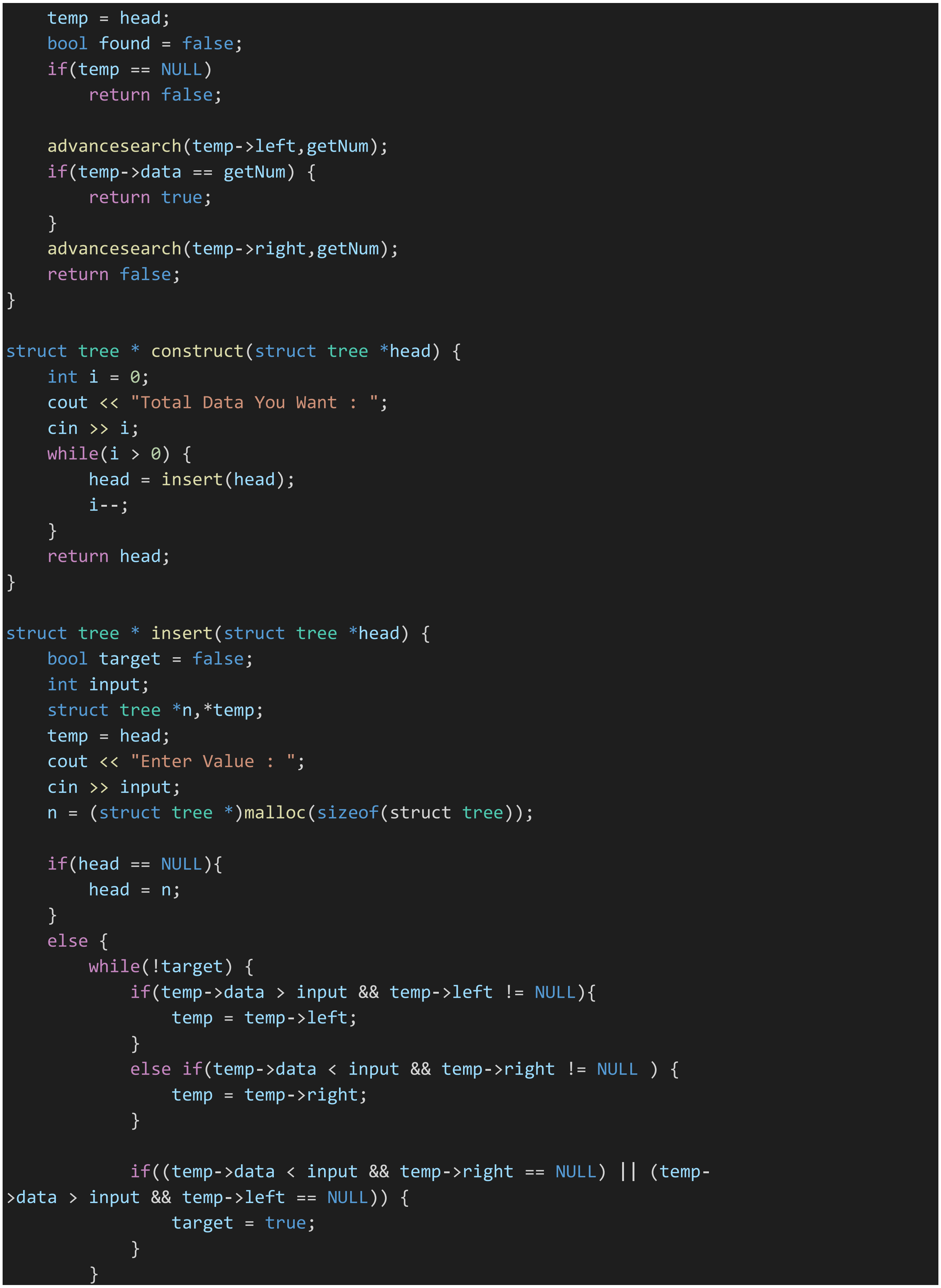
```
#include<iostream>
using namespace std;

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

// method declaration
void menu(struct tree *);
struct tree * construct(struct tree *);
void preorder(struct tree *);
void postorder(struct tree *);
void inorder(struct tree *);
void minmax(struct tree *,int []);
void searching(struct tree *);
bool advancesearch(struct tree *,int);
struct tree * insert(struct tree *);
struct tree * deleteNode(struct tree *,int);
struct tree * minfromRight(struct tree * );

int main(){
    struct tree *head = NULL;
    menu(head);
    return 0;
}


void menu(struct tree *head) {
    int listen = 0;
    while(1){
        cout << endl << "1. Create Tree" << endl << "2. PreOrder Traversal" << endl << "3. PostOrder Traversal" << endl << "4. InOrder Traversal" << endl << "5. Insertion" << endl << "6. Searching" << endl << "7. Find Minimum & Maximum" << endl << "8. Deletion" << endl << "9. Exit"  << endl << "Enter Your Choice : ";
        cin >> listen;
        switch (listen)
        {
        case 1:
            head = construct(head);
            break;
        case 2:
            preorder(head);
            break;
        case 3:
            postorder(head);
            break;
```



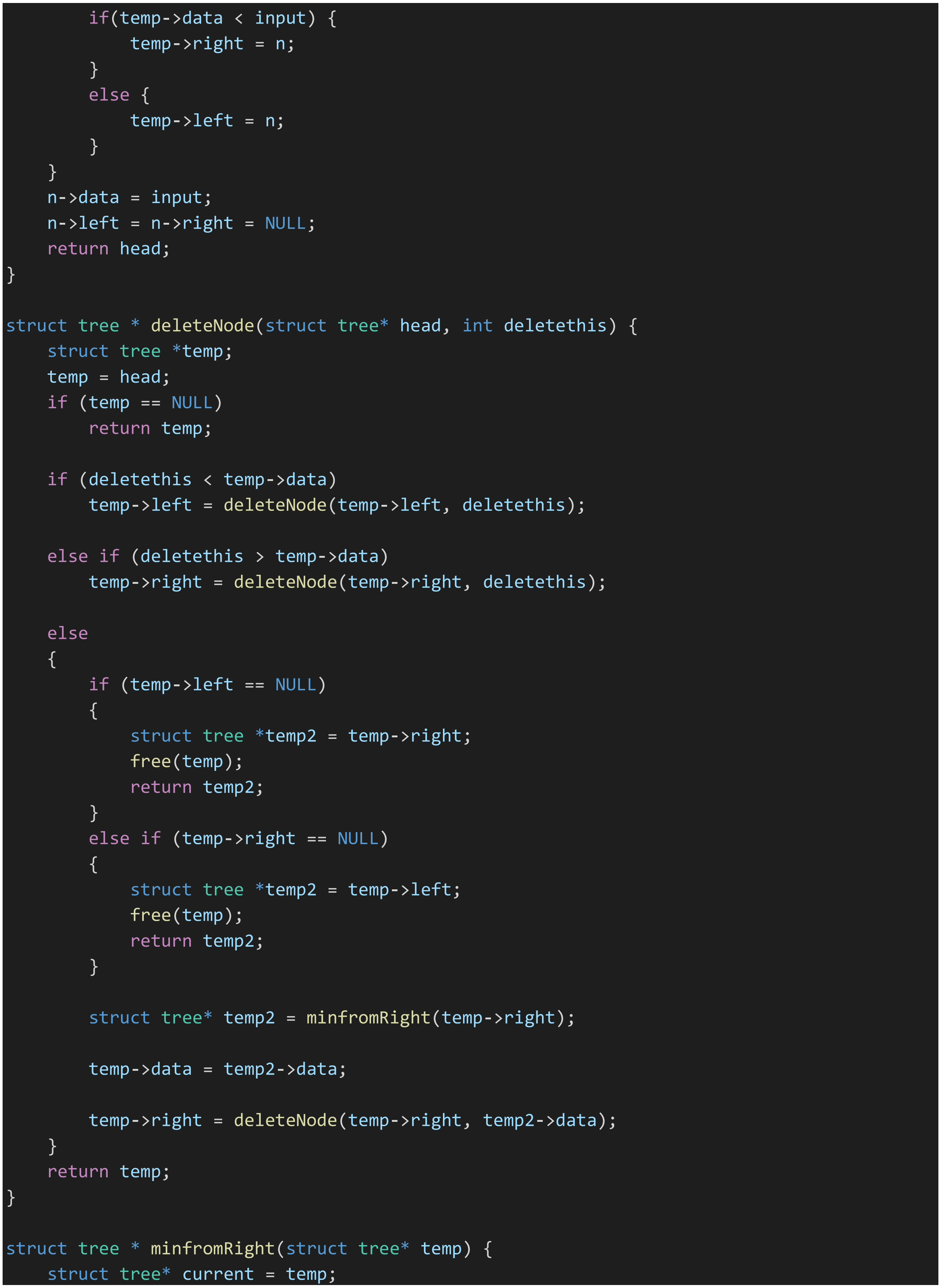
```
        case 4:
            inorder(head);
            break;
        case 5:
            head = insert(head);
            break;
        case 6:
            searching(head);
            break;
        case 7:
        {
            int m[2] = {head->data};
            minmax(head,m);
            cout << "Minimum From Tree is : " << m[0] << endl;
            cout << "Maximum From Tree is : " << m[1] << endl;
        }
        break;
        case 8:
            {
                int value;
                cout << "Enter The Number You want to delete : ";
                cin >> value;
                head = deleteNode(head,value);
            }
            break;
        case 9:
            exit(0);
        default:
            cout << "Select Valid Options." << endl;
            menu(head);
        }
    }
}

void preorder(struct tree *head) {
    struct tree *temp;
    temp = head;
    if(temp == NULL)
        return;

    cout << temp->data << " ";
    preorder(temp->left);
    preorder(temp->right);
}

void postorder(struct tree *head) {
    struct tree *temp;
    temp = head;
    if(temp == NULL)
        return;
```



```
    postorder(temp->left);
    postorder(temp->right);
    cout << temp->data << " ";
}

void inorder(struct tree *head) {
    struct tree *temp;
    temp = head;
    if(temp == NULL)
        return;

    inorder(temp->left);
    cout << temp->data << " ";
    inorder(temp->right);
}

void minmax(struct tree *head,int m[]) {
    struct tree *temp;
    temp = head;
    if(temp == NULL)
        return;

    preorder(temp->left);
    if(temp->data < m[0])
        m[0] = temp->data;
    if(temp->data > m[1])
        m[1] = temp->data;
    preorder(temp->right);
}

void searching(struct tree *head) {
    bool found = false;
    int getNum;
    cout << "Enter the Number You want to find from tree : ";
    cin >> getNum;

    if(head == NULL) {
        cout << "The Tree is Empty." << endl;
        return;
    }
    found = advancesearch(head,getNum);
    if(found)
        cout << "The Number You searching is present in the tree" << endl;
    else
        cout << "The Number You searching is not present in the tree" << endl;
}

bool advancesearch(struct tree *head,int getNum) {
    struct tree *temp;
```



```
    temp = head;
    bool found = false;
    if(temp == NULL)
        return false;

    advancesearch(temp->left,getNum);
    if(temp->data == getNum) {
        return true;
    }
    advancesearch(temp->right,getNum);
    return false;
}

struct tree * construct(struct tree *head) {
    int i = 0;
    cout << "Total Data You Want : ";
    cin >> i;
    while(i > 0) {
        head = insert(head);
        i--;
    }
    return head;
}

struct tree * insert(struct tree *head) {
    bool target = false;
    int input;
    struct tree *n,*temp;
    temp = head;
    cout << "Enter Value : ";
    cin >> input;
    n = (struct tree *)malloc(sizeof(struct tree));

    if(head == NULL){
        head = n;
    }
    else {
        while(!target) {
            if(temp->data > input && temp->left != NULL){
                temp = temp->left;
            }
            else if(temp->data < input && temp->right != NULL ) {
                temp = temp->right;
            }

            if((temp->data < input && temp->right == NULL) || (temp-
>data > input && temp->left == NULL)) {
                target = true;
            }
        }
```



```c
        if(temp->data < input) {
            temp->right = n;
        }
        else {
            temp->left = n;
        }
    }
    n->data = input;
    n->left = n->right = NULL;
    return head;
}

struct tree * deleteNode(struct tree* head, int deletethis) {
    struct tree *temp;
    temp = head;
    if (temp == NULL)
        return temp;

    if (deletethis < temp->data)
        temp->left = deleteNode(temp->left, deletethis);

    else if (deletethis > temp->data)
        temp->right = deleteNode(temp->right, deletethis);

    else
    {
        if (temp->left == NULL)
        {
            struct tree *temp2 = temp->right;
            free(temp);
            return temp2;
        }
        else if (temp->right == NULL)
        {
            struct tree *temp2 = temp->left;
            free(temp);
            return temp2;
        }

        struct tree* temp2 = minfromRight(temp->right);

        temp->data = temp2->data;

        temp->right = deleteNode(temp->right, temp2->data);
    }
    return temp;
}

struct tree * minfromRight(struct tree* temp) {
    struct tree* current = temp;
    while (current && current->left != NULL)
        current = current->left;

    return current;
}
```

Output :

```
PS E:\MCA\MCA SEM 3\DS\40%> .\BST.exe


1. Create Tree
2. PreOrder Traversal
3. PostOrder Traversal
4. InOrder Traversal
5. Insertion
6. Searching
7. Find Minimum & Maximum
8. Deletion
9. Exit
Enter Your Choice : 1
Total Data You Want : 5
Enter Value : 4
Enter Value : 2
Enter Value : 3
Enter Value : 7
Enter Value : 5


1. Create Tree
```

```
2. PreOrder Traversal
3. PostOrder Traversal
4. InOrder Traversal
5. Insertion
6. Searching
7. Find Minimum & Maximum
8. Deletion
9. Exit
Enter Your Choice : 2
4 2 3 7 5
1. Create Tree
2. PreOrder Traversal
3. PostOrder Traversal
4. InOrder Traversal
5. Insertion
6. Searching
7. Find Minimum & Maximum
8. Deletion
9. Exit
Enter Your Choice : 3
3 2 5 7 4
1. Create Tree
2. PreOrder Traversal
3. PostOrder Traversal
4. InOrder Traversal
```

```
5. Insertion
6. Searching
7. Find Minimum & Maximum
8. Deletion
9. Exit
Enter Your Choice : 4
2 3 4 5 7
1. Create Tree
2. PreOrder Traversal
3. PostOrder Traversal
4. InOrder Traversal
5. Insertion
6. Searching
7. Find Minimum & Maximum
8. Deletion
9. Exit
Enter Your Choice : 5
Enter Value : 6

1. Create Tree
2. PreOrder Traversal
3. PostOrder Traversal
4. InOrder Traversal
5. Insertion
6. Searching
```

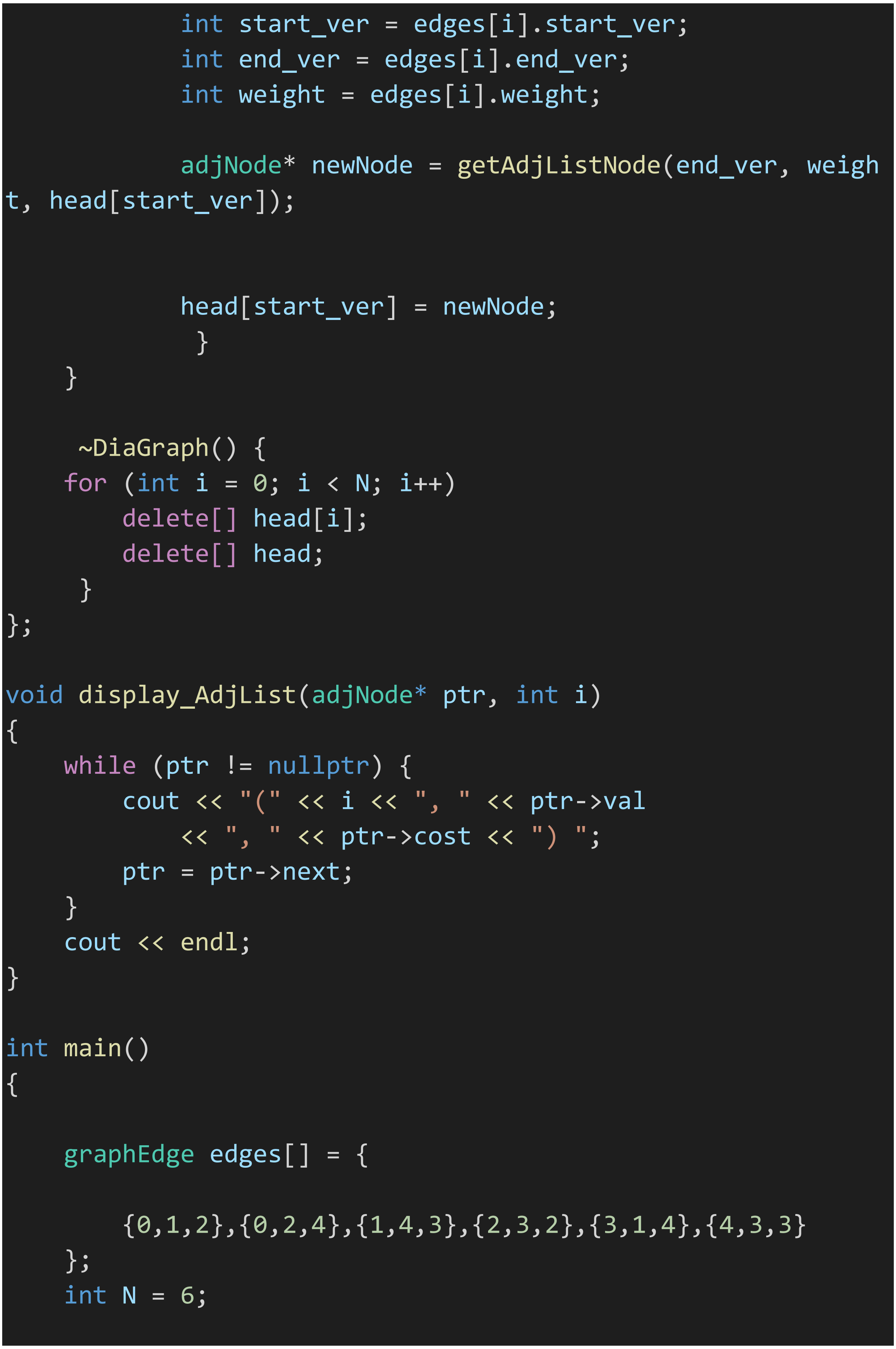
```
7. Find Minimum & Maximum
8. Deletion
9. Exit
Enter Your Choice : 4
2 3 4 5 6 7
1. Create Tree
2. PreOrder Traversal
3. PostOrder Traversal
4. InOrder Traversal
5. Insertion
6. Searching
7. Find Minimum & Maximum
8. Deletion
9. Exit
Enter Your Choice : 6
Enter the Number You want to find from tree : 5
The Number You searching is present in the tree

1. Create Tree
2. PreOrder Traversal
3. PostOrder Traversal
4. InOrder Traversal
5. Insertion
6. Searching
7. Find Minimum & Maximum
```

```
8. Deletion
9. Exit
Enter Your Choice : 7
Minimum From Tree is : 2
Maximum From Tree is : 7

1. Create Tree
2. PreOrder Traversal
3. PostOrder Traversal
4. InOrder Traversal
5. Insertion
6. Searching
7. Find Minimum & Maximum
8. Deletion
9. Exit
Enter Your Choice : 8
Enter The Number You want to delete : 5

1. Create Tree
2. PreOrder Traversal
3. PostOrder Traversal
4. InOrder Traversal
5. Insertion
6. Searching
7. Find Minimum & Maximum
```

