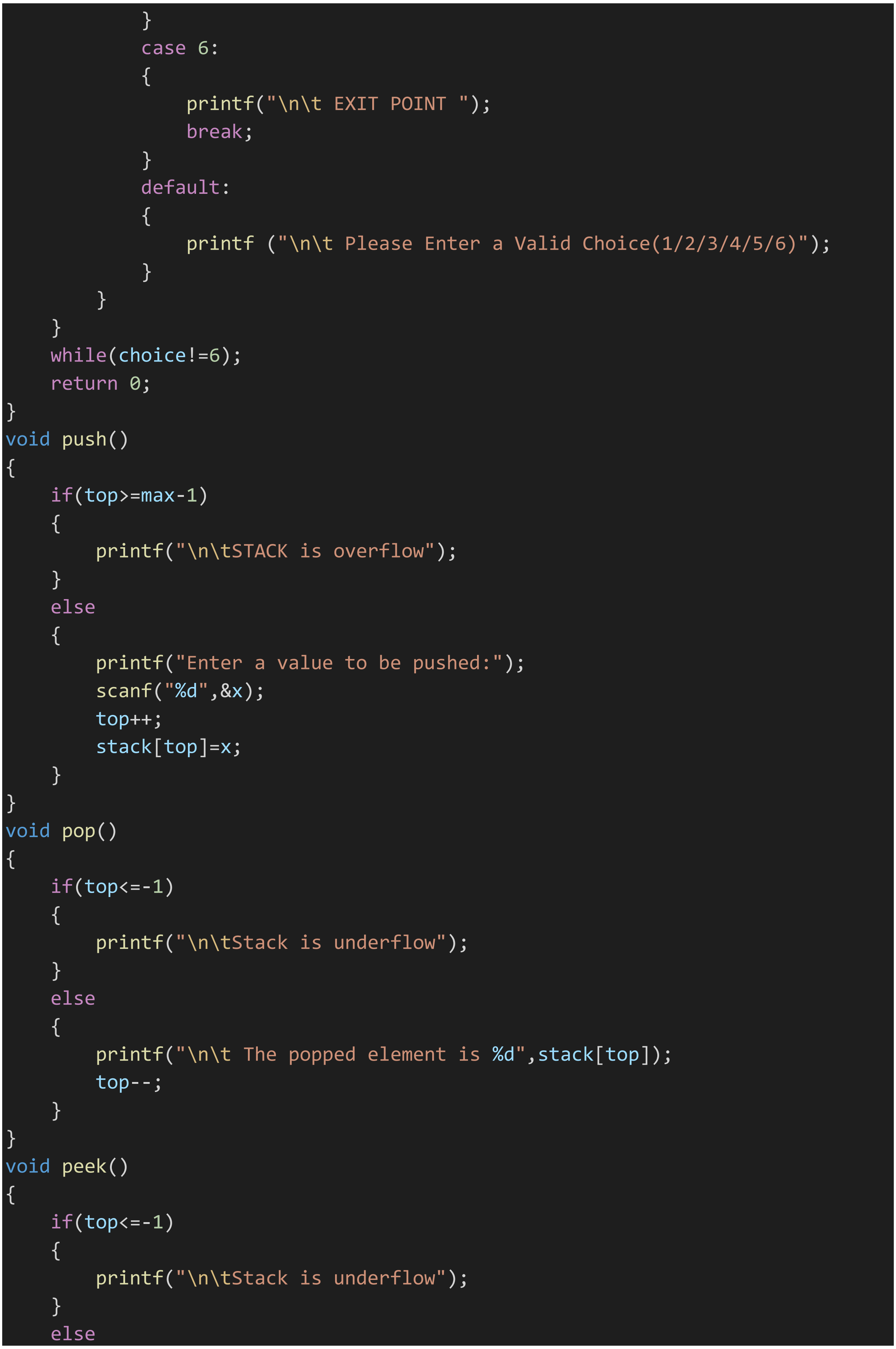
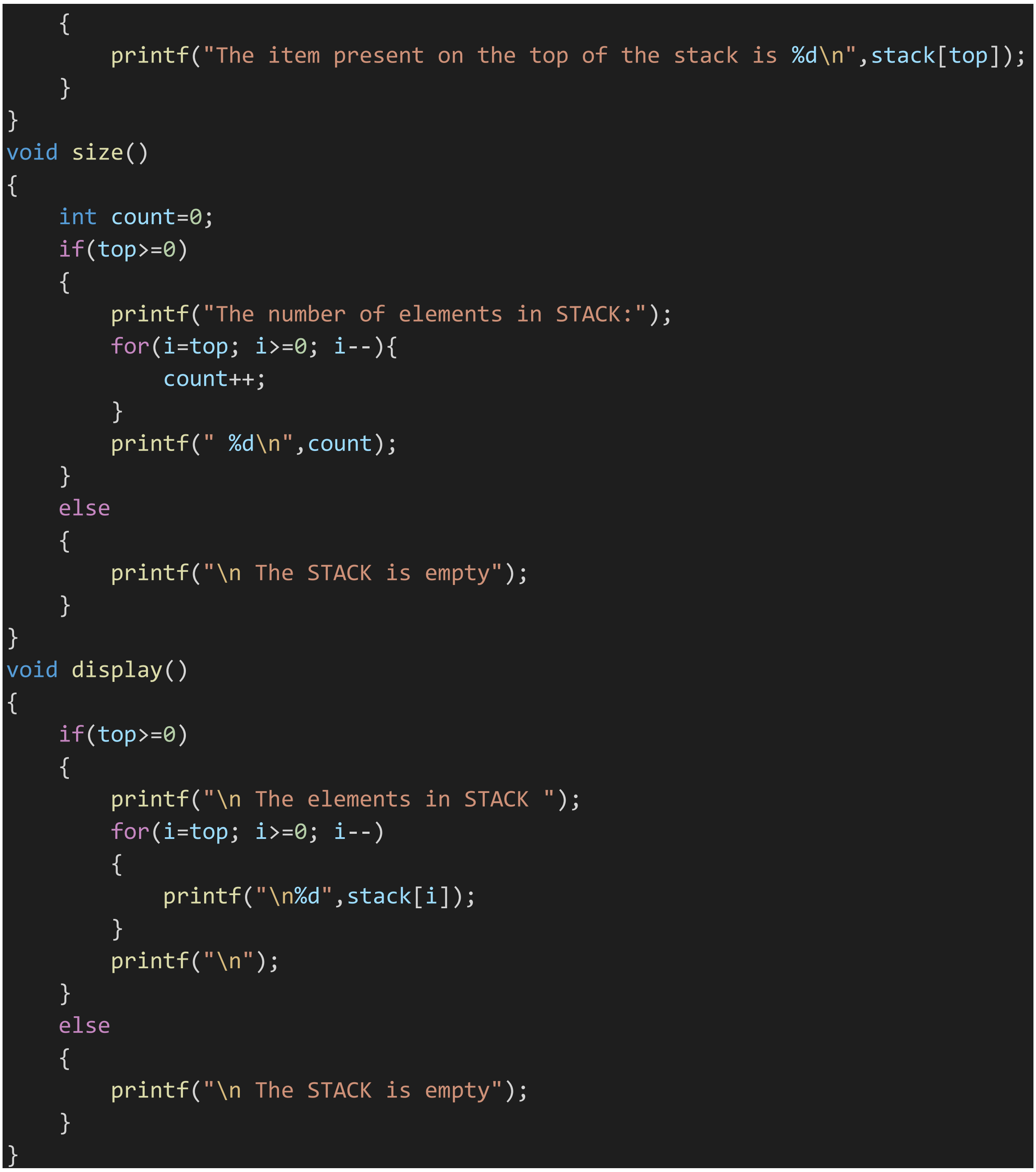
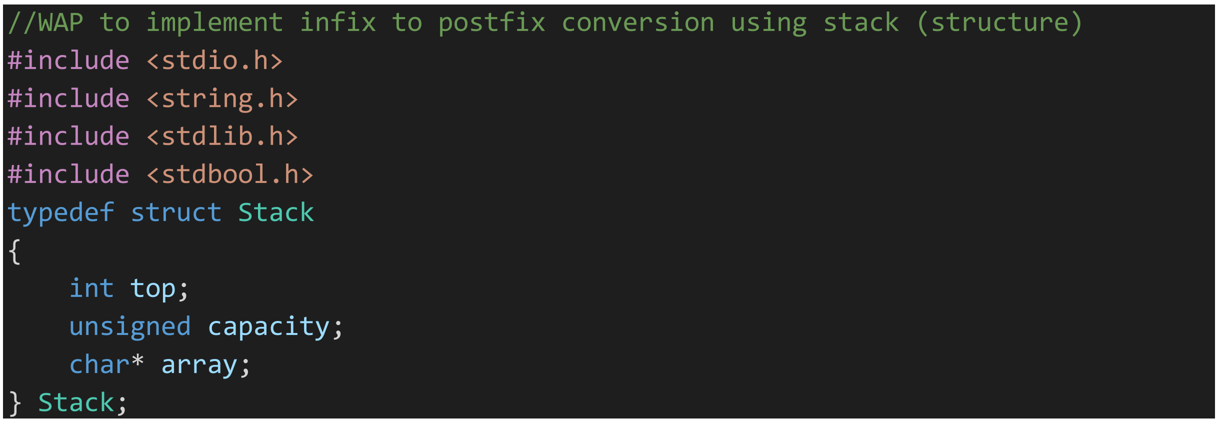
```
//Write a menu driven code to implement STACK ADT using arrays
#include <stdio.h>
int stack[10],choice,max,top,x,i;
void push(void);
void pop(void);
void peek(void);
void size(void);
void display(void);
int main()
{
    top=-1;
    printf("\n Enter the size of STACK[MAX=10]:");
    scanf("%d",&max);
    printf("\n\t STACK OPERATIONS USING ARRAY");
    printf("\n\t--------------------------------");
    printf("\n\t 1.PUSH\n\t 2.POP\n\t 3.PEEK\n\t 4.SIZE\n\t 5.DISPLAY\n\t
6.EXIT");
    printf("\n\t--------------------------------");
    do
    {
        printf("\n Enter the Choice:");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1:
            {
                push();
                break;
            }
            case 2:
            {
                pop();
                break;
            }
            case 3:
            {
                peek();
                break;
            }
            case 4:
            {
                size();
                break;
            }
            case 5:
            {
                display();
                break;
```



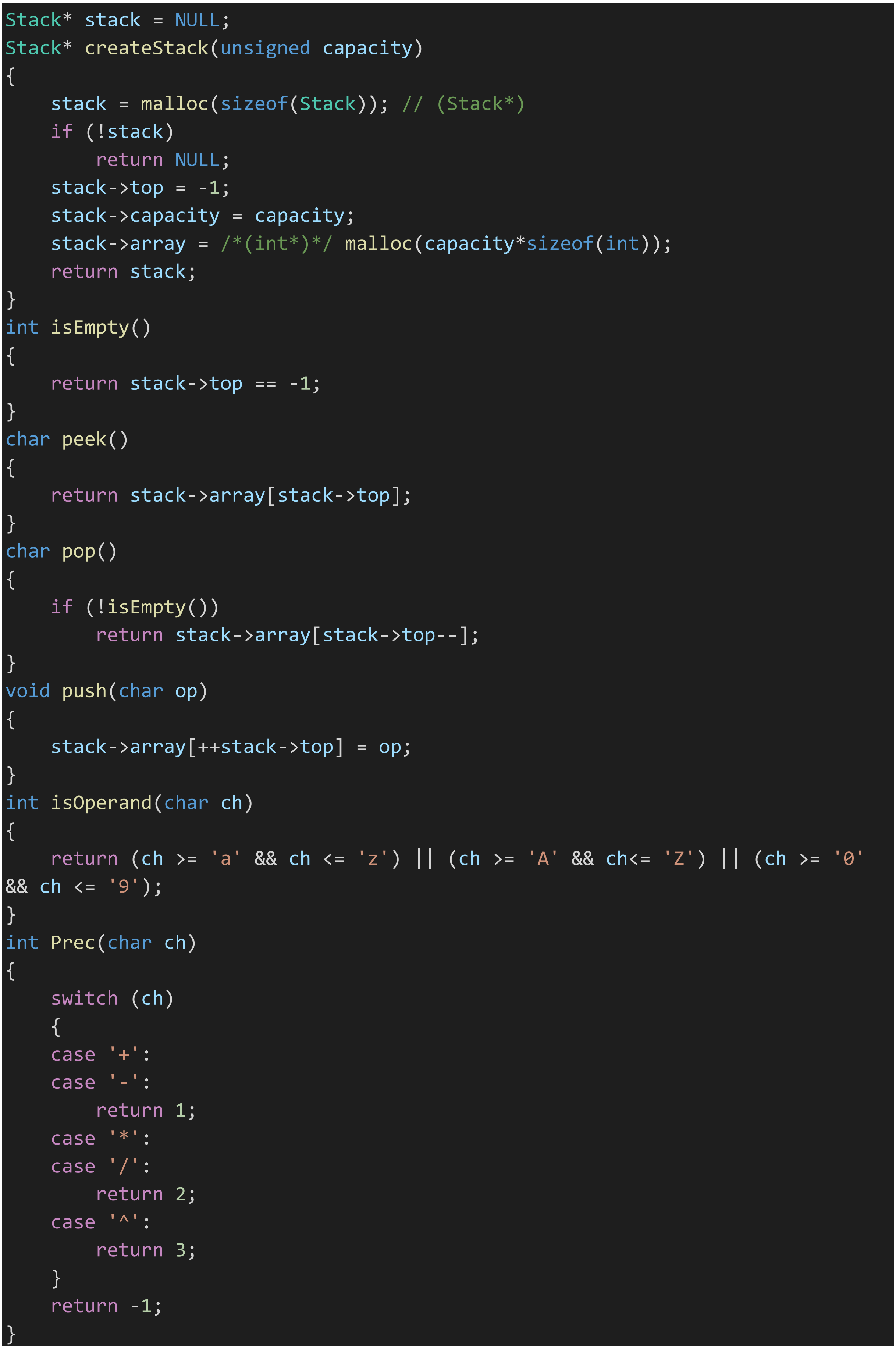
```
            }
            case 6:
            {
                printf("\n\t EXIT POINT ");
                break;
            }
            default:
            {
                printf ("\n\t Please Enter a Valid Choice(1/2/3/4/5/6)");
            }
        }
    }
    while(choice!=6);
    return 0;
}
void push()
{
    if(top>=max-1)
    {
        printf("\n\tSTACK is overflow");
    }
    else
    {
        printf("Enter a value to be pushed:");
        scanf("%d",&x);
        top++;
        stack[top]=x;
    }
}
void pop()
{
    if(top<=-1)
    {
        printf("\n\tStack is underflow");
    }
    else
    {
        printf("\n\t The popped element is %d",stack[top]);
        top--;
    }
}
void peek()
{
    if(top<=-1)
    {
        printf("\n\tStack is underflow");
    }
    else
```



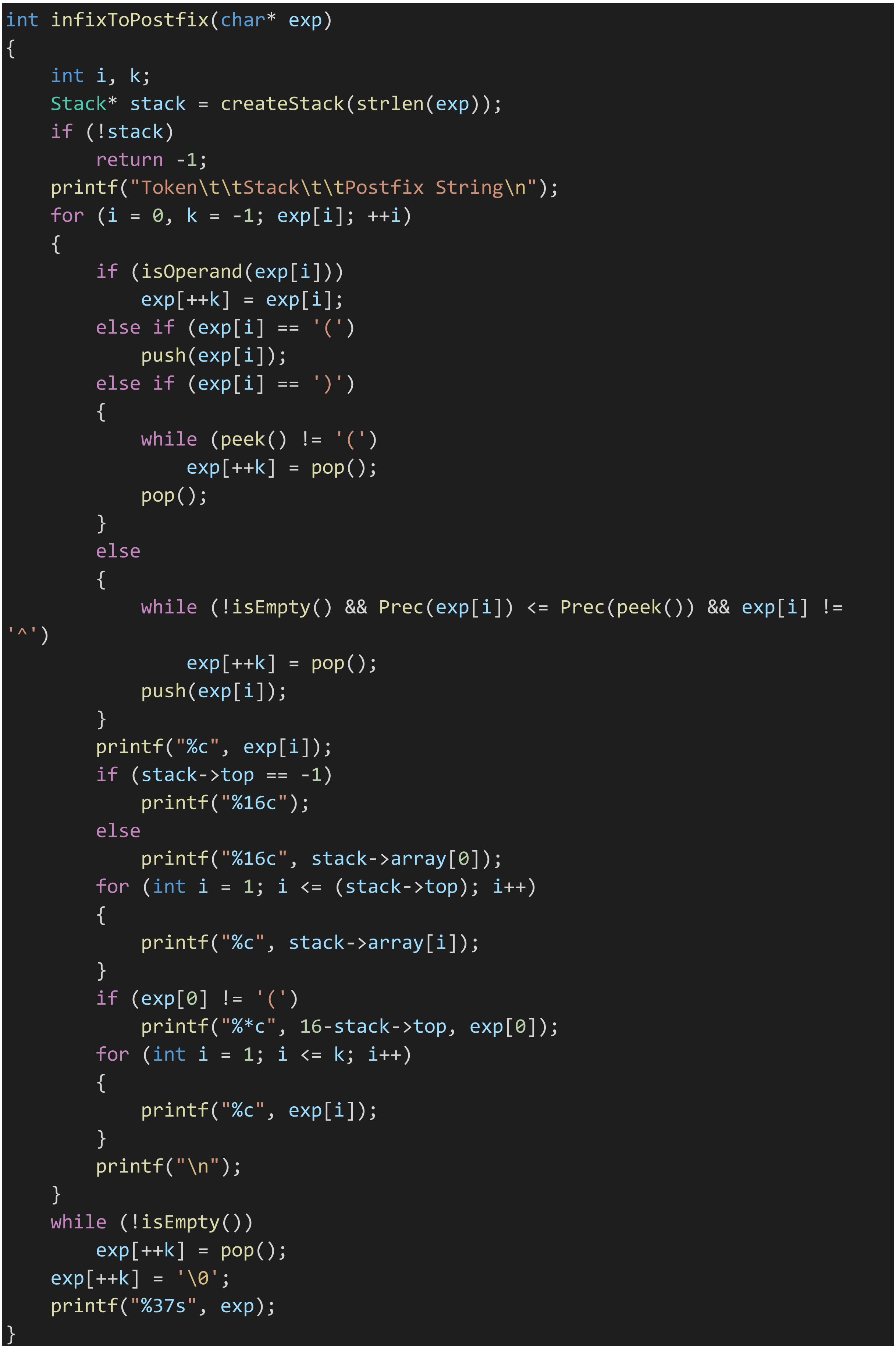