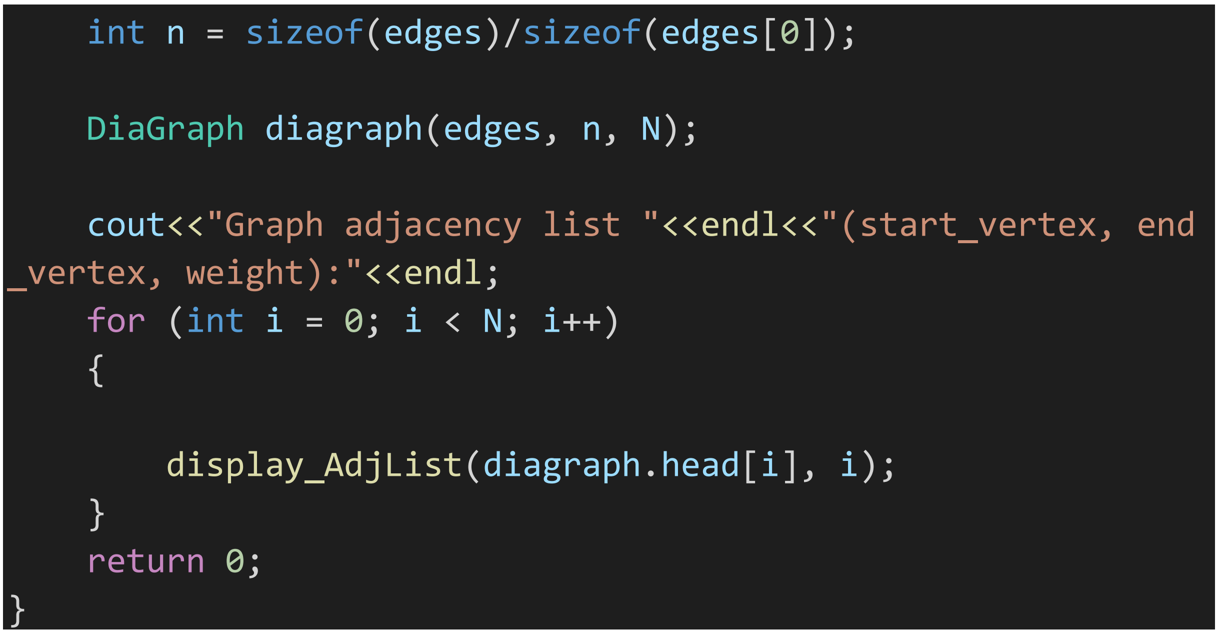
```
8. Deletion
9. Exit
Enter Your Choice : 4
2 3 4 6 7
1. Create Tree
2. PreOrder Traversal
3. PostOrder Traversal
4. InOrder Traversal
5. Insertion
6. Searching
7. Find Minimum & Maximum
8. Deletion
9. Exit
Enter Your Choice : 2
4 2 3 7 6
1. Create Tree
2. PreOrder Traversal
3. PostOrder Traversal
4. InOrder Traversal
5. Insertion
6. Searching
7. Find Minimum & Maximum
8. Deletion
9. Exit
Enter Your Choice : 9
```

Graph Adjacent Node

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

struct adjNode {
    int val, cost;
    adjNode* next;
};

struct graphEdge {
    int start_ver, end_ver, weight;
};
class DiaGraph{

    adjNode* getAdjListNode(int value, int weight, adjNode*
head)   {
        adjNode* newNode = new adjNode;
        newNode->val = value;
        newNode->cost = weight;

        newNode->next = head;
        return newNode;
    }
    int N;
public:
    adjNode **head;

    DiaGraph(graphEdge edges[], int n, int N)  {

        head = new adjNode*[N]();
        this->N = N;

        for (int i = 0; i < N; ++i)
            head[i] = nullptr;

        for (unsigned i = 0; i < n; i++)  {
```

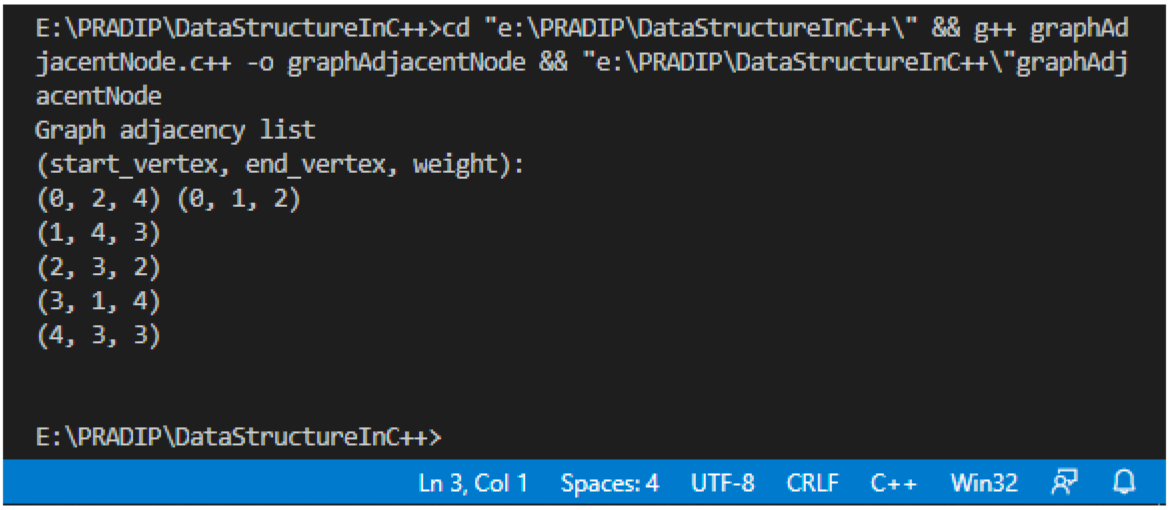


```
            int start_ver = edges[i].start_ver;
            int end_ver = edges[i].end_ver;
            int weight = edges[i].weight;

            adjNode* newNode = getAdjListNode(end_ver, weigh
t, head[start_ver]);


            head[start_ver] = newNode;
             }
    }

     ~DiaGraph() {
    for (int i = 0; i < N; i++)
        delete[] head[i];
        delete[] head;
     }
};

void display_AdjList(adjNode* ptr, int i)
{
    while (ptr != nullptr) {
        cout << "(" << i << ", " << ptr->val
            << ", " << ptr->cost << ") ";
        ptr = ptr->next;
    }
    cout << endl;
}

int main()
{

    graphEdge edges[] = {

        {0,1,2},{0,2,4},{1,4,3},{2,3,2},{3,1,4},{4,3,3}
    };
    int N = 6;
```



```
    int n = sizeof(edges)/sizeof(edges[0]);

    DiaGraph diagraph(edges, n, N);

    cout<<"Graph adjacency list "<<endl<<"(start_vertex, end
_vertex, weight):"<<endl;
    for (int i = 0; i < N; i++)
    {

        display_AdjList(diagraph.head[i], i);
    }
    return 0;
}
```

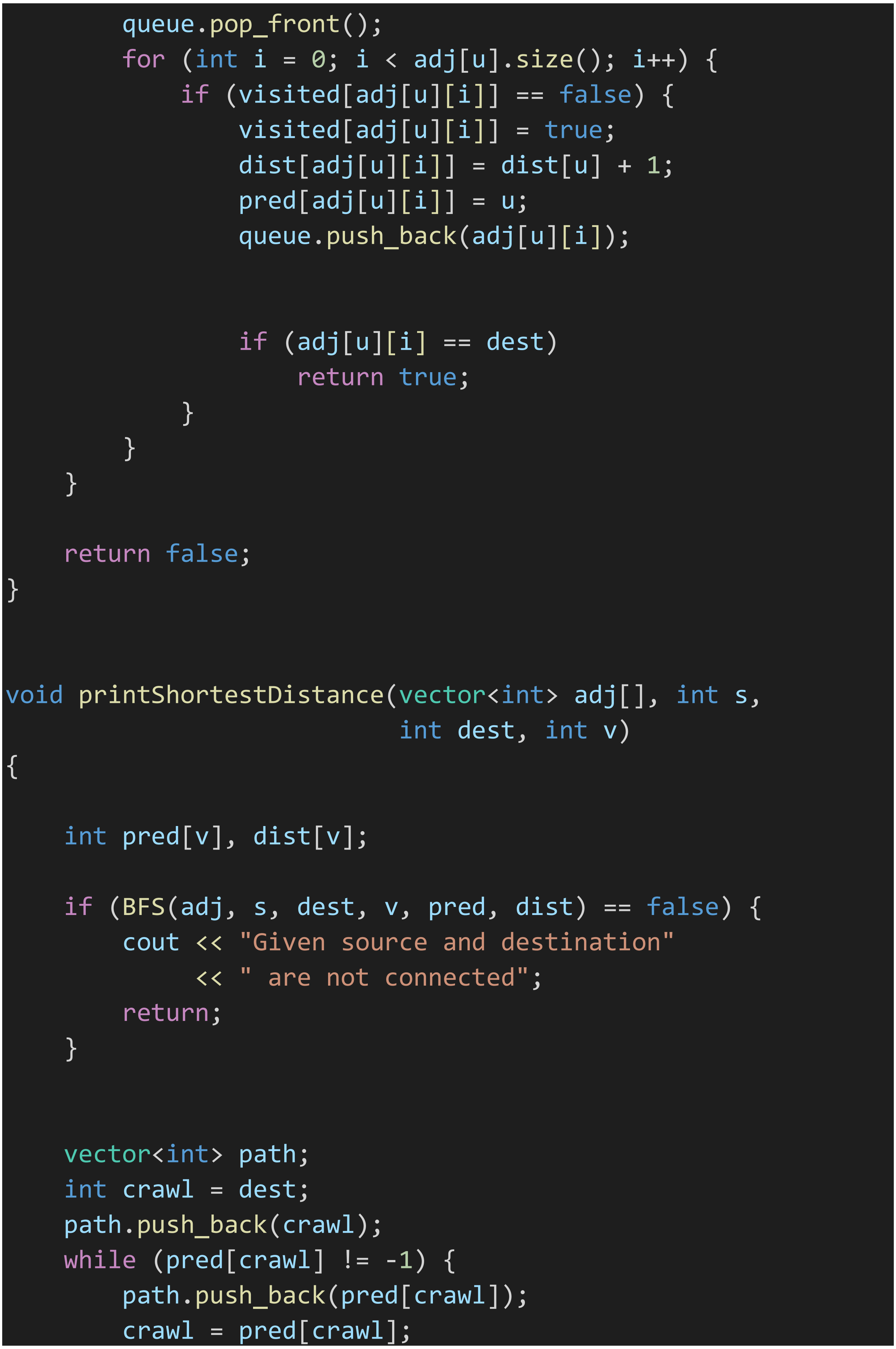
Output:



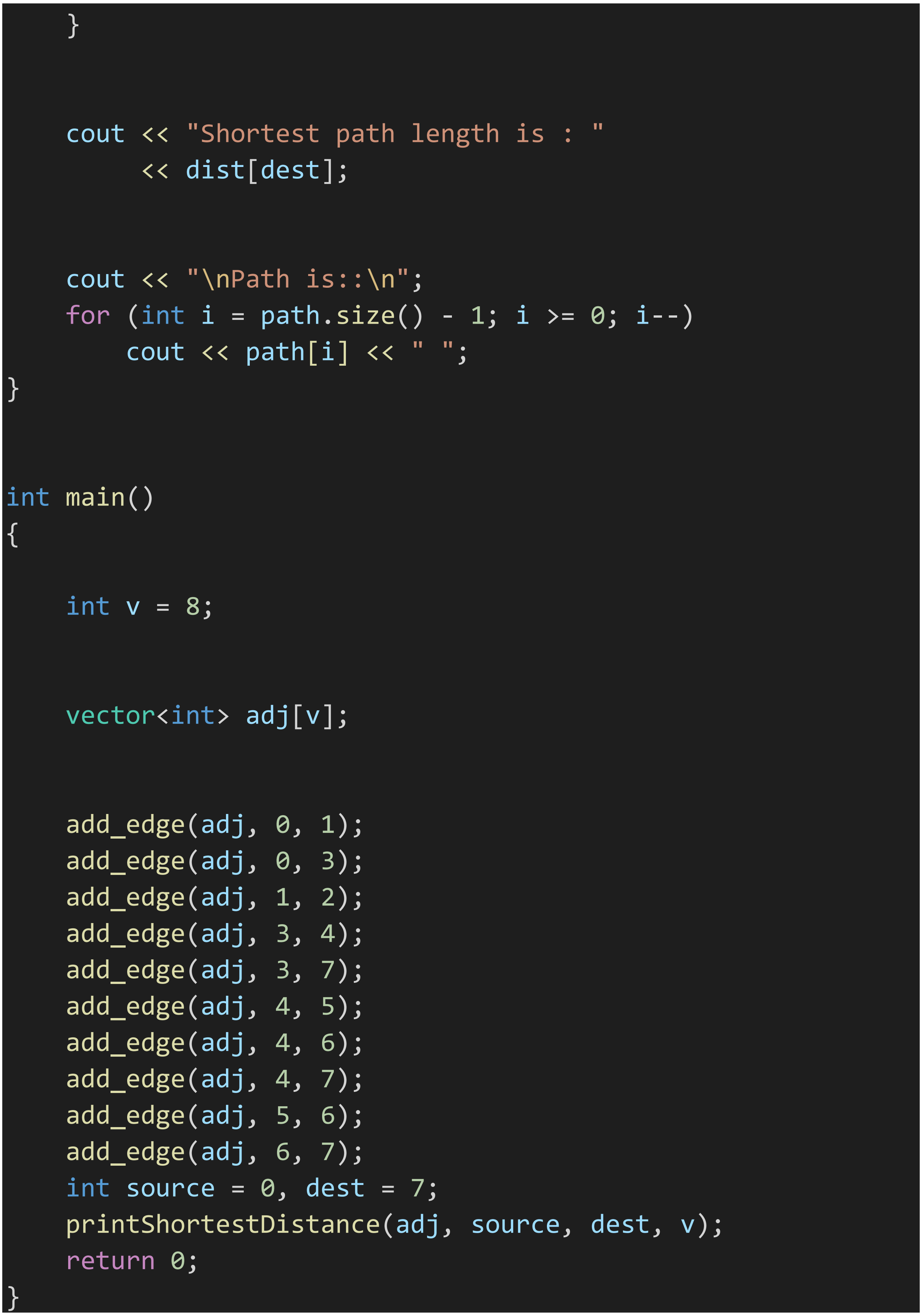
```
E:\PRADIP\DataStructureInC++>cd "e:\PRADIP\DataStructureInC++\" && g++ graphAd
jacentNode.c++ -o graphAdjacentNode && "e:\PRADIP\DataStructureInC++\"graphAdj
acentNode
Graph adjacency list
(start_vertex, end_vertex, weight):
(0, 2, 4) (0, 1, 2)
(1, 4, 3)
(2, 3, 2)
(3, 1, 4)
(4, 3, 3)


E:\PRADIP\DataStructureInC++>
```

Unweighted Graph Shortest path

```cpp
#include <bits/stdc++.h>
using namespace std;

void add_edge(vector<int> adj[], int src, int dest)
{
    adj[src].push_back(dest);
    adj[dest].push_back(src);
}


bool BFS(vector<int> adj[], int src, int dest, int v,
         int pred[], int dist[])
{

    list<int> queue;


    bool visited[v];


    for (int i = 0; i < v; i++) {
        visited[i] = false;
        dist[i] = INT_MAX;
        pred[i] = -1;
    }


    visited[src] = true;
    dist[src] = 0;
    queue.push_back(src);


    while (!queue.empty()) {
        int u = queue.front();
```

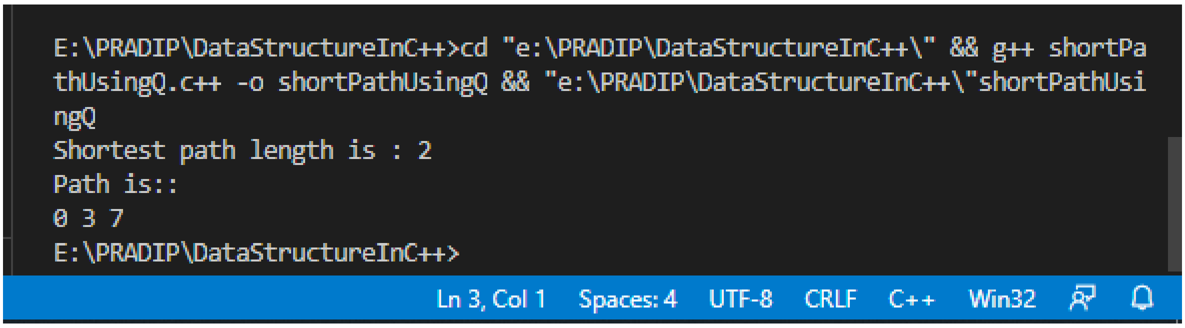


```
        queue.pop_front();
        for (int i = 0; i < adj[u].size(); i++) {
            if (visited[adj[u][i]] == false) {
                visited[adj[u][i]] = true;
                dist[adj[u][i]] = dist[u] + 1;
                pred[adj[u][i]] = u;
                queue.push_back(adj[u][i]);


                if (adj[u][i] == dest)
                    return true;
            }
        }
    }

    return false;
}


void printShortestDistance(vector<int> adj[], int s,
                           int dest, int v)
{

    int pred[v], dist[v];

    if (BFS(adj, s, dest, v, pred, dist) == false) {
        cout << "Given source and destination"
             << " are not connected";
        return;
    }


    vector<int> path;
    int crawl = dest;
    path.push_back(crawl);
    while (pred[crawl] != -1) {
        path.push_back(pred[crawl]);
        crawl = pred[crawl];
```



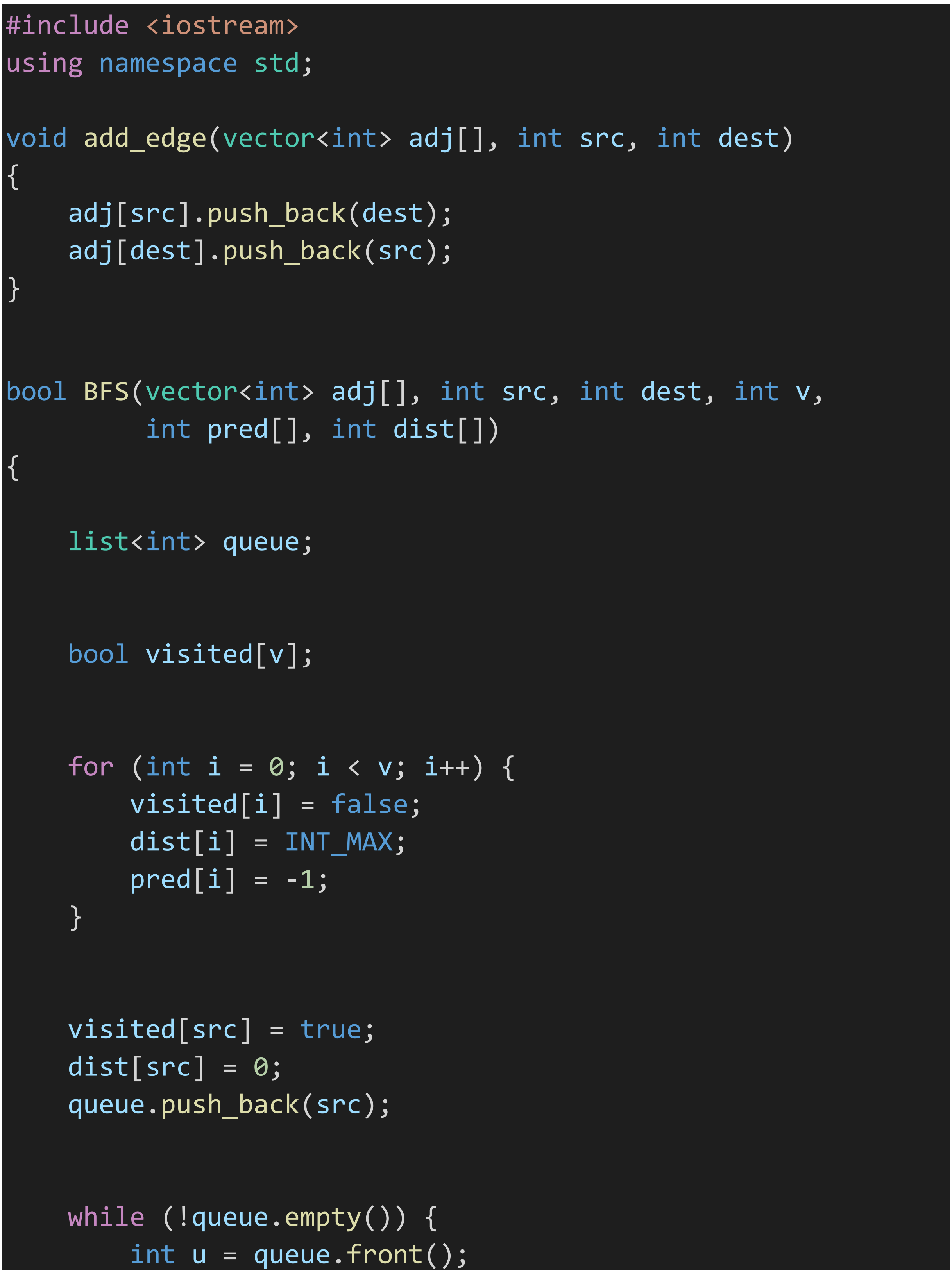
```
    }