```
    {
        printf("The item present on the top of the stack is %d\n",stack[top]);
    }
}
void size()
{
    int count=0;
    if(top>=0)
    {
        printf("The number of elements in STACK:");
        for(i=top; i>=0; i--){
            count++;
        }
        printf(" %d\n",count);
    }
    else
    {
        printf("\n The STACK is empty");
    }
}
void display()
{
    if(top>=0)
    {
        printf("\n The elements in STACK ");
        for(i=top; i>=0; i--)
        {
            printf("\n%d",stack[i]);
        }
        printf("\n");
    }
    else
    {
        printf("\n The STACK is empty");
    }
}
```



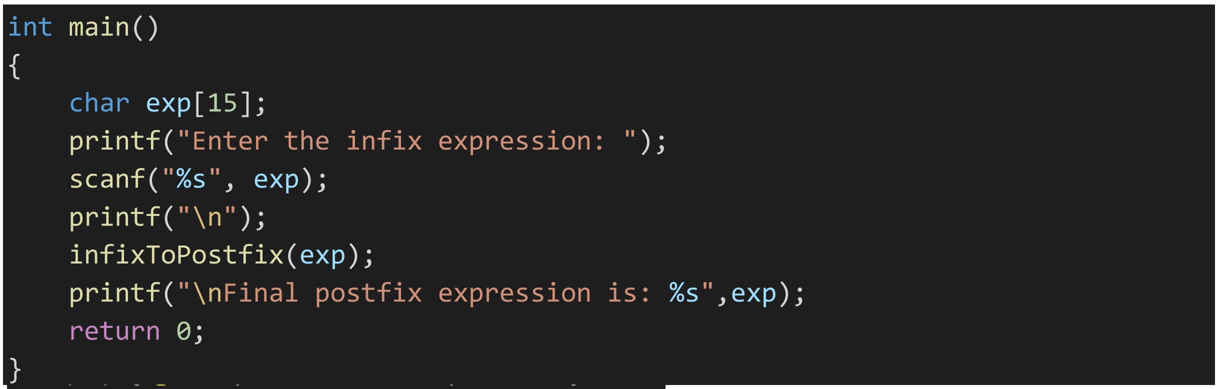
```
//WAP to implement infix to postfix conversion using stack (structure)
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <stdbool.h>
typedef struct Stack
{
    int top;
    unsigned capacity;
    char* array;
} Stack;
```



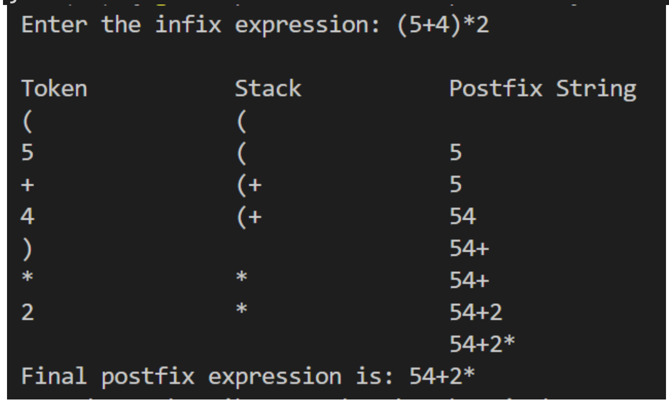
```
Stack* stack = NULL;
Stack* createStack(unsigned capacity)
{
    stack = malloc(sizeof(Stack)); // (Stack*)
    if (!stack)
        return NULL;
    stack->top = -1;
    stack->capacity = capacity;
    stack->array = /*(int*)*/ malloc(capacity*sizeof(int));
    return stack;
}
int isEmpty()
{
    return stack->top == -1;
}
char peek()
{
    return stack->array[stack->top];
}
char pop()
{
    if (!isEmpty())
        return stack->array[stack->top--];
}
void push(char op)
{
    stack->array[++stack->top] = op;
}
int isOperand(char ch)
{
    return (ch >= 'a' && ch <= 'z') || (ch >= 'A' && ch<= 'Z') || (ch >= '0'
&& ch <= '9');
}
int Prec(char ch)
{
    switch (ch)
    {
    case '+':
    case '-':
        return 1;
    case '*':
    case '/':
        return 2;
    case '^':
        return 3;
    }
    return -1;
}
```



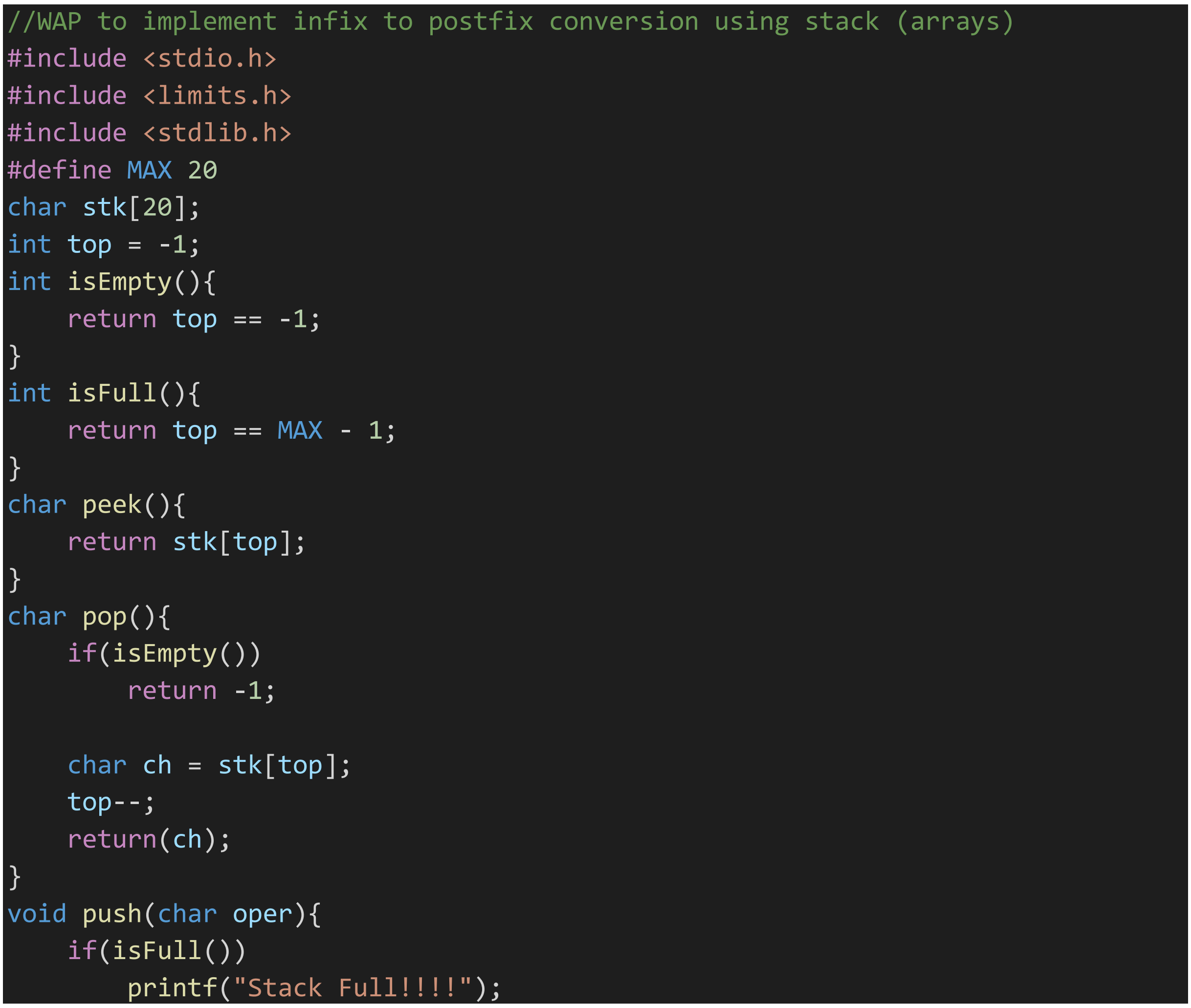
```c
int infixToPostfix(char* exp)
{
    int i, k;
    Stack* stack = createStack(strlen(exp));
    if (!stack)
        return -1;
    printf("Token\t\tStack\t\tPostfix String\n");
    for (i = 0, k = -1; exp[i]; ++i)
    {
        if (isOperand(exp[i]))