    cout << "Shortest path length is : "
         << dist[dest];


    cout << "\nPath is::\n";
    for (int i = path.size() - 1; i >= 0; i--)
        cout << path[i] << " ";
}


int main()
{

    int v = 8;


    vector<int> adj[v];


    add_edge(adj, 0, 1);
    add_edge(adj, 0, 3);
    add_edge(adj, 1, 2);
    add_edge(adj, 3, 4);
    add_edge(adj, 3, 7);
    add_edge(adj, 4, 5);
    add_edge(adj, 4, 6);
    add_edge(adj, 4, 7);
    add_edge(adj, 5, 6);
    add_edge(adj, 6, 7);
    int source = 0, dest = 7;
    printShortestDistance(adj, source, dest, v);
    return 0;
}
```

Output:

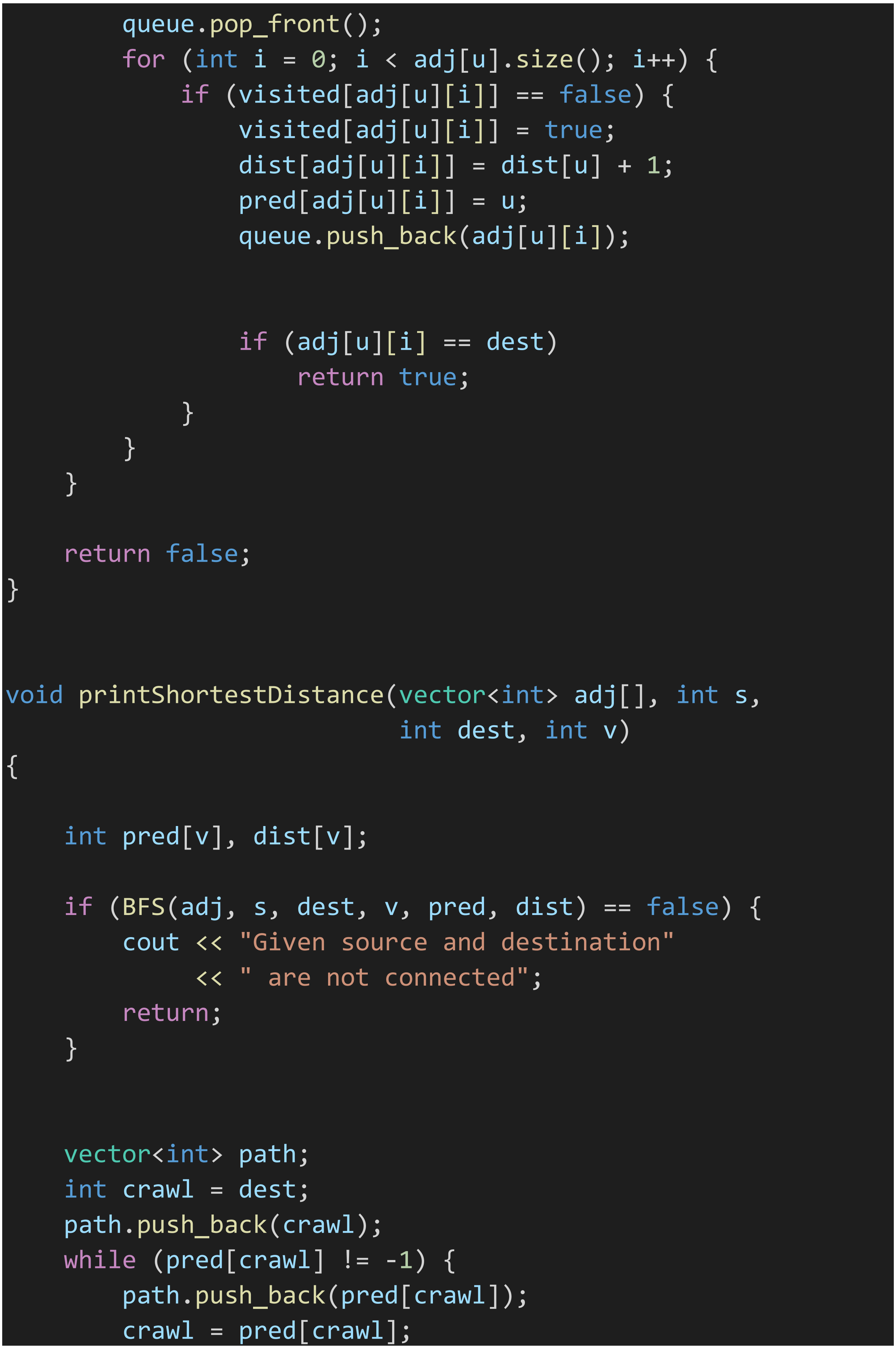


```
E:\PRADIP\DataStructureInC++>cd "e:\PRADIP\DataStructureInC++\" && g++ shortPa
thUsingQ.c++ -o shortPathUsingQ && "e:\PRADIP\DataStructureInC++\"shortPathUsi
ngQ
Shortest path length is : 2
Path is::
0 3 7
E:\PRADIP\DataStructureInC++>
```

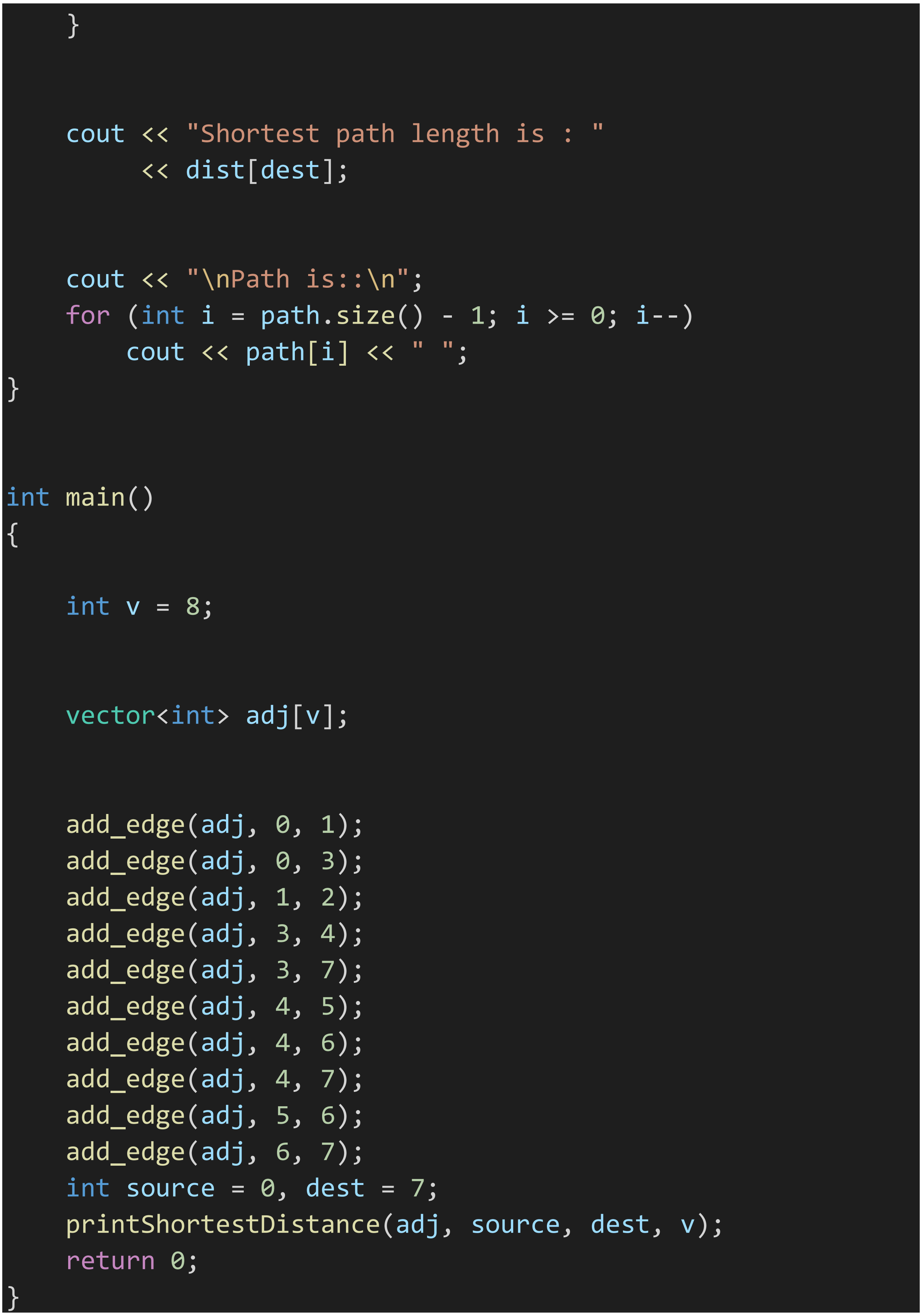
Unweighted Graph Shortest path using Queue



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

void add_edge(vector<int> adj[], int src, int dest)
{
    adj[src].push_back(dest);
    adj[dest].push_back(src);
}


bool BFS(vector<int> adj[], int src, int dest, int v,
         int pred[], int dist[])
{

    list<int> queue;


    bool visited[v];


    for (int i = 0; i < v; i++) {
        visited[i] = false;
        dist[i] = INT_MAX;
        pred[i] = -1;
    }


    visited[src] = true;
    dist[src] = 0;
    queue.push_back(src);


    while (!queue.empty()) {
        int u = queue.front();
```

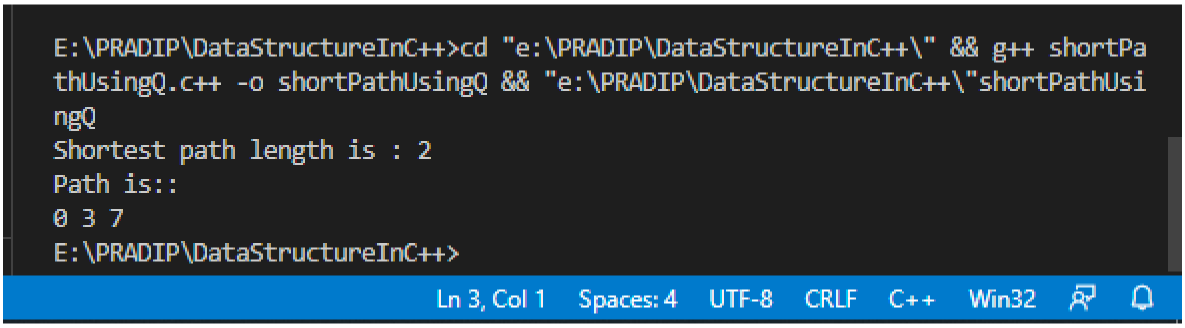


```
        queue.pop_front();
        for (int i = 0; i < adj[u].size(); i++) {
            if (visited[adj[u][i]] == false) {
                visited[adj[u][i]] = true;
                dist[adj[u][i]] = dist[u] + 1;
                pred[adj[u][i]] = u;
                queue.push_back(adj[u][i]);


                if (adj[u][i] == dest)
                    return true;
            }
        }
    }

    return false;
}


void printShortestDistance(vector<int> adj[], int s,
                           int dest, int v)
{

    int pred[v], dist[v];

    if (BFS(adj, s, dest, v, pred, dist) == false) {
        cout << "Given source and destination"
             << " are not connected";
        return;
    }


    vector<int> path;
    int crawl = dest;
    path.push_back(crawl);
    while (pred[crawl] != -1) {
        path.push_back(pred[crawl]);
        crawl = pred[crawl];
```



```
    }


    cout << "Shortest path length is : "
         << dist[dest];


    cout << "\nPath is::\n";
    for (int i = path.size() - 1; i >= 0; i--)
        cout << path[i] << " ";
}


int main()
{

    int v = 8;


    vector<int> adj[v];


    add_edge(adj, 0, 1);
    add_edge(adj, 0, 3);
    add_edge(adj, 1, 2);
    add_edge(adj, 3, 4);
    add_edge(adj, 3, 7);
    add_edge(adj, 4, 5);
    add_edge(adj, 4, 6);
    add_edge(adj, 4, 7);
    add_edge(adj, 5, 6);
    add_edge(adj, 6, 7);
    int source = 0, dest = 7;
    printShortestDistance(adj, source, dest, v);
    return 0;
}
```

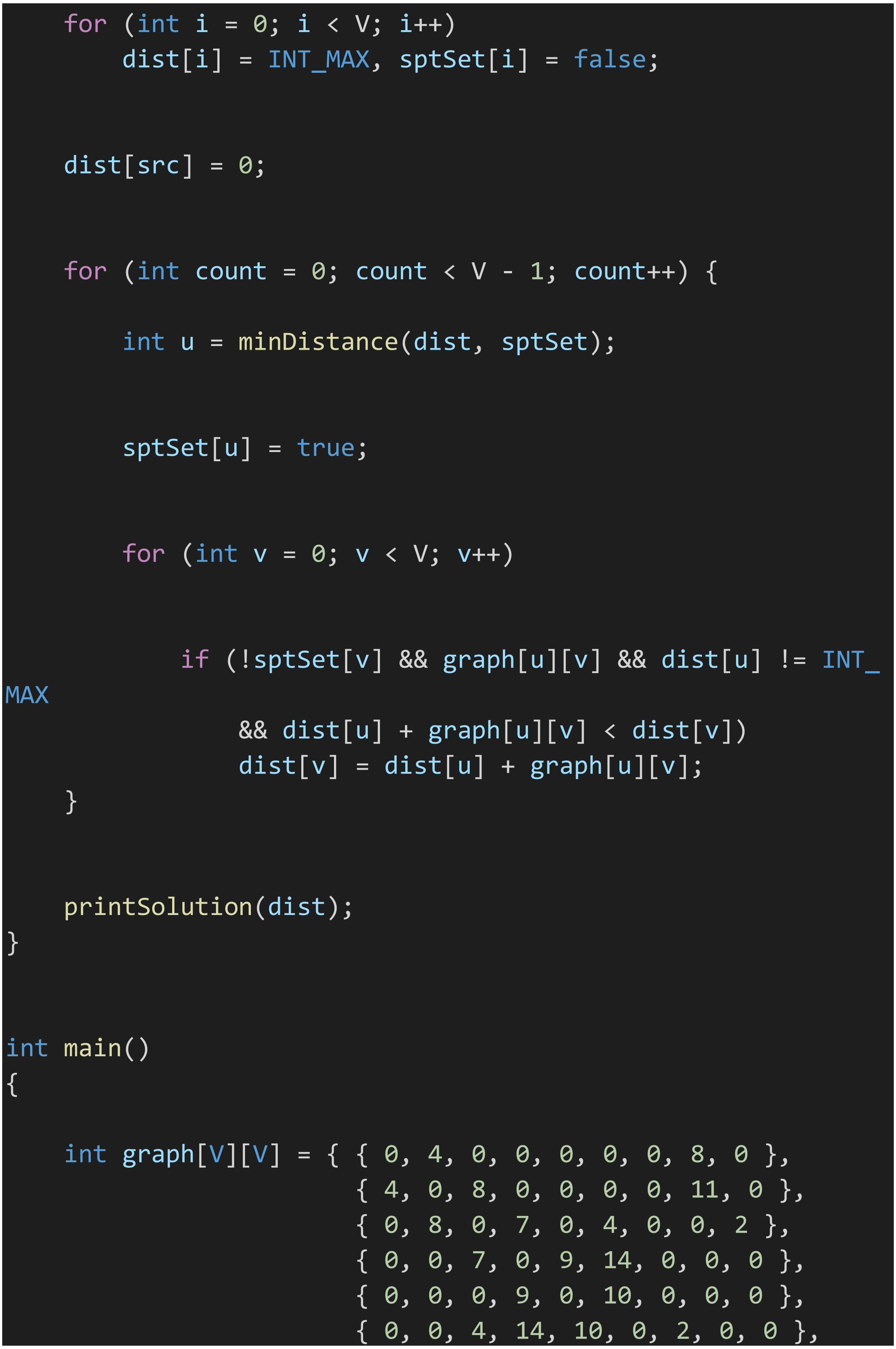
Output:



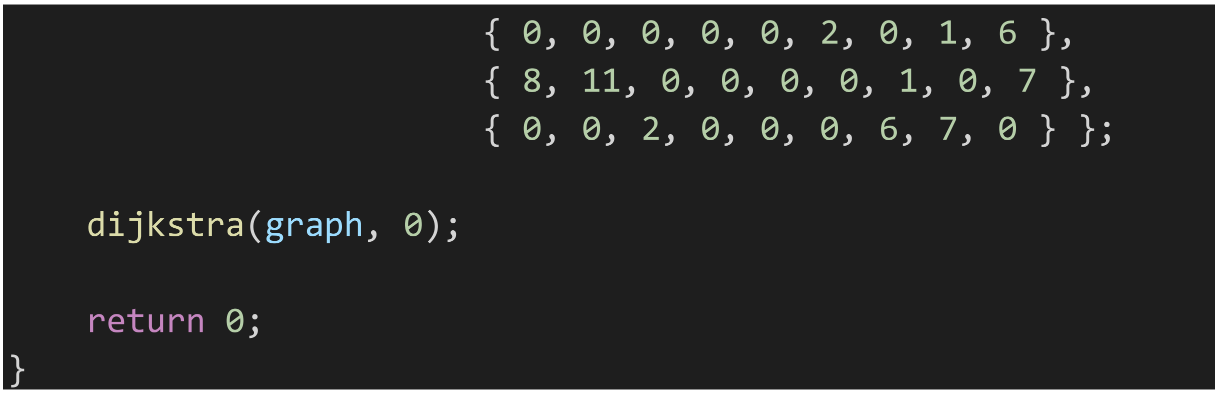
```
E:\PRADIP\DataStructureInC++>cd "e:\PRADIP\DataStructureInC++\" && g++ shortPa
thUsingQ.c++ -o shortPathUsingQ && "e:\PRADIP\DataStructureInC++\"shortPathUsi
ngQ
Shortest path length is : 2
Path is::
0 3 7
E:\PRADIP\DataStructureInC++>
```