            exp[++k] = exp[i];
        else if (exp[i] == '(')
            push(exp[i]);
        else if (exp[i] == ')')
        {
            while (peek() != '(')
                exp[++k] = pop();
            pop();
        }
        else
        {
            while (!isEmpty() && Prec(exp[i]) <= Prec(peek()) && exp[i] !=
'^')
                exp[++k] = pop();
            push(exp[i]);
        }
        printf("%c", exp[i]);
        if (stack->top == -1)
            printf("%16c");
        else
            printf("%16c", stack->array[0]);
        for (int i = 1; i <= (stack->top); i++)
        {
            printf("%c", stack->array[i]);
        }
        if (exp[0] != '(')
            printf("%*c", 16-stack->top, exp[0]);
        for (int i = 1; i <= k; i++)
        {
            printf("%c", exp[i]);
        }
        printf("\n");
    }
    while (!isEmpty())
        exp[++k] = pop();
    exp[++k] = '\0';
    printf("%37s", exp);
}
```



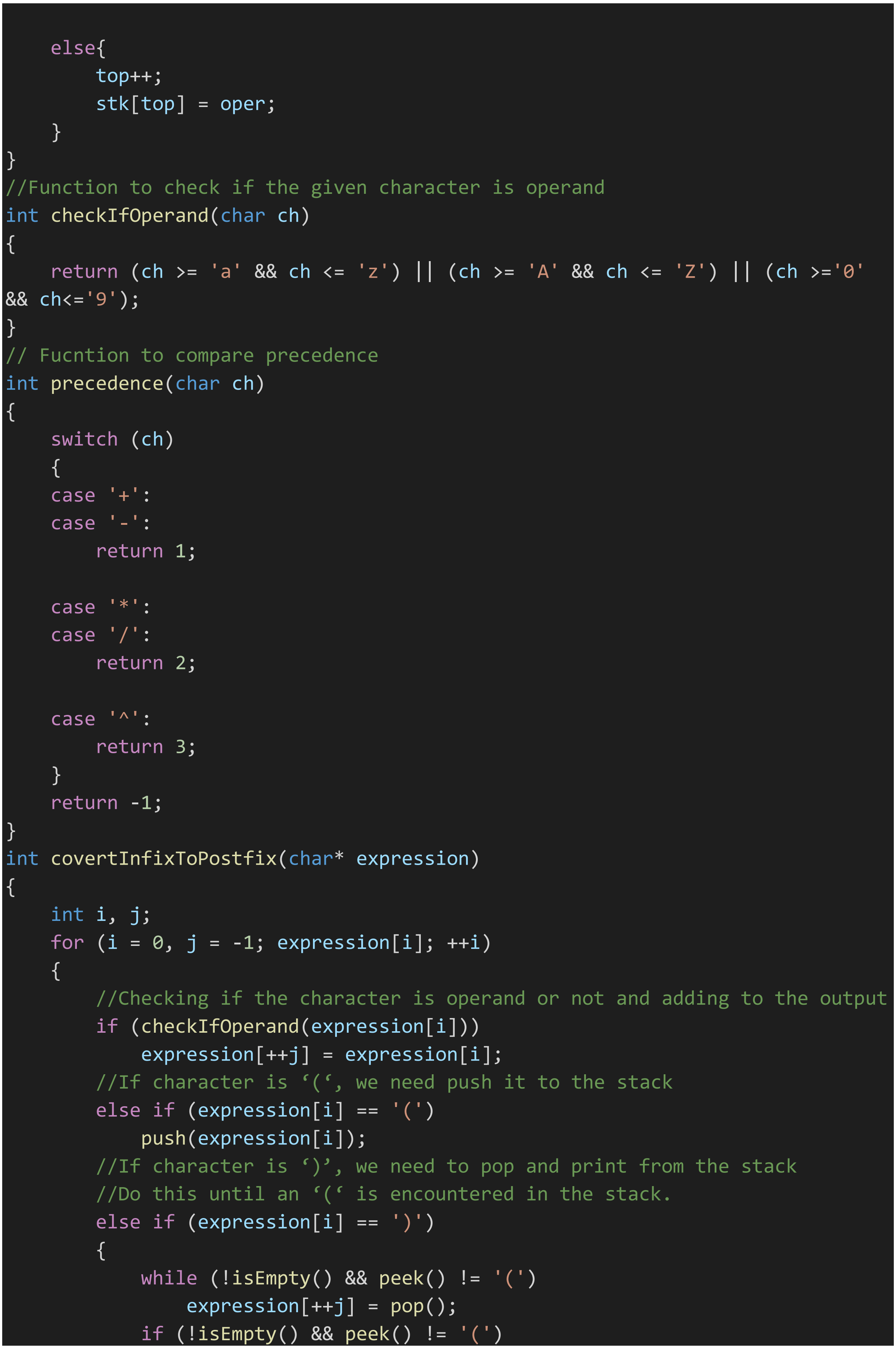
```c
int main()
{
    char exp[15];