Dijkstra's Weighted Graph Shortest Path

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


#define V 9


int minDistance(int dist[], bool sptSet[])
{

    int min = INT_MAX, min_index;

    for (int v = 0; v < V; v++)
        if (sptSet[v] == false && dist[v] <= min)
            min = dist[v], min_index = v;

    return min_index;
}


void printSolution(int dist[])
{
    printf("Vertex \t\t Distance from Source\n");
    for (int i = 0; i < V; i++)
        printf("%d \t\t %d\n", i, dist[i]);
}


void dijkstra(int graph[V][V], int src)
{
    int dist[V];


    bool sptSet[V];
```

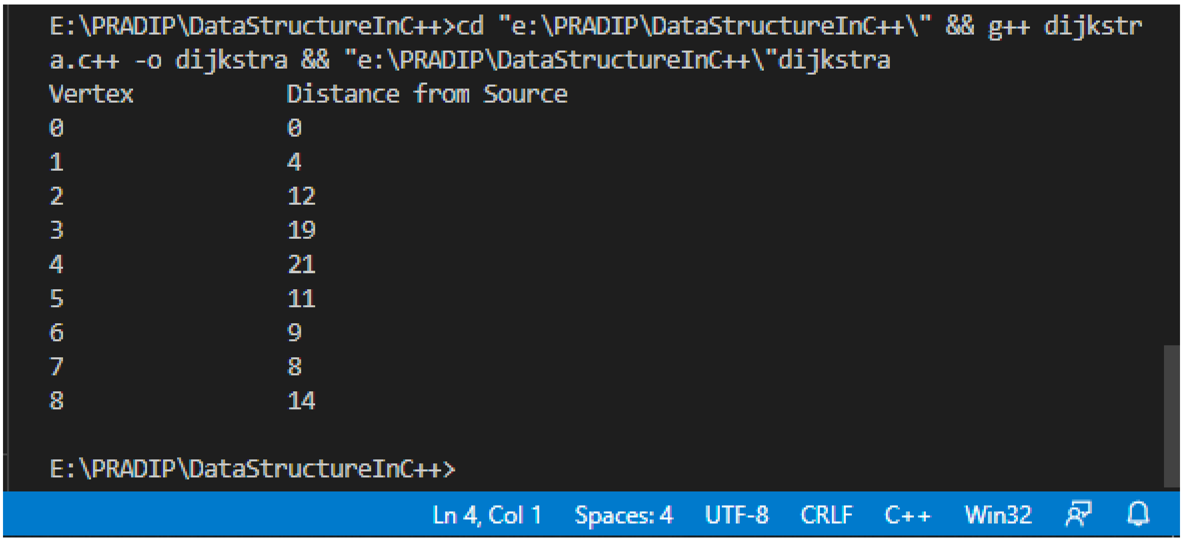


```
    for (int i = 0; i < V; i++)
        dist[i] = INT_MAX, sptSet[i] = false;


    dist[src] = 0;


    for (int count = 0; count < V - 1; count++) {

        int u = minDistance(dist, sptSet);


        sptSet[u] = true;


        for (int v = 0; v < V; v++)


            if (!sptSet[v] && graph[u][v] && dist[u] != INT_
MAX
                && dist[u] + graph[u][v] < dist[v])
                dist[v] = dist[u] + graph[u][v];
    }


    printSolution(dist);
}


int main()
{

    int graph[V][V] = { { 0, 4, 0, 0, 0, 0, 0, 8, 0 },
                        { 4, 0, 8, 0, 0, 0, 0, 11, 0 },
                        { 0, 8, 0, 7, 0, 4, 0, 0, 2 },
                        { 0, 0, 7, 0, 9, 14, 0, 0, 0 },
                        { 0, 0, 0, 9, 0, 10, 0, 0, 0 },
                        { 0, 0, 4, 14, 10, 0, 2, 0, 0 },
```



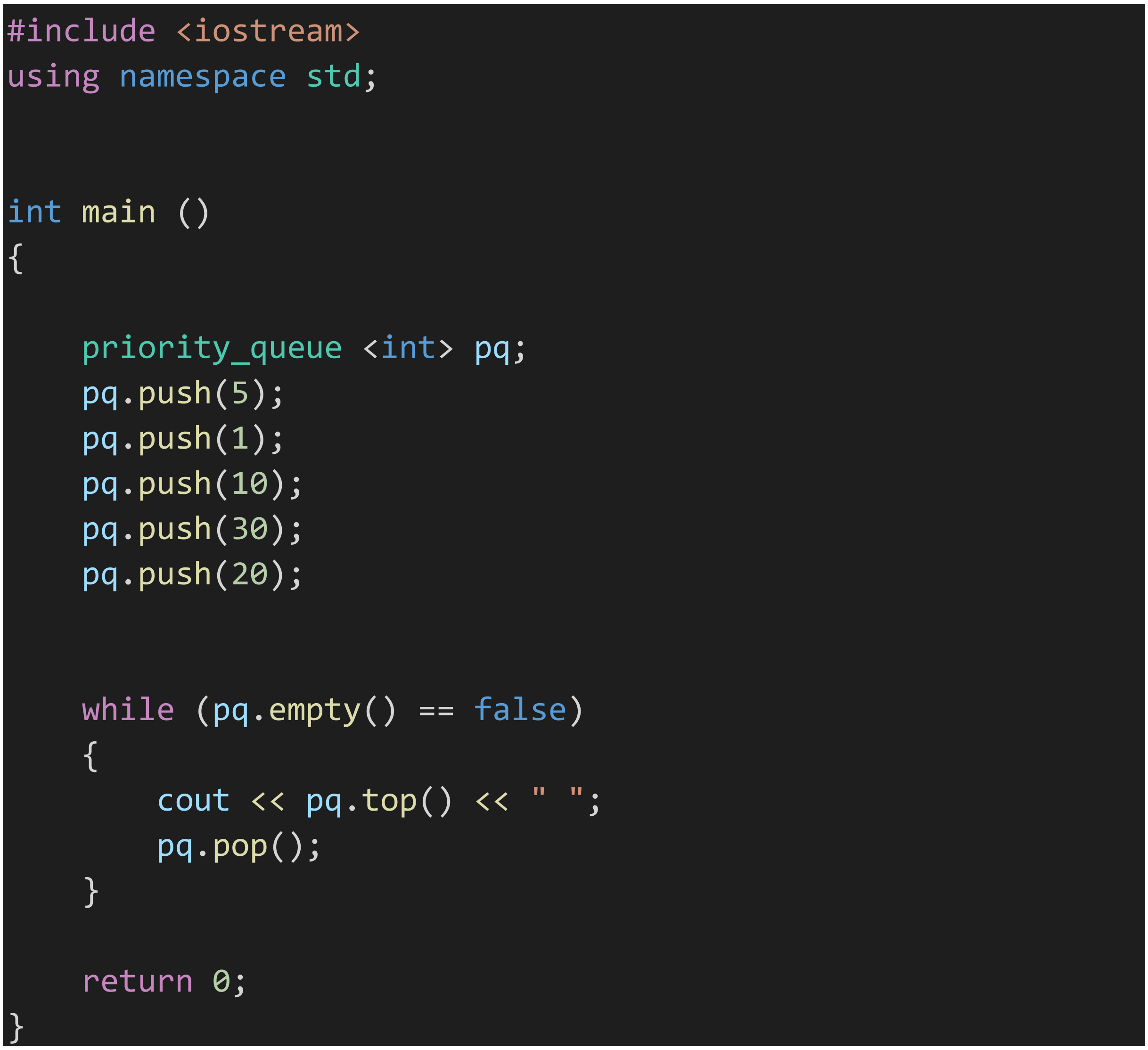
```
                        { 0, 0, 0, 0, 0, 2, 0, 1, 6 },
                        { 8, 11, 0, 0, 0, 0, 1, 0, 7 },
                        { 0, 0, 2, 0, 0, 0, 6, 7, 0 } };

    dijkstra(graph, 0);

    return 0;
}
```

Output:



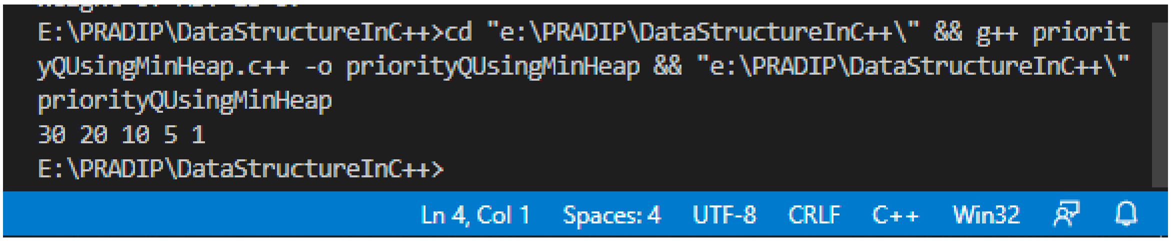
```
E:\PRADIP\DataStructureInC++>cd "e:\PRADIP\DataStructureInC++\" && g++ dijkstr
a.c++ -o dijkstra && "e:\PRADIP\DataStructureInC++\"dijkstra
Vertex           Distance from Source
0                0
1                4
2                12
3                19
4                21
5                11
6                9
7                8
8                14

E:\PRADIP\DataStructureInC++>
```

Priority Queue using Min Heap



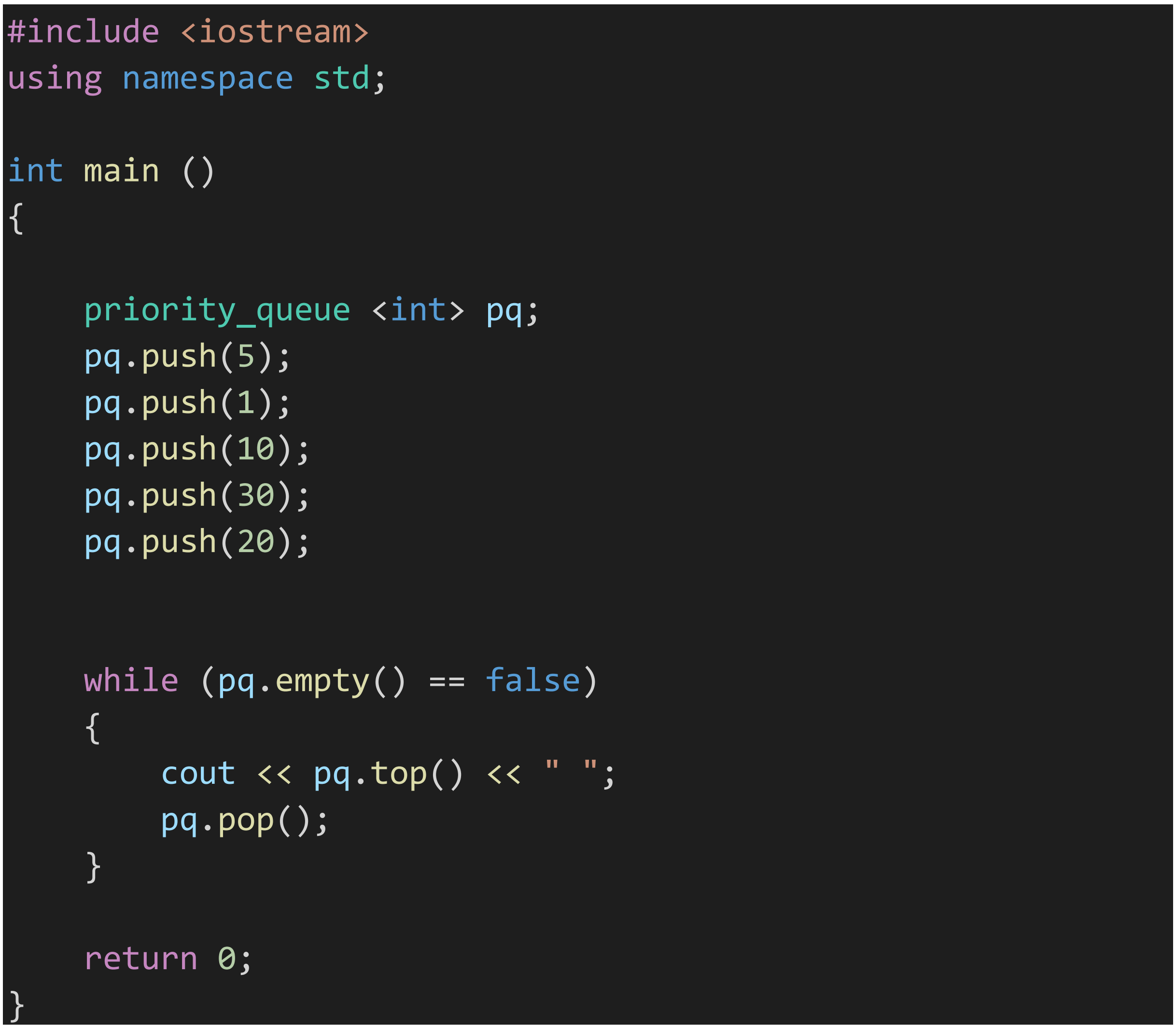
```cpp
#include <iostream>
using namespace std;


int main ()
{

    priority_queue <int> pq;
    pq.push(5);
    pq.push(1);
    pq.push(10);
    pq.push(30);
    pq.push(20);


    while (pq.empty() == false)
    {
        cout << pq.top() << " ";
        pq.pop();
    }

    return 0;
}
```

Output:



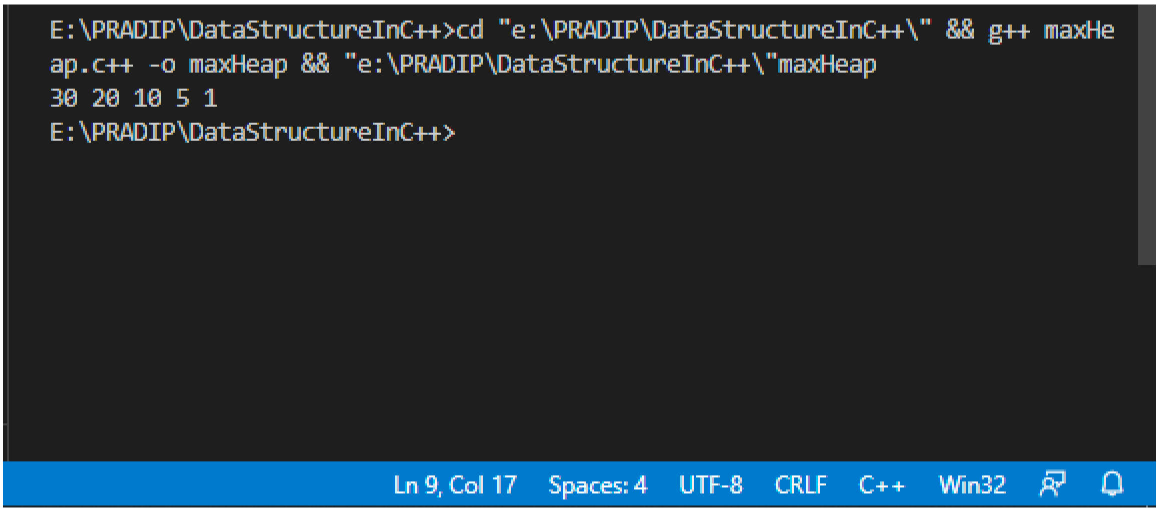
```
E:\PRADIP\DataStructureInC++>cd "e:\PRADIP\DataStructureInC++\" && g++ priorit
yQUsingMinHeap.c++ -o priorityQUsingMinHeap && "e:\PRADIP\DataStructureInC++\"
priorityQUsingMinHeap
30 20 10 5 1
E:\PRADIP\DataStructureInC++>
```

Max Heap



```cpp
#include <iostream>
using namespace std;

int main ()
{

    priority_queue <int> pq;
    pq.push(5);
    pq.push(1);
    pq.push(10);
    pq.push(30);
    pq.push(20);


    while (pq.empty() == false)
    {
        cout << pq.top() << " ";
        pq.pop();
    }

    return 0;
}
```

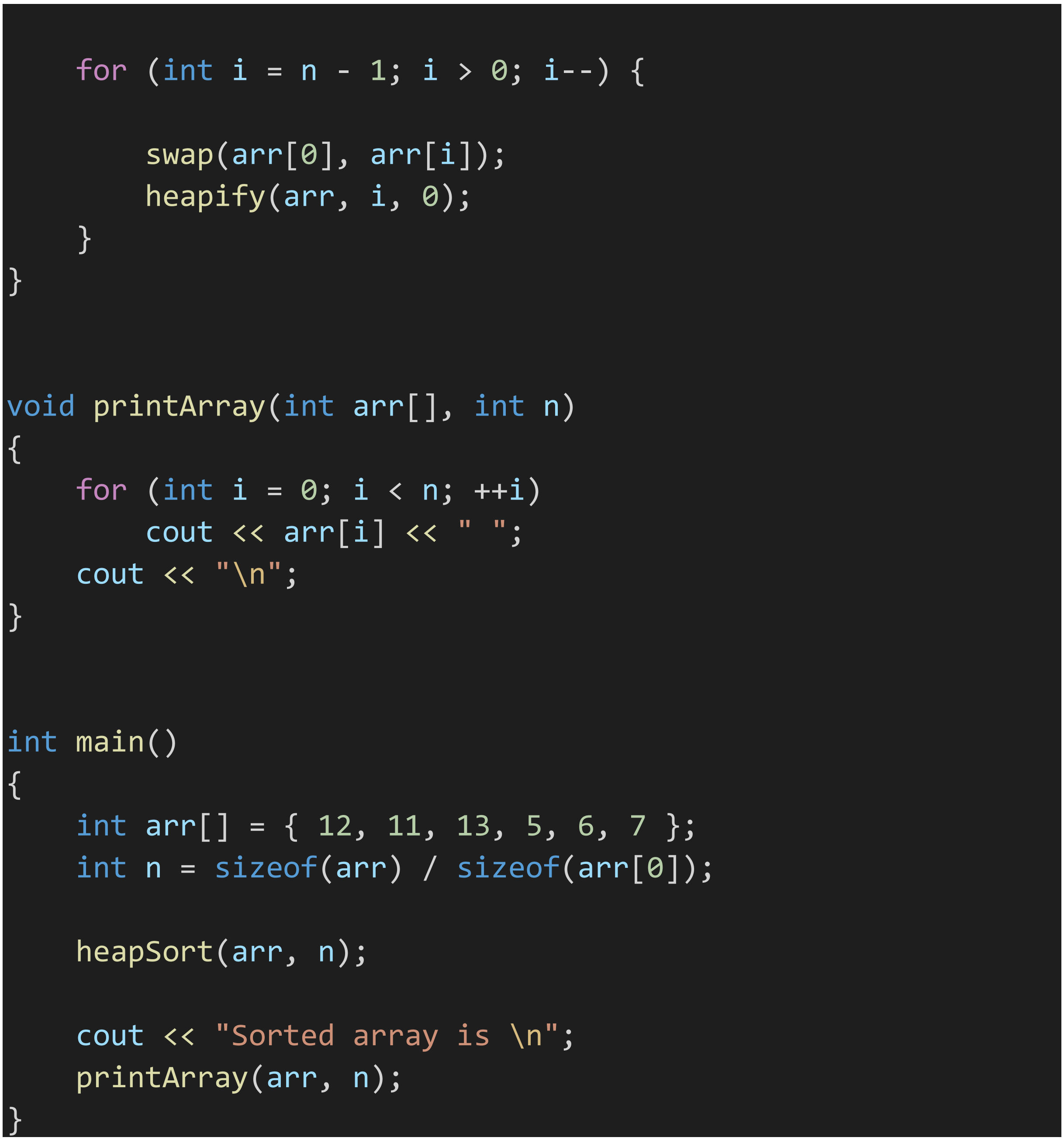
Output:



```
E:\PRADIP\DataStructureInC++>cd "e:\PRADIP\DataStructureInC++\" && g++ maxHe
ap.c++ -o maxHeap && "e:\PRADIP\DataStructureInC++\"maxHeap
30 20 10 5 1
E:\PRADIP\DataStructureInC++>
```

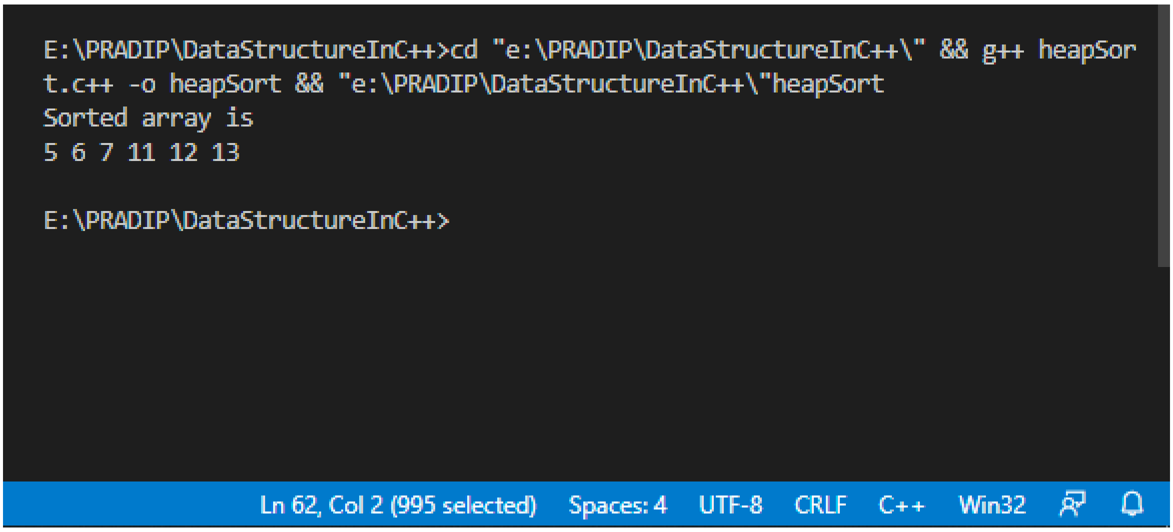
Heap sort

```
#include <iostream>

using namespace std;


void heapify(int arr[], int n, int i)
{
    int largest = i;
    int l = 2 * i + 1;
    int r = 2 * i + 2;


    if (l < n && arr[l] > arr[largest])
        largest = l;


    if (r < n && arr[r] > arr[largest])
        largest = r;


    if (largest != i) {
        swap(arr[i], arr[largest]);


        heapify(arr, n, largest);
    }
}


void heapSort(int arr[], int n)
{

    for (int i = n / 2 - 1; i >= 0; i--)
        heapify(arr, n, i);
```



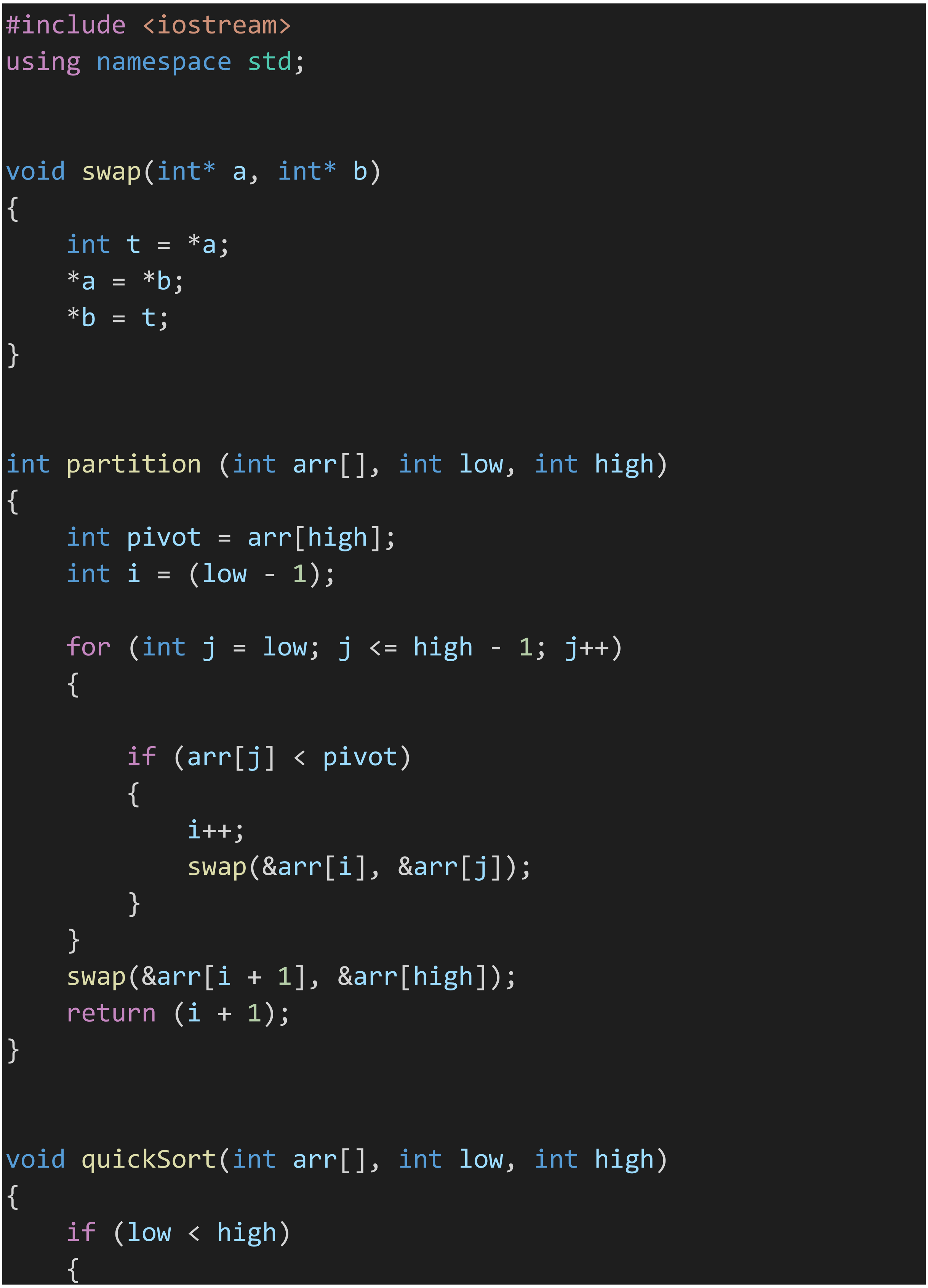
```cpp
    for (int i = n - 1; i > 0; i--) {

        swap(arr[0], arr[i]);
        heapify(arr, i, 0);
    }
}


void printArray(int arr[], int n)
{
    for (int i = 0; i < n; ++i)
        cout << arr[i] << " ";
    cout << "\n";
}


int main()
{
    int arr[] = { 12, 11, 13, 5, 6, 7 };
    int n = sizeof(arr) / sizeof(arr[0]);

    heapSort(arr, n);

    cout << "Sorted array is \n";
    printArray(arr, n);
}
```

Output:

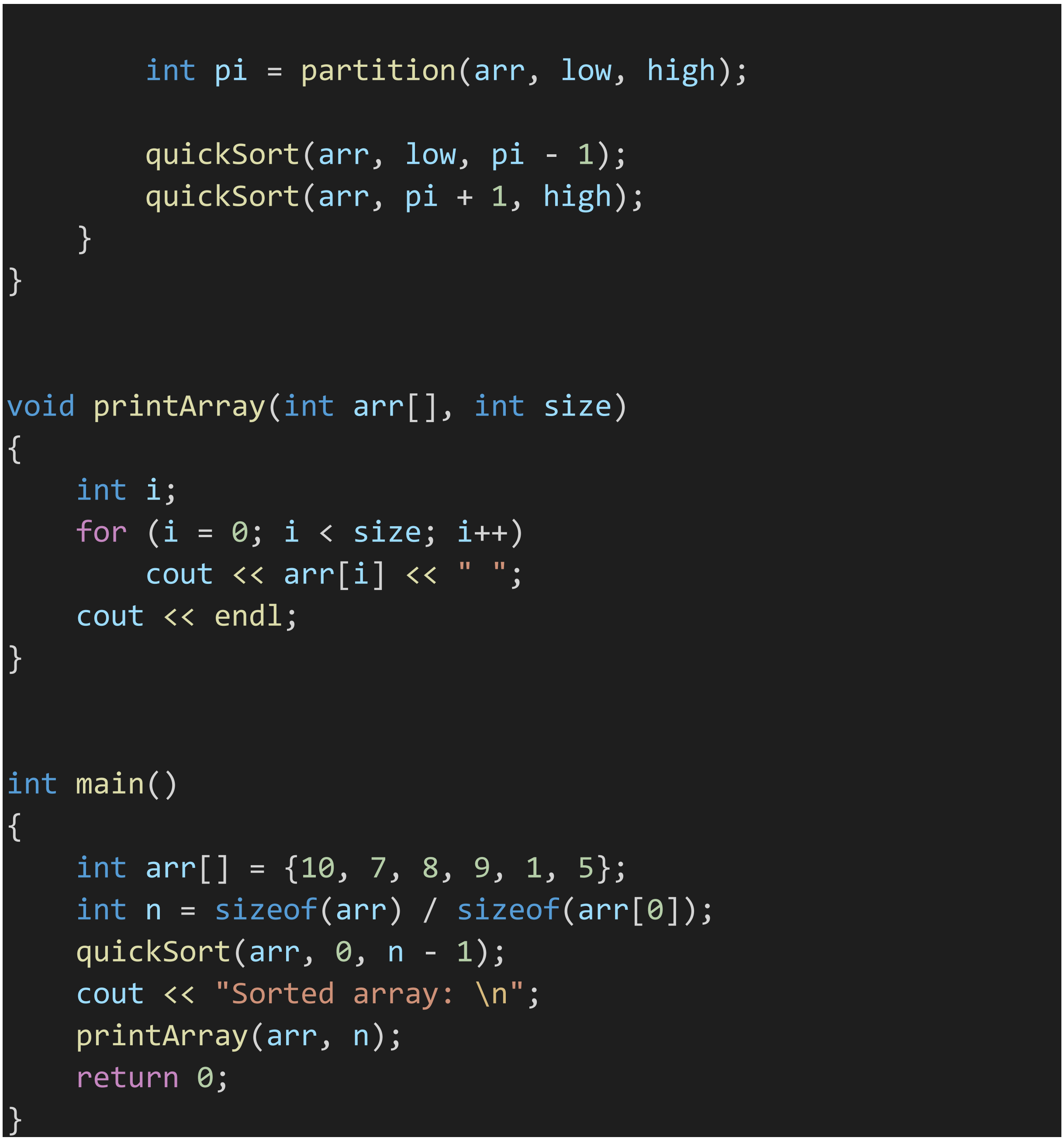


```
E:\PRADIP\DataStructureInC++>cd "e:\PRADIP\DataStructureInC++\" && g++ heapSor
t.c++ -o heapSort && "e:\PRADIP\DataStructureInC++\"heapSort
Sorted array is
5 6 7 11 12 13

E:\PRADIP\DataStructureInC++>
```

Quick Sort

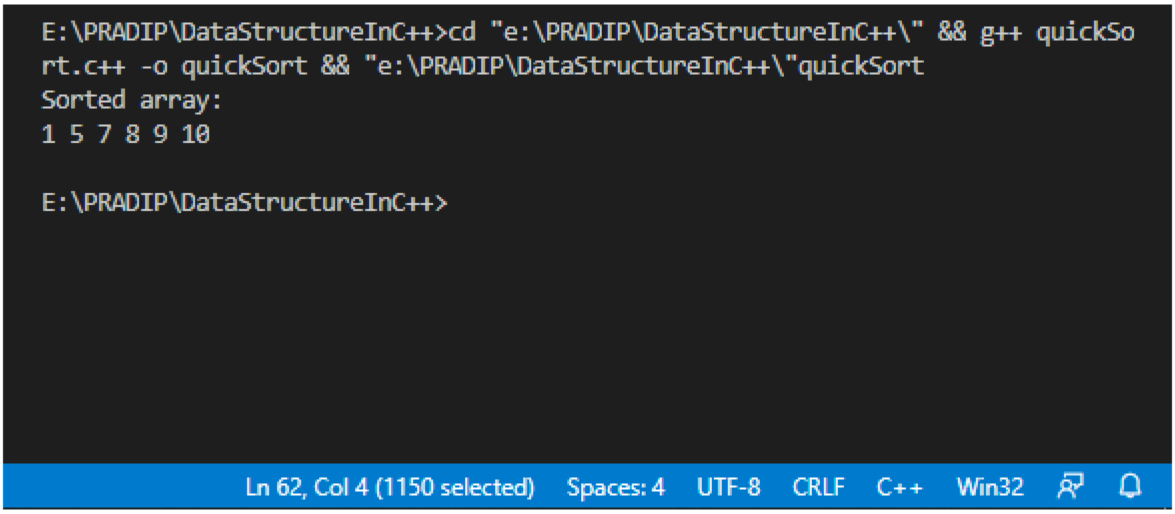


```
#include <iostream>
using namespace std;


void swap(int* a, int* b)
{
    int t = *a;
    *a = *b;
    *b = t;
}


int partition (int arr[], int low, int high)
{
    int pivot = arr[high];
    int i = (low - 1);

    for (int j = low; j <= high - 1; j++)
    {

        if (arr[j] < pivot)
        {
            i++;
            swap(&arr[i], &arr[j]);
        }
    }
    swap(&arr[i + 1], &arr[high]);
    return (i + 1);
}


void quickSort(int arr[], int low, int high)
{
    if (low < high)
    {
```



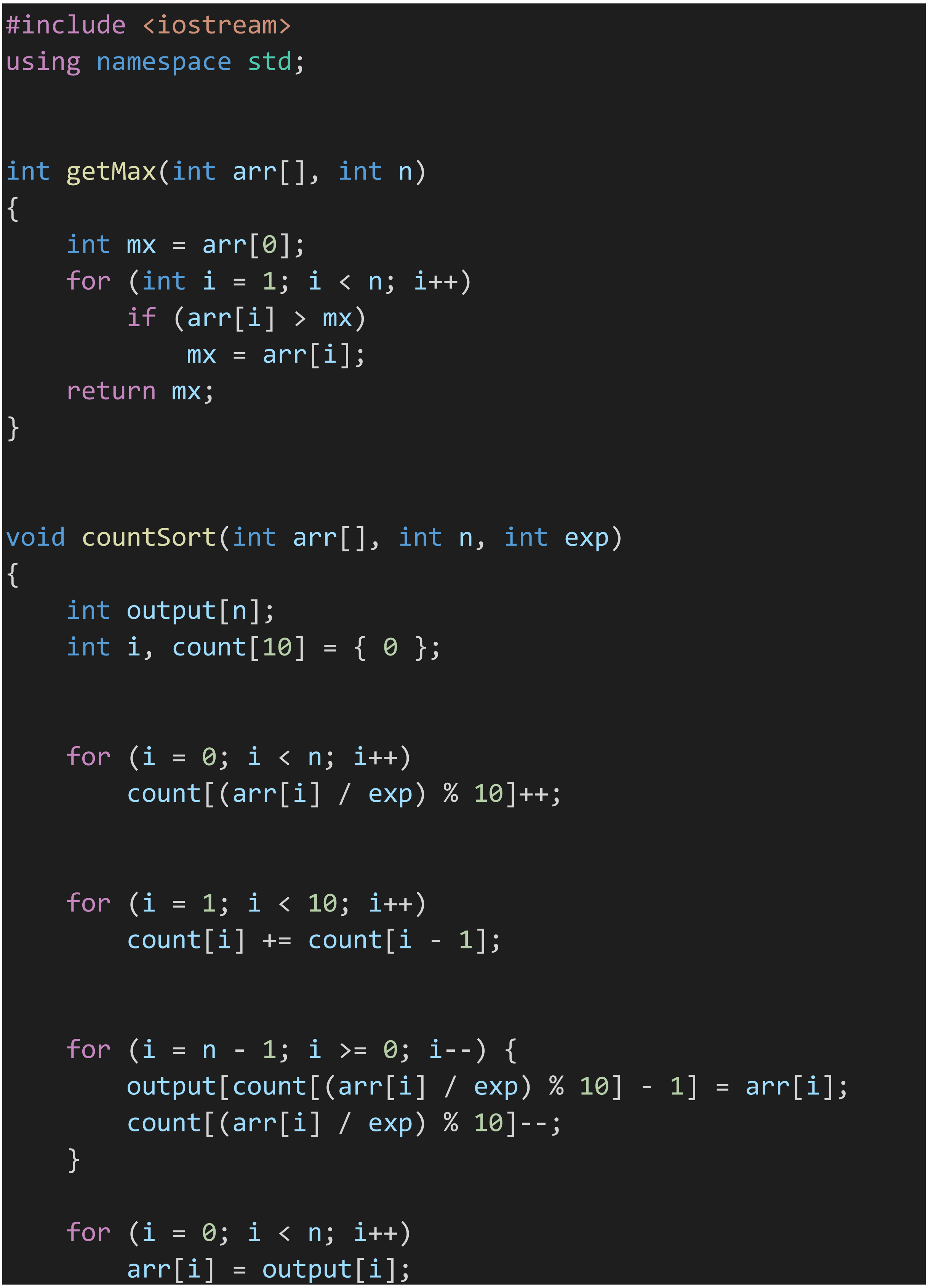
```
        int pi = partition(arr, low, high);

        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}


void printArray(int arr[], int size)
{
    int i;
    for (i = 0; i < size; i++)
        cout << arr[i] << " ";
    cout << endl;
}


int main()
{
    int arr[] = {10, 7, 8, 9, 1, 5};
    int n = sizeof(arr) / sizeof(arr[0]);
    quickSort(arr, 0, n - 1);
    cout << "Sorted array: \n";
    printArray(arr, n);
    return 0;
}
```

Output:

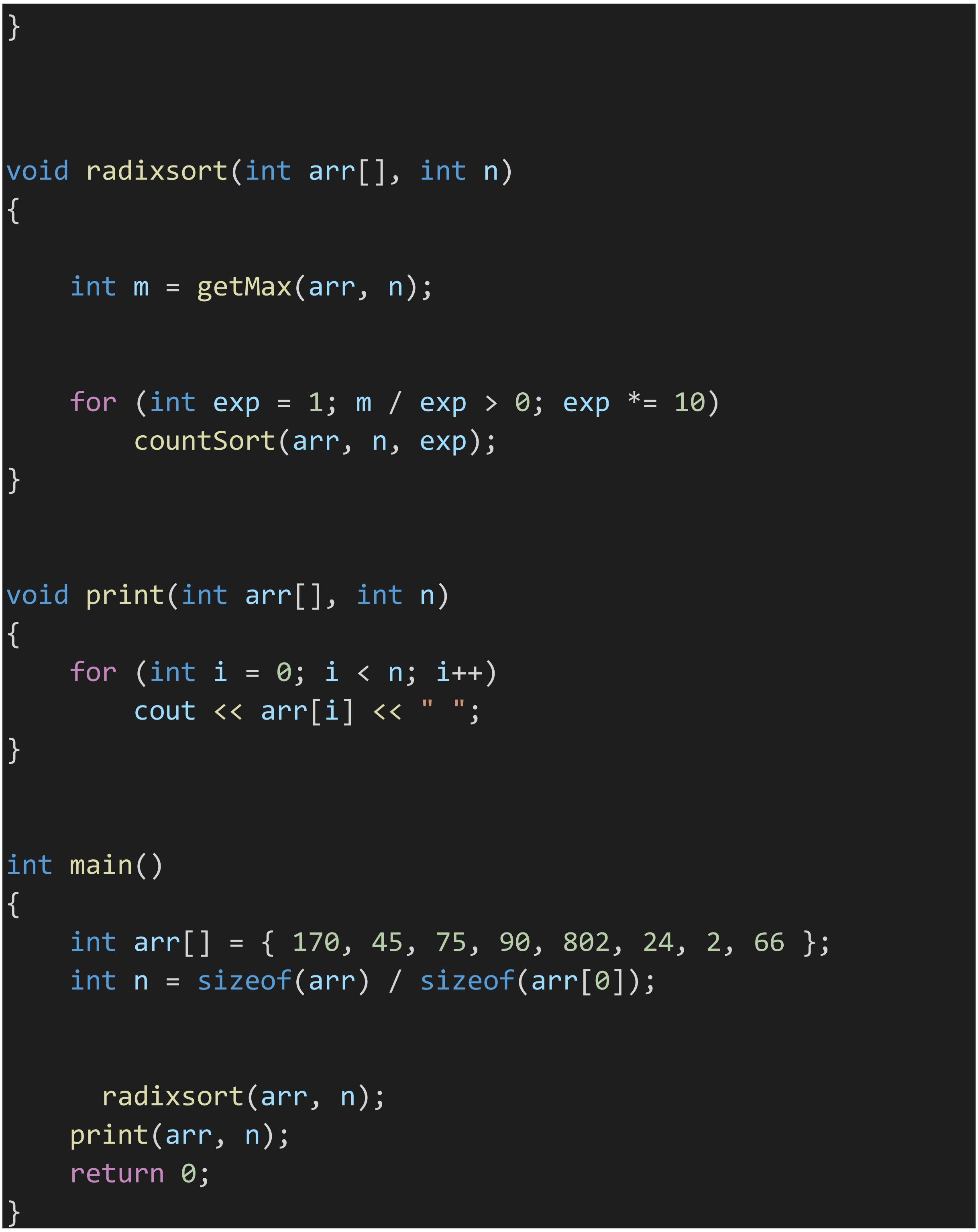


```
E:\PRADIP\DataStructureInC++>cd "e:\PRADIP\DataStructureInC++\" && g++ quickSo
rt.c++ -o quickSort && "e:\PRADIP\DataStructureInC++\"quickSort
Sorted array:
1 5 7 8 9 10

E:\PRADIP\DataStructureInC++>
```

Radix Sort

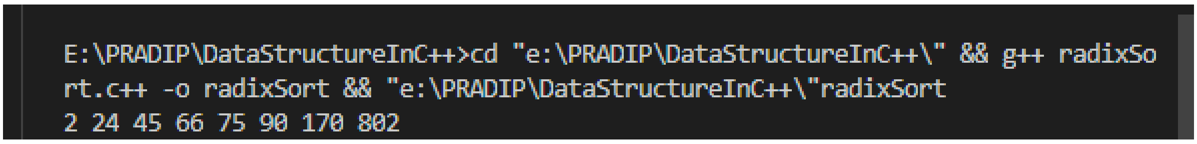