    printf("Enter the infix expression: ");
    scanf("%s", exp);
    printf("\n");
    infixToPostfix(exp);
    printf("\nFinal postfix expression is: %s",exp);
    return 0;
}
```



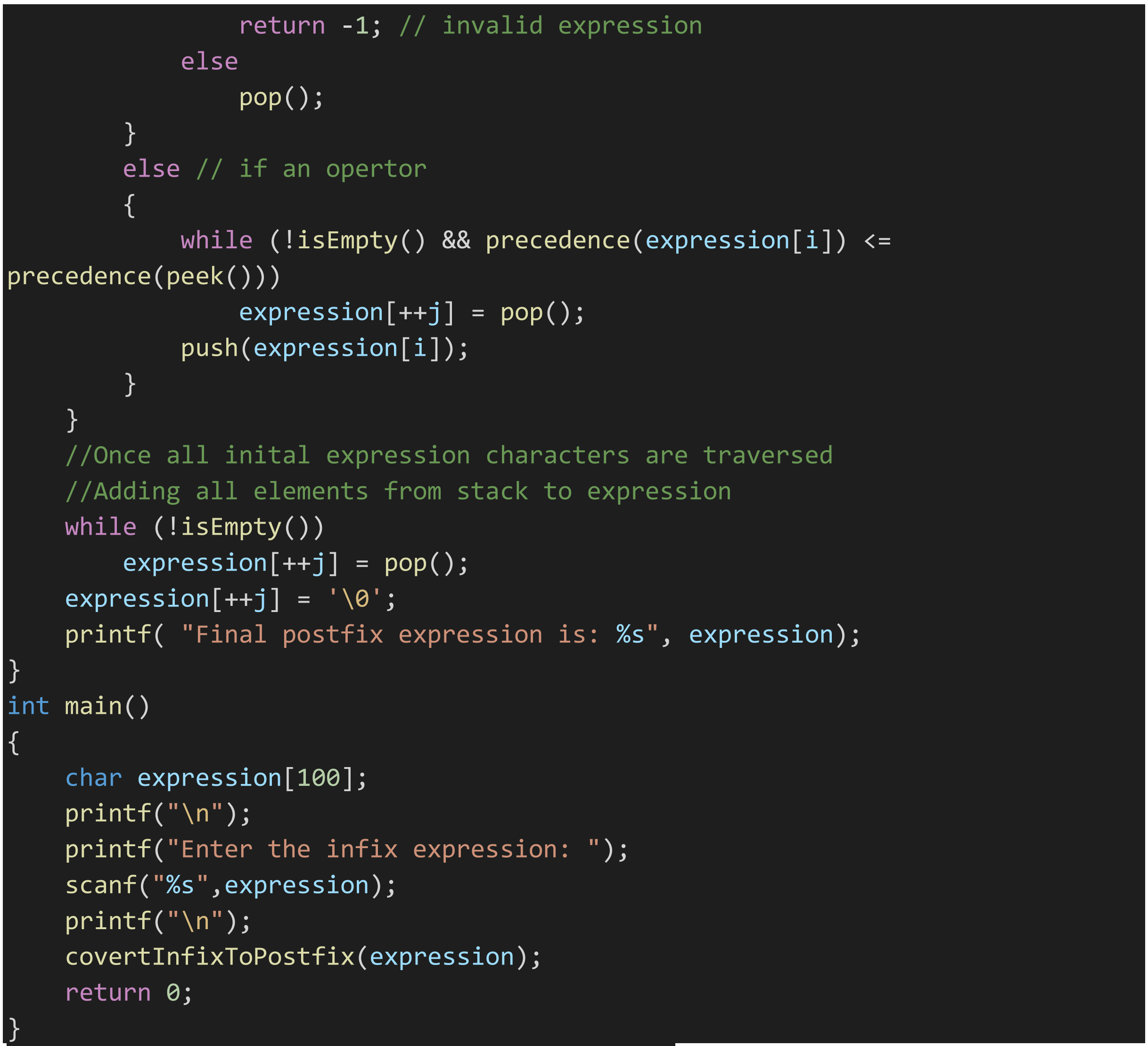
```
Enter the infix expression: (5+4)*2

Token           Stack           Postfix String
(               (
5               (               5
+               (+              5
4               (+              54
)                               54+
*               *               54+
2               *               54+2
                                54+2*
Final postfix expression is: 54+2*
```



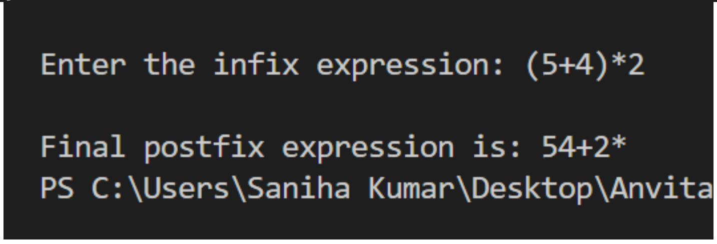
```c
//WAP to implement infix to postfix conversion using stack (arrays)
#include <stdio.h>
#include <limits.h>
#include <stdlib.h>
#define MAX 20
char stk[20];
int top = -1;
int isEmpty(){
    return top == -1;
}
int isFull(){
    return top == MAX - 1;
}
char peek(){
    return stk[top];
}
char pop(){
    if(isEmpty())
        return -1;

    char ch = stk[top];
    top--;
    return(ch);
}