```cpp
#include <iostream>
using namespace std;


int getMax(int arr[], int n)
{
    int mx = arr[0];
    for (int i = 1; i < n; i++)
        if (arr[i] > mx)
            mx = arr[i];
    return mx;
}


void countSort(int arr[], int n, int exp)
{
    int output[n];
    int i, count[10] = { 0 };


    for (i = 0; i < n; i++)
        count[(arr[i] / exp) % 10]++;


    for (i = 1; i < 10; i++)
        count[i] += count[i - 1];


    for (i = n - 1; i >= 0; i--) {
        output[count[(arr[i] / exp) % 10] - 1] = arr[i];
        count[(arr[i] / exp) % 10]--;
    }

    for (i = 0; i < n; i++)
        arr[i] = output[i];
```



```
}


void radixsort(int arr[], int n)
{

    int m = getMax(arr, n);


    for (int exp = 1; m / exp > 0; exp *= 10)
        countSort(arr, n, exp);
}


void print(int arr[], int n)
{
    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";
}


int main()
{
    int arr[] = { 170, 45, 75, 90, 802, 24, 2, 66 };
    int n = sizeof(arr) / sizeof(arr[0]);


      radixsort(arr, n);
    print(arr, n);
    return 0;
}
```

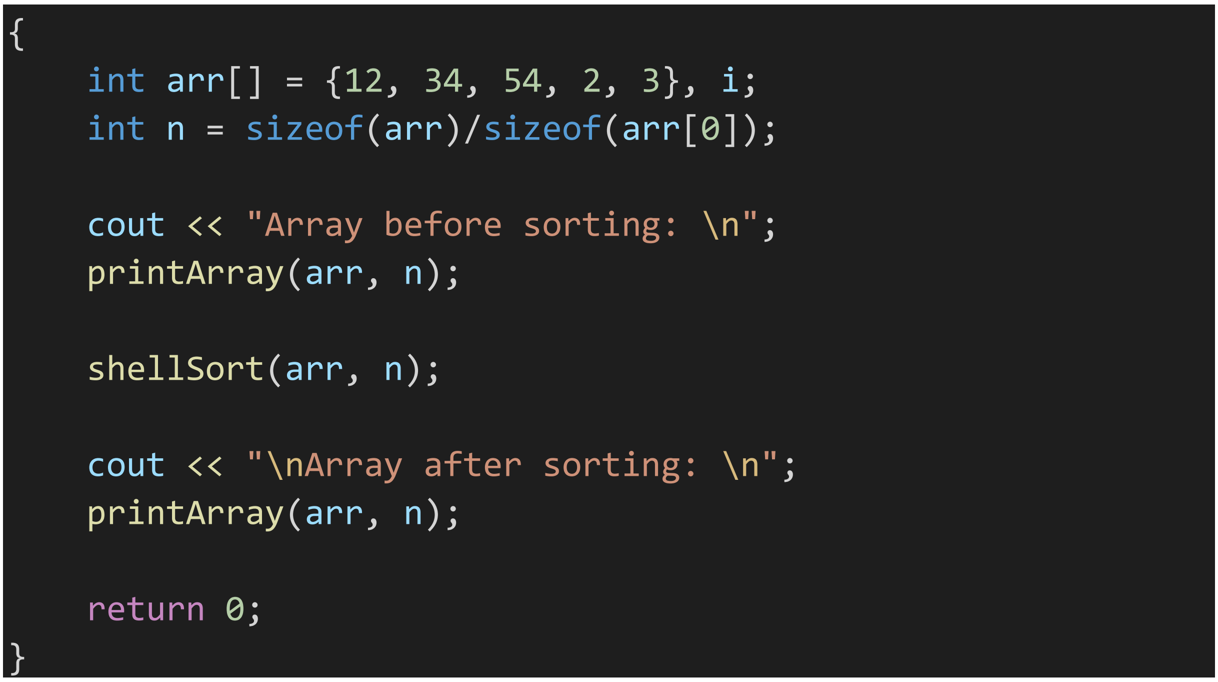
Output:



```
E:\PRADIP\DataStructureInC++>cd "e:\PRADIP\DataStructureInC++\" && g++ radixSo
rt.c++ -o radixSort && "e:\PRADIP\DataStructureInC++\"radixSort
2 24 45 66 75 90 170 802
```

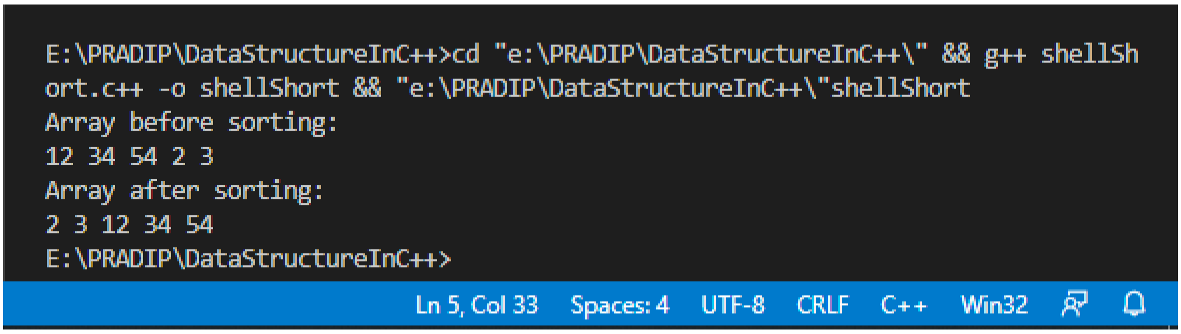
Shell sort

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


int shellSort(int arr[], int n)
{

    for (int gap = n/2; gap > 0; gap /= 2)
    {

        for (int i = gap; i < n; i += 1)
        {

            int temp = arr[i];


            int j;
            for (j = i; j >= gap && arr[j - gap] > temp; j -
= gap)
                arr[j] = arr[j - gap];


            arr[j] = temp;
        }
    }
    return 0;
}

void printArray(int arr[], int n)
{
    for (int i=0; i<n; i++)
        cout << arr[i] << " ";
}

int main()
```



```
{
    int arr[] = {12, 34, 54, 2, 3}, i;
    int n = sizeof(arr)/sizeof(arr[0]);

    cout << "Array before sorting: \n";
    printArray(arr, n);

    shellSort(arr, n);

    cout << "\nArray after sorting: \n";
    printArray(arr, n);

    return 0;
}
```

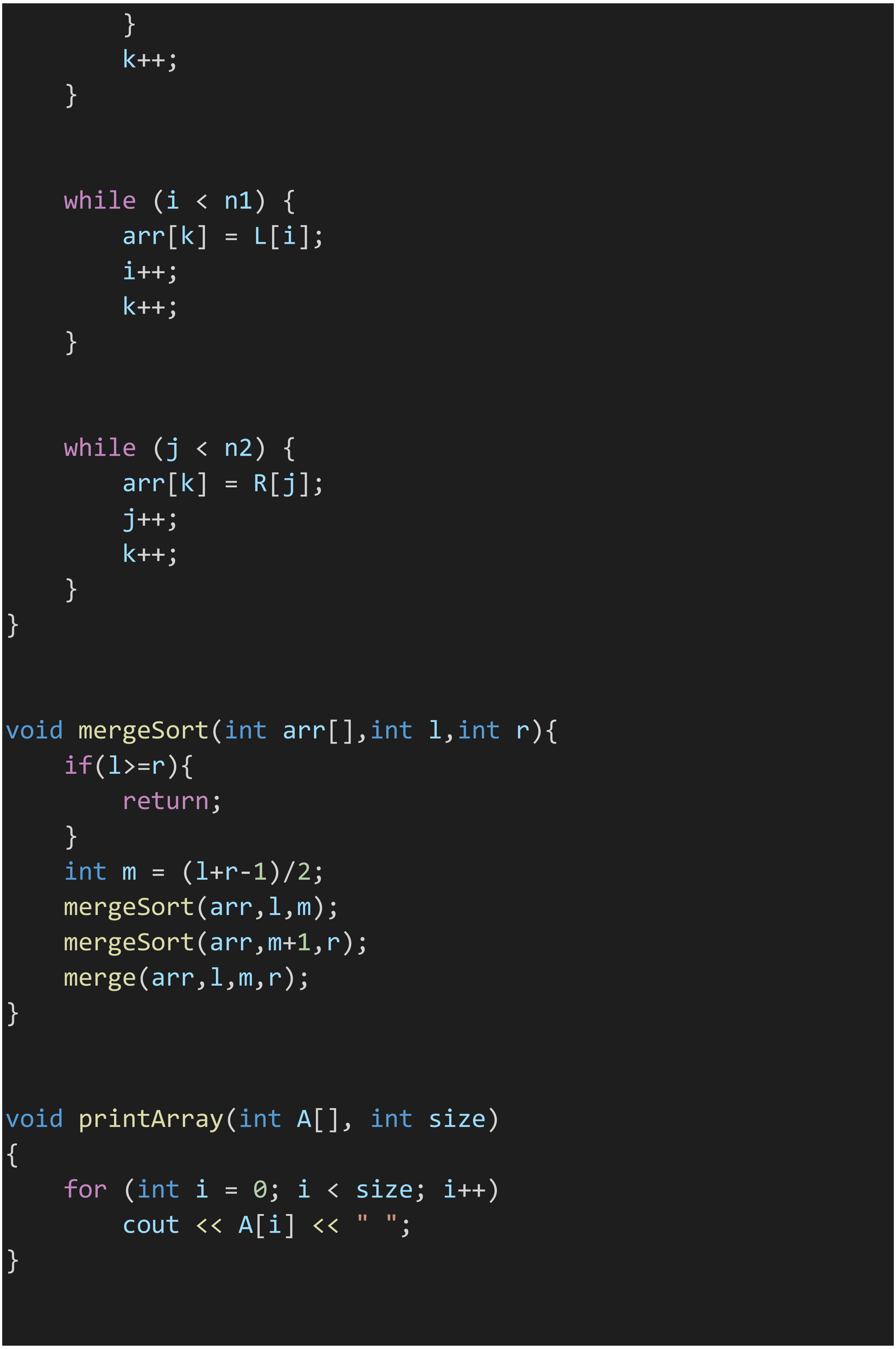
Output:



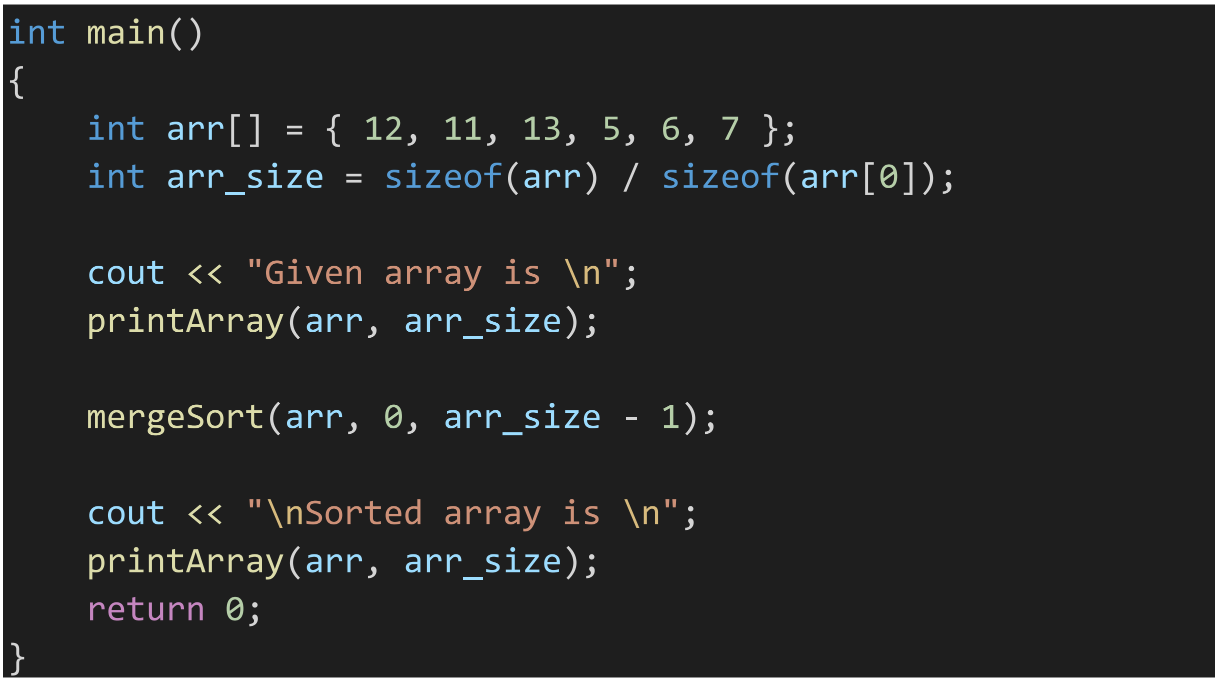
```
E:\PRADIP\DataStructureInC++>cd "e:\PRADIP\DataStructureInC++\" && g++ shellSh
ort.c++ -o shellShort && "e:\PRADIP\DataStructureInC++\"shellShort
Array before sorting:
12 34 54 2 3
Array after sorting:
2 3 12 34 54
E:\PRADIP\DataStructureInC++>
```

Merge sort

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


void merge(int arr[], int l, int m, int r)
{
    int n1 = m - l + 1;
    int n2 = r - m;


    int L[n1], R[n2];


    for (int i = 0; i < n1; i++)
        L[i] = arr[l + i];
    for (int j = 0; j < n2; j++)
        R[j] = arr[m + 1 + j];


    int i = 0;


    int j = 0;


    int k = l;

    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k] = L[i];
            i++;
        }
        else {
            arr[k] = R[j];
            j++;
```

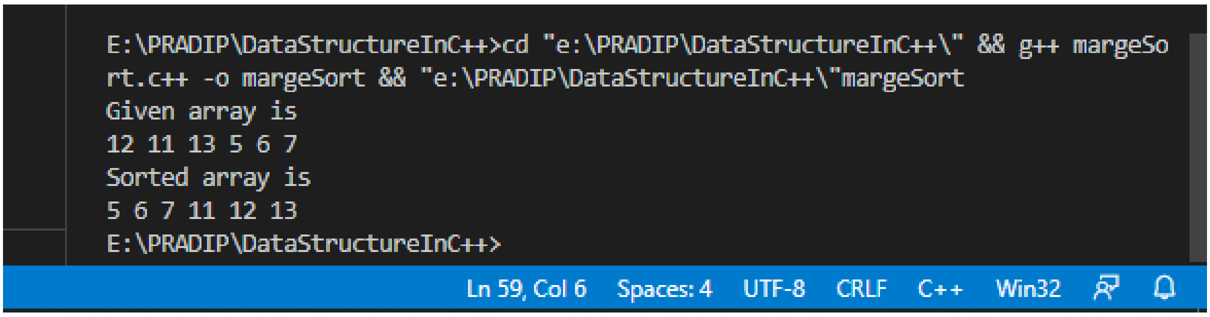


```
        }
        k++;
    }


    while (i < n1) {
        arr[k] = L[i];
        i++;
        k++;
    }


    while (j < n2) {
        arr[k] = R[j];
        j++;
        k++;
    }
}


void mergeSort(int arr[],int l,int r){
    if(l>=r){
        return;
    }
    int m = (l+r-1)/2;
    mergeSort(arr,l,m);
    mergeSort(arr,m+1,r);
    merge(arr,l,m,r);
}


void printArray(int A[], int size)
{
    for (int i = 0; i < size; i++)
        cout << A[i] << " ";
}
```



```
int main()
{
    int arr[] = { 12, 11, 13, 5, 6, 7 };
    int arr_size = sizeof(arr) / sizeof(arr[0]);

    cout << "Given array is \n";
    printArray(arr, arr_size);

    mergeSort(arr, 0, arr_size - 1);

    cout << "\nSorted array is \n";
    printArray(arr, arr_size);
    return 0;
}
```

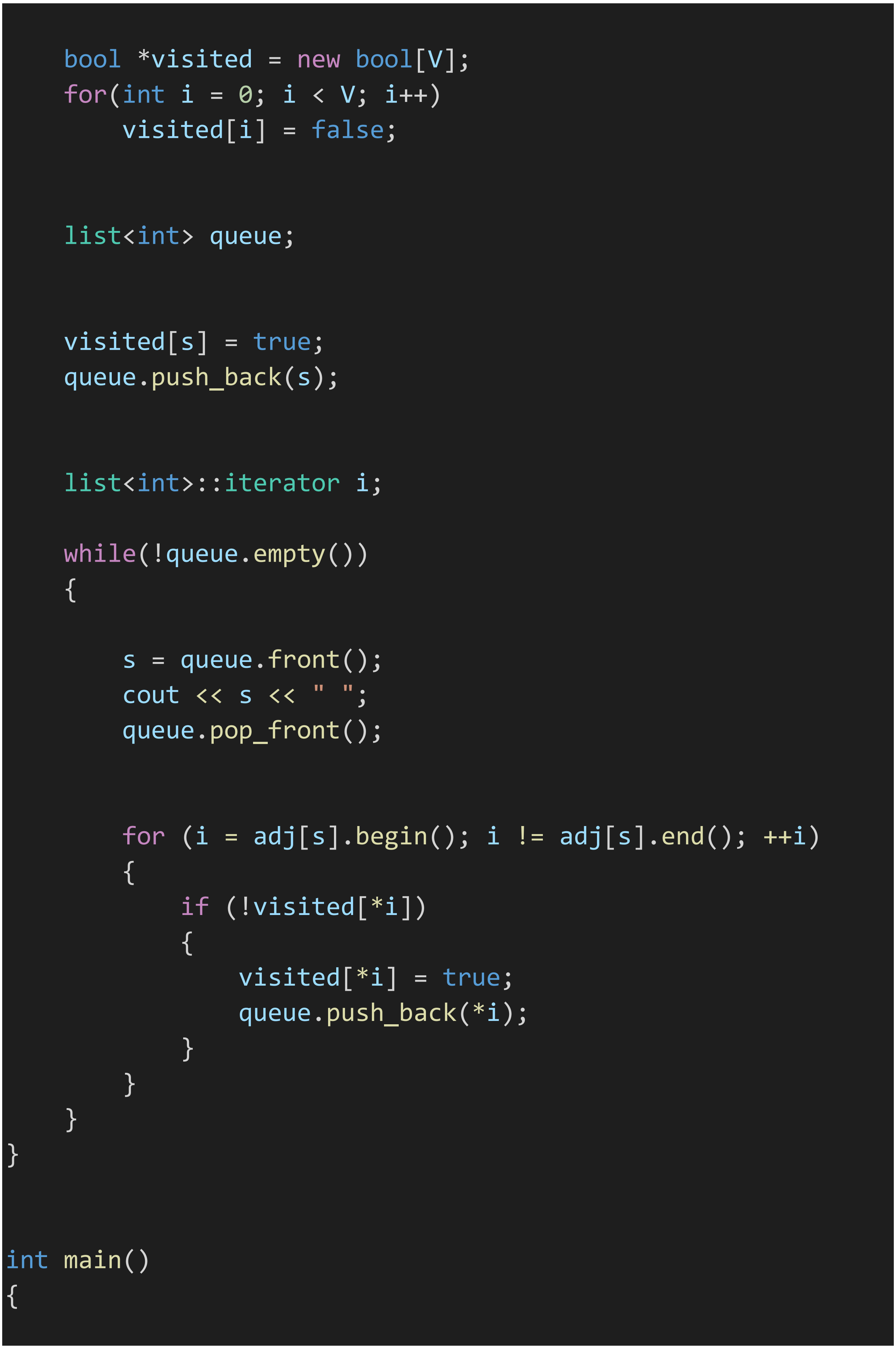
Output:



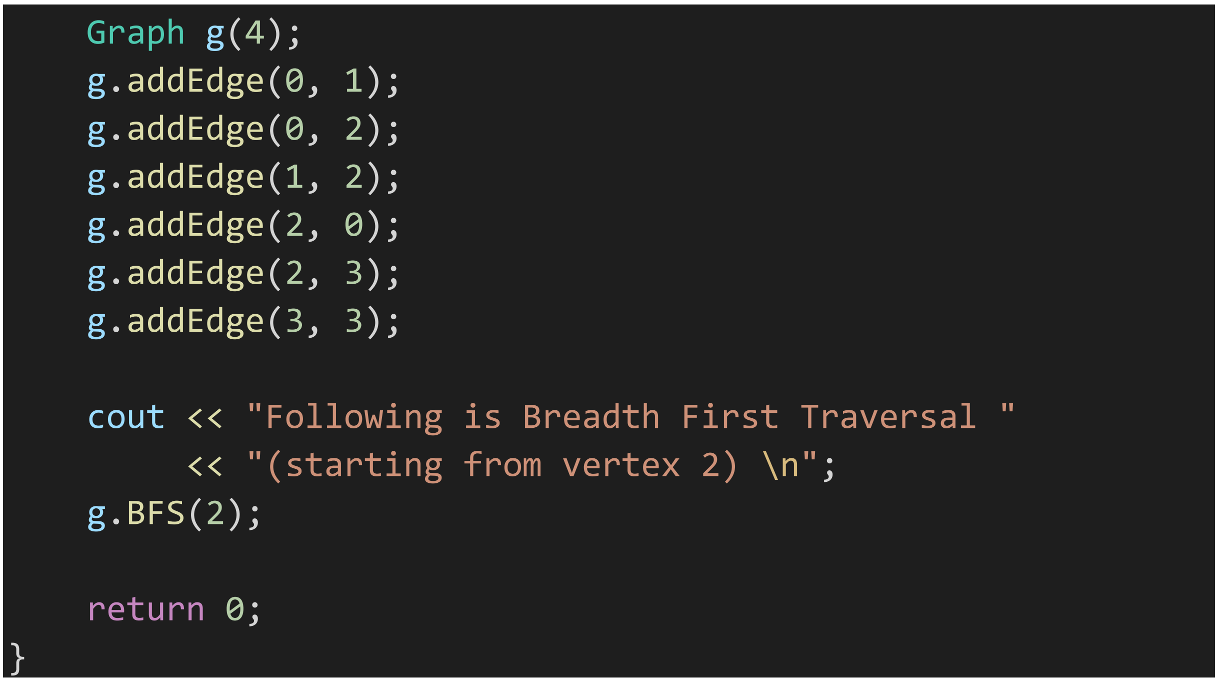
```
E:\PRADIP\DataStructureInC++>cd "e:\PRADIP\DataStructureInC++\" && g++ margeSo
rt.c++ -o margeSort && "e:\PRADIP\DataStructureInC++\"margeSort
Given array is
12 11 13 5 6 7
Sorted array is
5 6 7 11 12 13
E:\PRADIP\DataStructureInC++>
```

BFS

```
#include<iostream>
#include <list>

using namespace std;



class Graph
{
    int V;


    list<int> *adj;
public:
    Graph(int V);


    void addEdge(int v, int w);


    void BFS(int s);
};

Graph::Graph(int V)
{
    this->V = V;
    adj = new list<int>[V];
}

void Graph::addEdge(int v, int w)
{
    adj[v].push_back(w);
}

void Graph::BFS(int s)
{
```

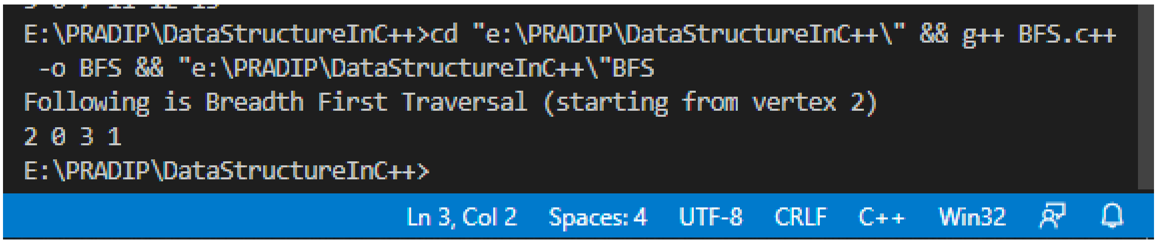


```
    bool *visited = new bool[V];
    for(int i = 0; i < V; i++)
        visited[i] = false;


    list<int> queue;


    visited[s] = true;
    queue.push_back(s);


    list<int>::iterator i;

    while(!queue.empty())
    {

        s = queue.front();
        cout << s << " ";
        queue.pop_front();


        for (i = adj[s].begin(); i != adj[s].end(); ++i)
        {
            if (!visited[*i])
            {
                visited[*i] = true;
                queue.push_back(*i);
            }
        }
    }
}


int main()
{
```



```
    Graph g(4);
    g.addEdge(0, 1);
    g.addEdge(0, 2);
    g.addEdge(1, 2);
    g.addEdge(2, 0);
    g.addEdge(2, 3);
    g.addEdge(3, 3);

    cout << "Following is Breadth First Traversal "
         << "(starting from vertex 2) \n";
    g.BFS(2);

    return 0;
}
```

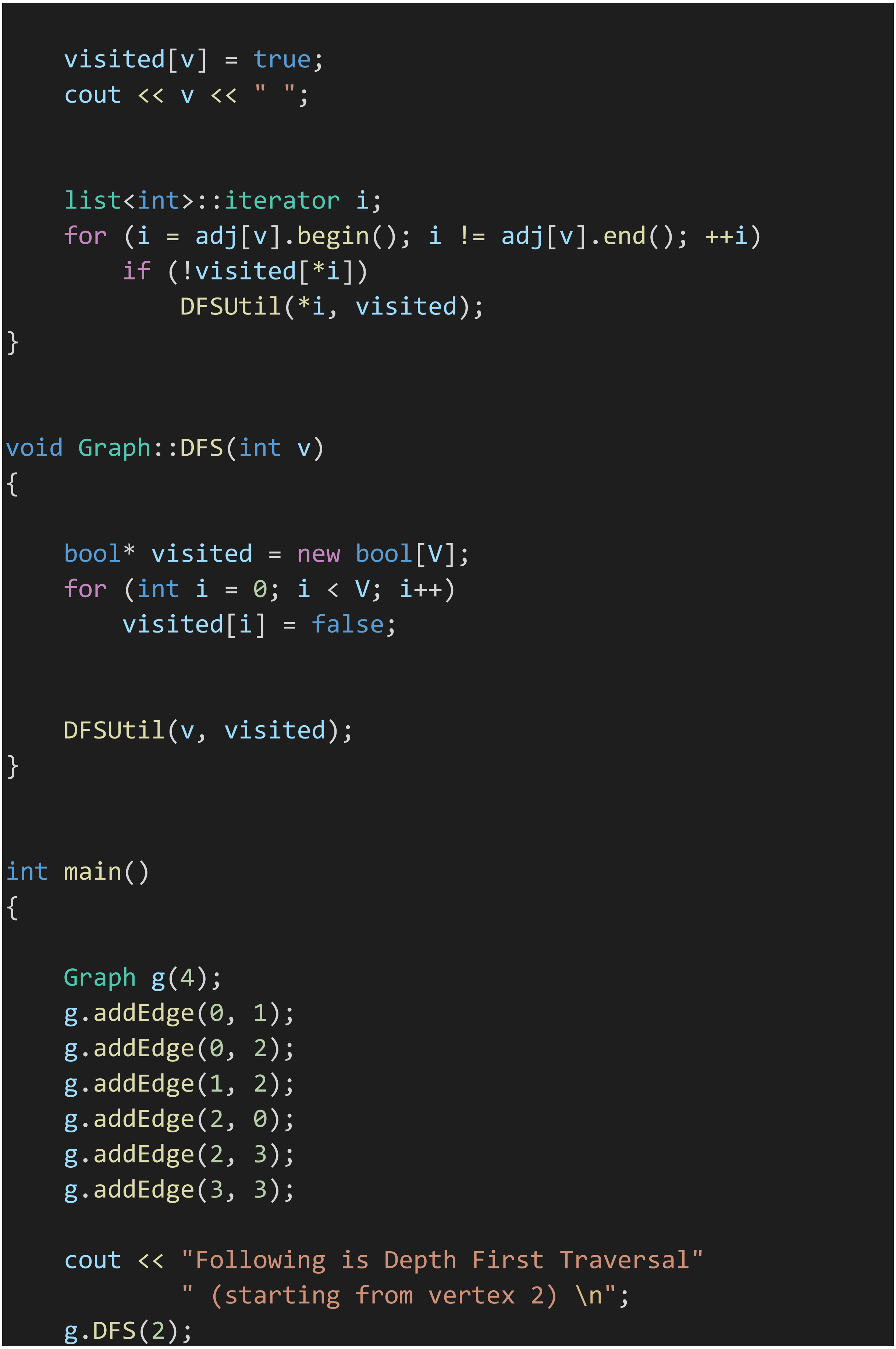
Output:



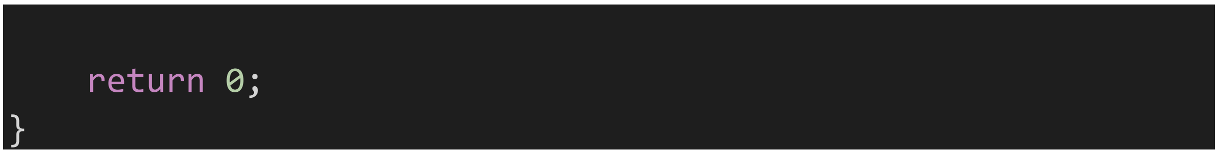
```
E:\PRADIP\DataStructureInC++>cd "e:\PRADIP\DataStructureInC++\" && g++ BFS.c++
 -o BFS && "e:\PRADIP\DataStructureInC++\"BFS
Following is Breadth First Traversal (starting from vertex 2)
2 0 3 1
E:\PRADIP\DataStructureInC++>
```

DFS

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


class Graph {
    int V;


    list<int>* adj;


    void DFSUtil(int v, bool visited[]);

public:
    Graph(int V);

    void addEdge(int v, int w);


    void DFS(int v);
};

Graph::Graph(int V)
{
    this->V = V;
    adj = new list<int>[V];
}

void Graph::addEdge(int v, int w)
{
    adj[v].push_back(w);
}

void Graph::DFSUtil(int v, bool visited[])
{
```

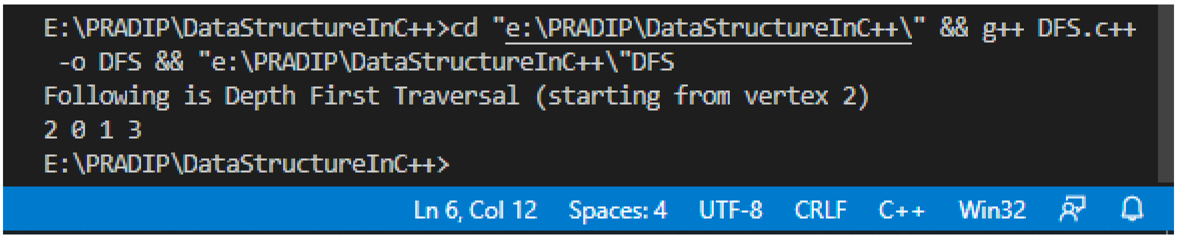


```
    visited[v] = true;
    cout << v << " ";


    list<int>::iterator i;
    for (i = adj[v].begin(); i != adj[v].end(); ++i)
        if (!visited[*i])
            DFSUtil(*i, visited);
}


void Graph::DFS(int v)
{

    bool* visited = new bool[V];
    for (int i = 0; i < V; i++)
        visited[i] = false;


    DFSUtil(v, visited);
}


int main()
{

    Graph g(4);
    g.addEdge(0, 1);
    g.addEdge(0, 2);
    g.addEdge(1, 2);
    g.addEdge(2, 0);
    g.addEdge(2, 3);
    g.addEdge(3, 3);

    cout << "Following is Depth First Traversal"
            " (starting from vertex 2) \n";
    g.DFS(2);
```



```
    return 0;
}
```

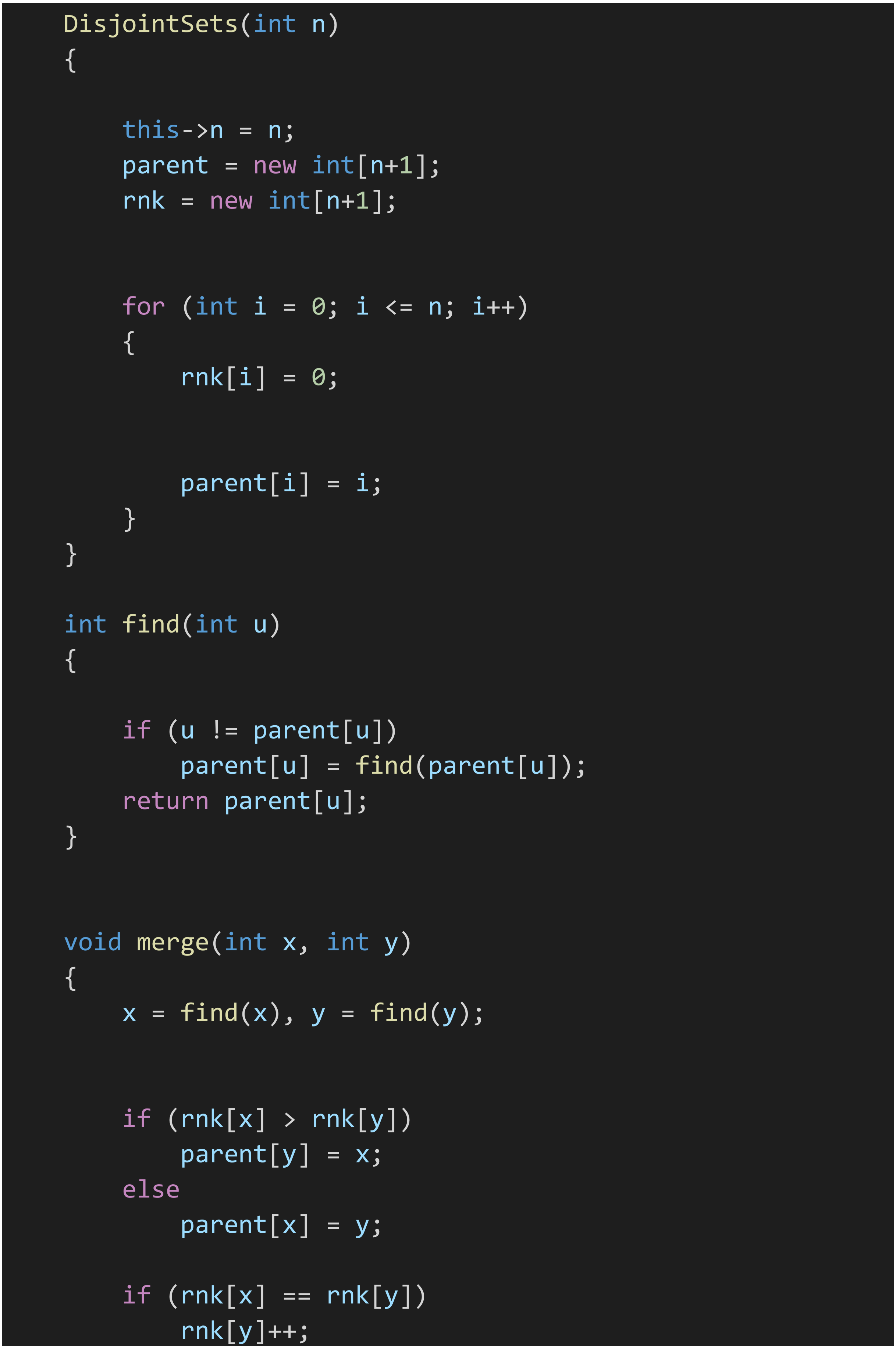
Output:



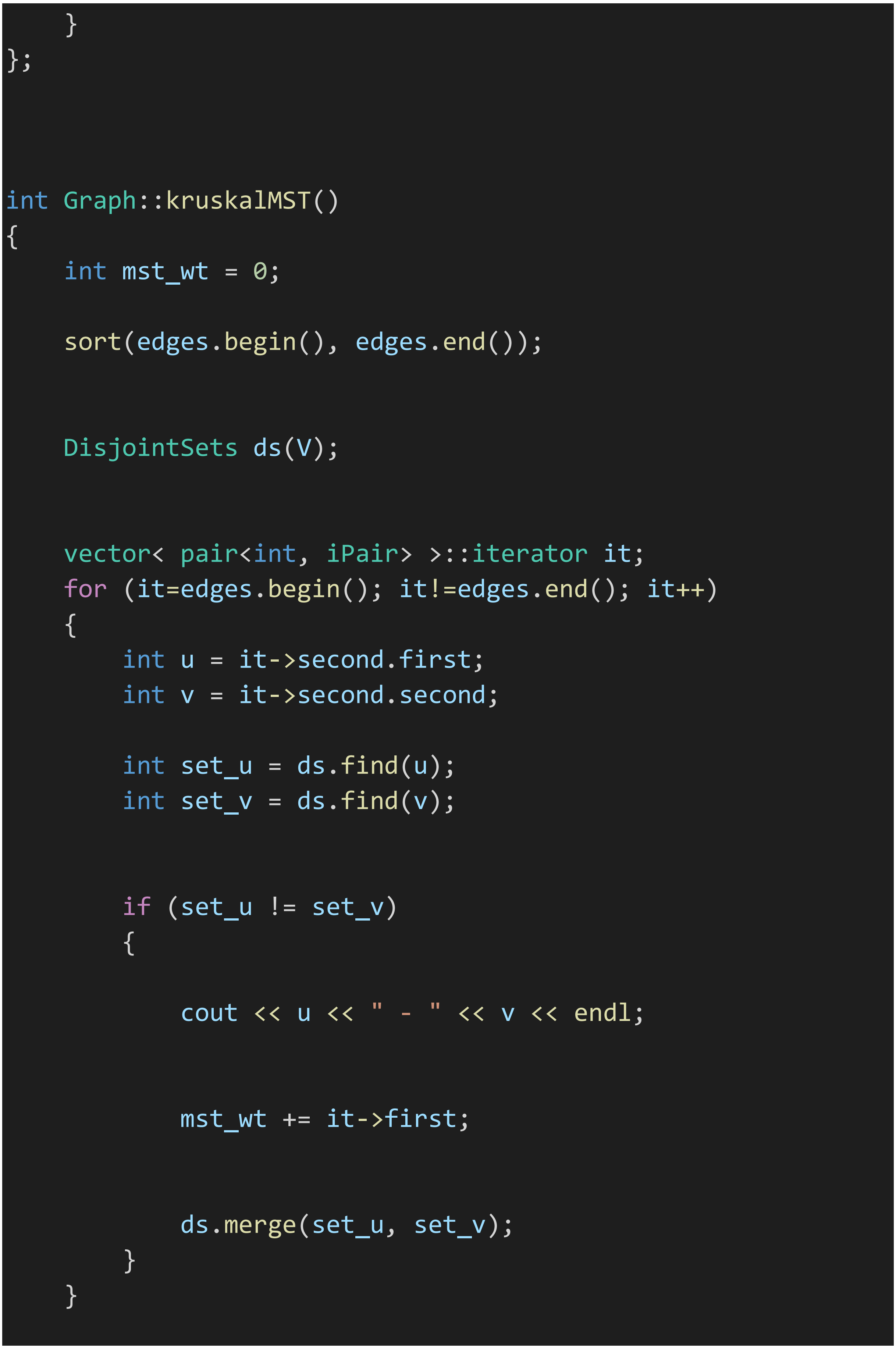
```
E:\PRADIP\DataStructureInC++>cd "e:\PRADIP\DataStructureInC++\" && g++ DFS.c++
 -o DFS && "e:\PRADIP\DataStructureInC++\"DFS
Following is Depth First Traversal (starting from vertex 2)
2 0 1 3
E:\PRADIP\DataStructureInC++>
```

Kruskals

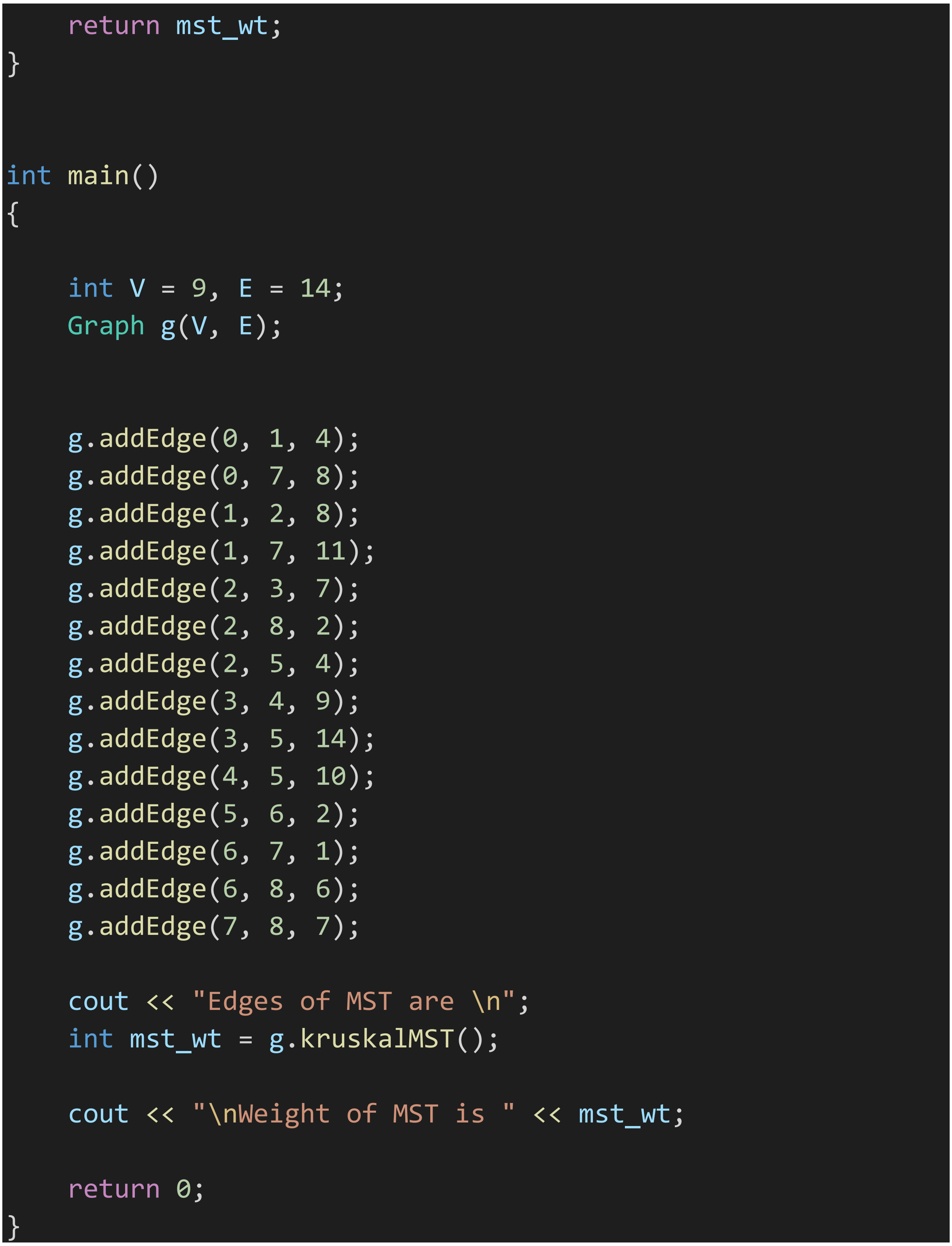
```
#include <iostream>
using namespace std;


typedef  pair<int, int> iPair;


struct Graph
{
    int V, E;
    vector< pair<int, iPair> > edges;


    Graph(int V, int E)
    {
        this->V = V;
        this->E = E;
    }

    void addEdge(int u, int v, int w)
    {
        edges.push_back({w, {u, v}});
    }


    int kruskalMST();
};


struct DisjointSets
{
    int *parent, *rnk;
    int n;
```



```
    DisjointSets(int n)
    {

        this->n = n;
        parent = new int[n+1];
        rnk = new int[n+1];


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


            parent[i] = i;
        }
    }

    int find(int u)
    {

        if (u != parent[u])
            parent[u] = find(parent[u]);
        return parent[u];
    }


    void merge(int x, int y)
    {
        x = find(x), y = find(y);


        if (rnk[x] > rnk[y])
            parent[y] = x;
        else
            parent[x] = y;

        if (rnk[x] == rnk[y])
            rnk[y]++;
```

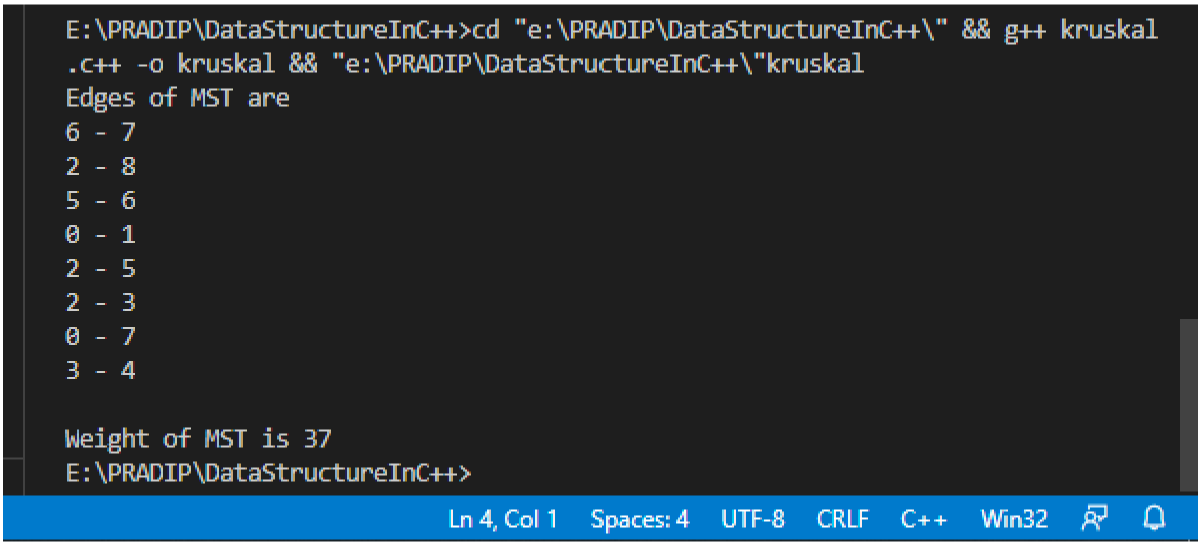


```
    }
};



int Graph::kruskalMST()
{
    int mst_wt = 0;

    sort(edges.begin(), edges.end());


    DisjointSets ds(V);


    vector< pair<int, iPair> >::iterator it;
    for (it=edges.begin(); it!=edges.end(); it++)
    {
        int u = it->second.first;
        int v = it->second.second;

        int set_u = ds.find(u);
        int set_v = ds.find(v);


        if (set_u != set_v)
        {

            cout << u << " - " << v << endl;


            mst_wt += it->first;


            ds.merge(set_u, set_v);
        }
    }
```



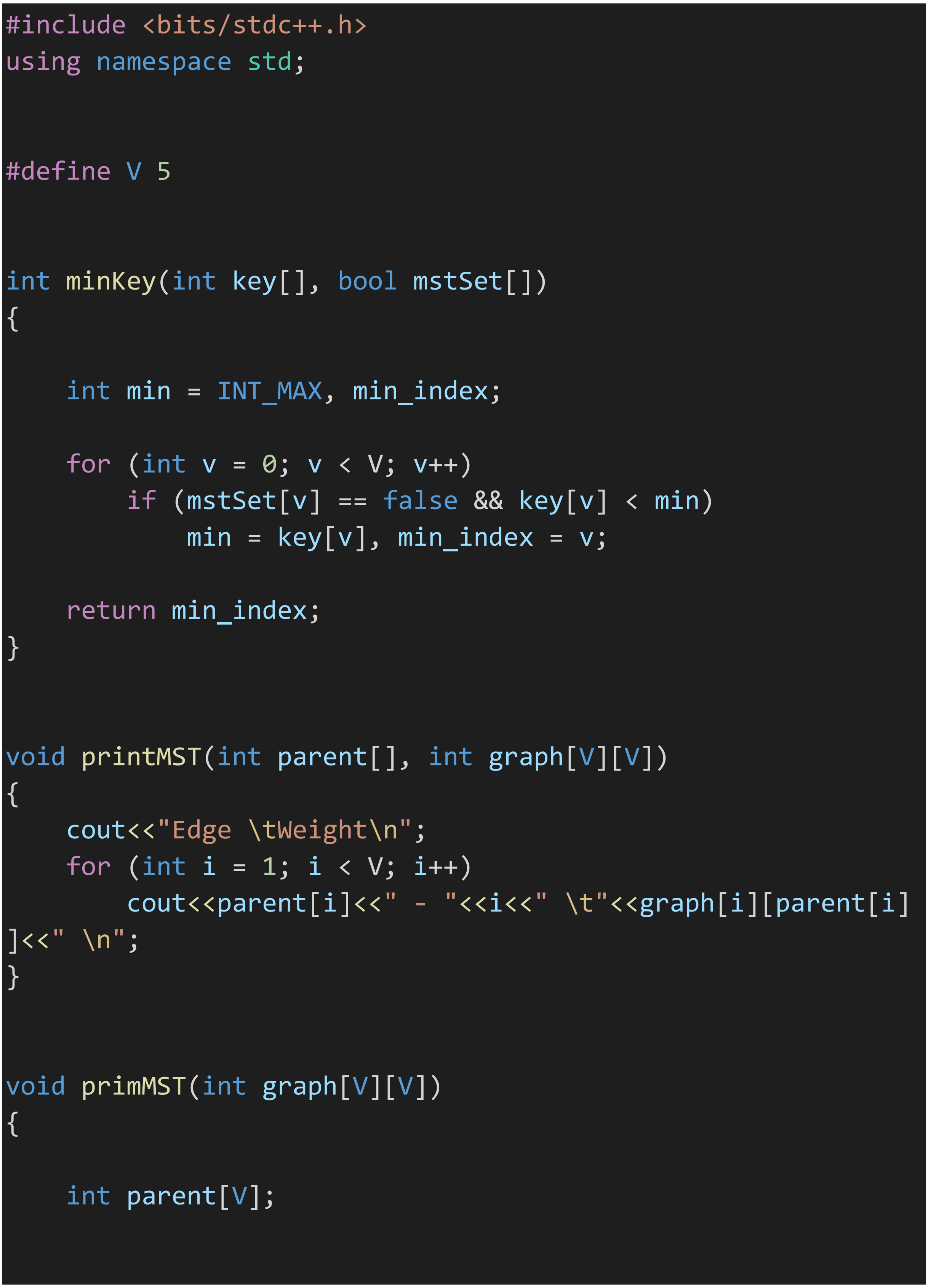
```
    return mst_wt;
}


int main()
{

    int V = 9, E = 14;
    Graph g(V, E);


    g.addEdge(0, 1, 4);
    g.addEdge(0, 7, 8);
    g.addEdge(1, 2, 8);
    g.addEdge(1, 7, 11);
    g.addEdge(2, 3, 7);
    g.addEdge(2, 8, 2);
    g.addEdge(2, 5, 4);
    g.addEdge(3, 4, 9);
    g.addEdge(3, 5, 14);
    g.addEdge(4, 5, 10);
    g.addEdge(5, 6, 2);
    g.addEdge(6, 7, 1);
    g.addEdge(6, 8, 6);
    g.addEdge(7, 8, 7);

    cout << "Edges of MST are \n";
    int mst_wt = g.kruskalMST();

    cout << "\nWeight of MST is " << mst_wt;

    return 0;
}
```

Output:

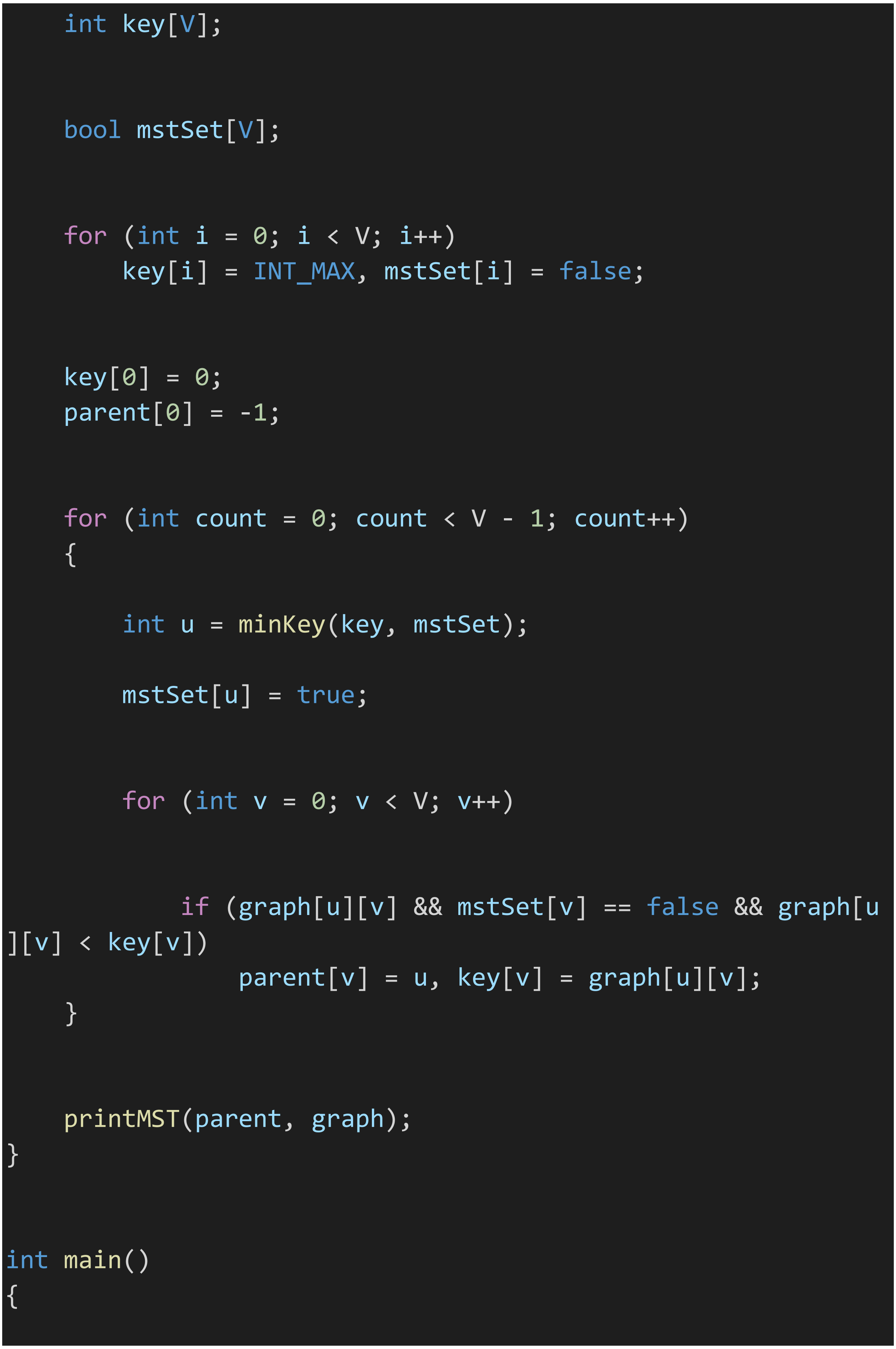


```
E:\PRADIP\DataStructureInC++>cd "e:\PRADIP\DataStructureInC++\" && g++ kruskal
.c++ -o kruskal && "e:\PRADIP\DataStructureInC++\"kruskal
Edges of MST are
6 - 7
2 - 8
5 - 6
0 - 1
2 - 5
2 - 3
0 - 7
3 - 4

Weight of MST is 37
E:\PRADIP\DataStructureInC++>
```

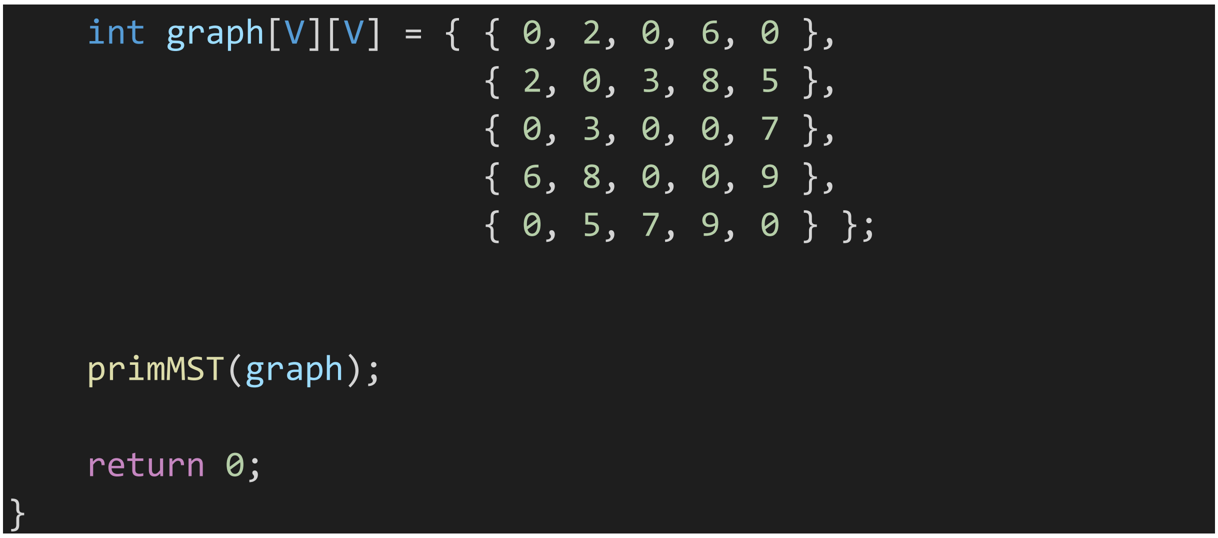
Prim's Algo



```
#include <bits/stdc++.h>
using namespace std;


#define V 5


int minKey(int key[], bool mstSet[])
{

    int min = INT_MAX, min_index;

    for (int v = 0; v < V; v++)
        if (mstSet[v] == false && key[v] < min)
            min = key[v], min_index = v;

    return min_index;
}


void printMST(int parent[], int graph[V][V])
{
    cout<<"Edge \tWeight\n";
    for (int i = 1; i < V; i++)
        cout<<parent[i]<<" - "<<i<<" \t"<<graph[i][parent[i]]<<" \n";
}


void primMST(int graph[V][V])
{

    int parent[V];

```

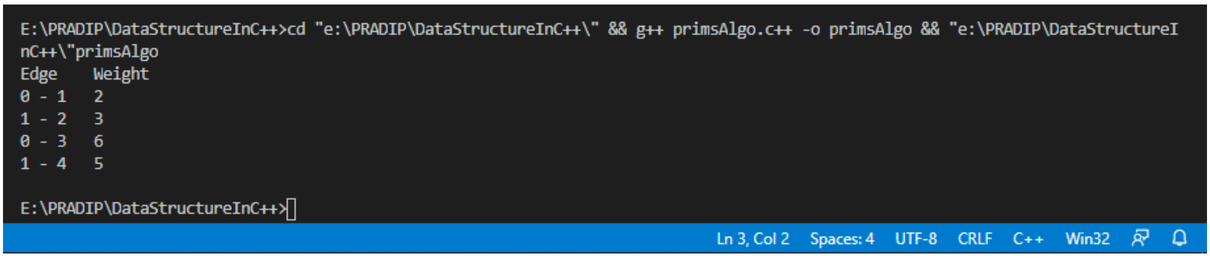


```
    int key[V];


    bool mstSet[V];


    for (int i = 0; i < V; i++)
        key[i] = INT_MAX, mstSet[i] = false;


    key[0] = 0;
    parent[0] = -1;


    for (int count = 0; count < V - 1; count++)
    {

        int u = minKey(key, mstSet);

        mstSet[u] = true;


        for (int v = 0; v < V; v++)


            if (graph[u][v] && mstSet[v] == false && graph[u][v] < key[v])
                parent[v] = u, key[v] = graph[u][v];
    }


    printMST(parent, graph);
}


int main()
{
```



```
    int graph[V][V] = { { 0, 2, 0, 6, 0 },
                        { 2, 0, 3, 8, 5 },
                        { 0, 3, 0, 0, 7 },
                        { 6, 8, 0, 0, 9 },
                        { 0, 5, 7, 9, 0 } };


    primMST(graph);

    return 0;
}
```

Output:



```
E:\PRADIP\DataStructureInC++>cd "e:\PRADIP\DataStructureInC++\" && g++ primsAlgo.c++ -o primsAlgo && "e:\PRADIP\DataStructureI
nC++\"primsAlgo
Edge    Weight
0 - 1   2
1 - 2   3
0 - 3   6
1 - 4   5

E:\PRADIP\DataStructureInC++>
```