void push(char oper){
    if(isFull())
        printf("Stack Full!!!!");
```



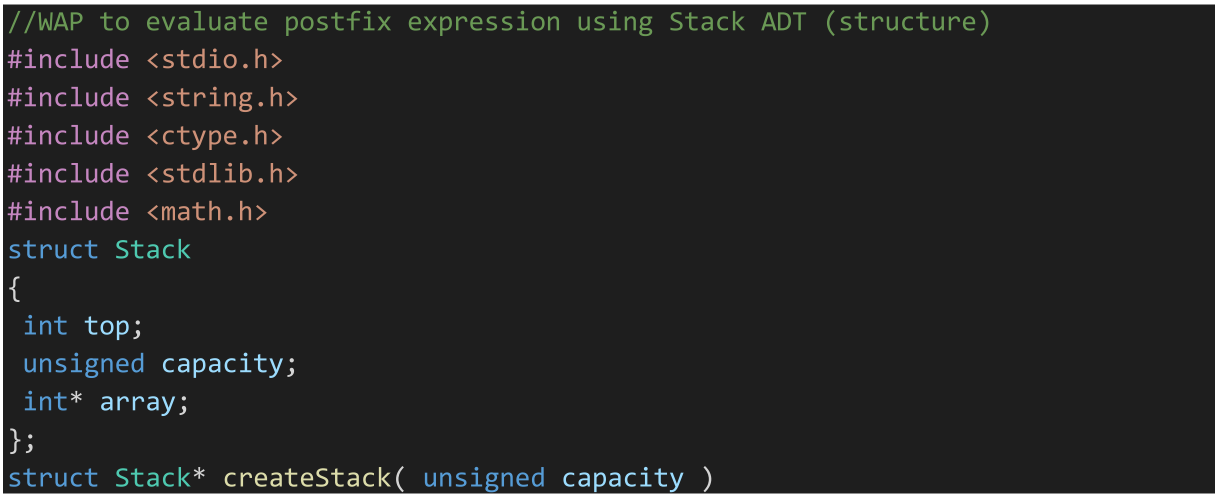
```
    else{
        top++;
        stk[top] = oper;
    }
}
//Function to check if the given character is operand
int checkIfOperand(char ch)
{
    return (ch >= 'a' && ch <= 'z') || (ch >= 'A' && ch <= 'Z') || (ch >='0'
&& ch<='9');
}
// Fucntion to compare precedence
int precedence(char ch)
{
    switch (ch)
    {
    case '+':
    case '-':
        return 1;

    case '*':
    case '/':
        return 2;

    case '^':
        return 3;
    }
    return -1;
}
int covertInfixToPostfix(char* expression)
{
    int i, j;
    for (i = 0, j = -1; expression[i]; ++i)
    {
        //Checking if the character is operand or not and adding to the output
        if (checkIfOperand(expression[i]))
            expression[++j] = expression[i];
        //If character is ‘(‘, we need push it to the stack
        else if (expression[i] == '(')
            push(expression[i]);
        //If character is ‘)’, we need to pop and print from the stack
        //Do this until an ‘(‘ is encountered in the stack.
        else if (expression[i] == ')')
        {
            while (!isEmpty() && peek() != '(')
                expression[++j] = pop();
            if (!isEmpty() && peek() != '(')
```



```
                return -1; // invalid expression
            else
                pop();
        }
        else // if an opertor
        {
            while (!isEmpty() && precedence(expression[i]) <=
precedence(peek()))
                expression[++j] = pop();
            push(expression[i]);
        }
    }
    //Once all inital expression characters are traversed
    //Adding all elements from stack to expression
    while (!isEmpty())
        expression[++j] = pop();
    expression[++j] = '\0';
    printf( "Final postfix expression is: %s", expression);
}
int main()
{
    char expression[100];
    printf("\n");
    printf("Enter the infix expression: ");
    scanf("%s",expression);
    printf("\n");
    covertInfixToPostfix(expression);
    return 0;
}
```

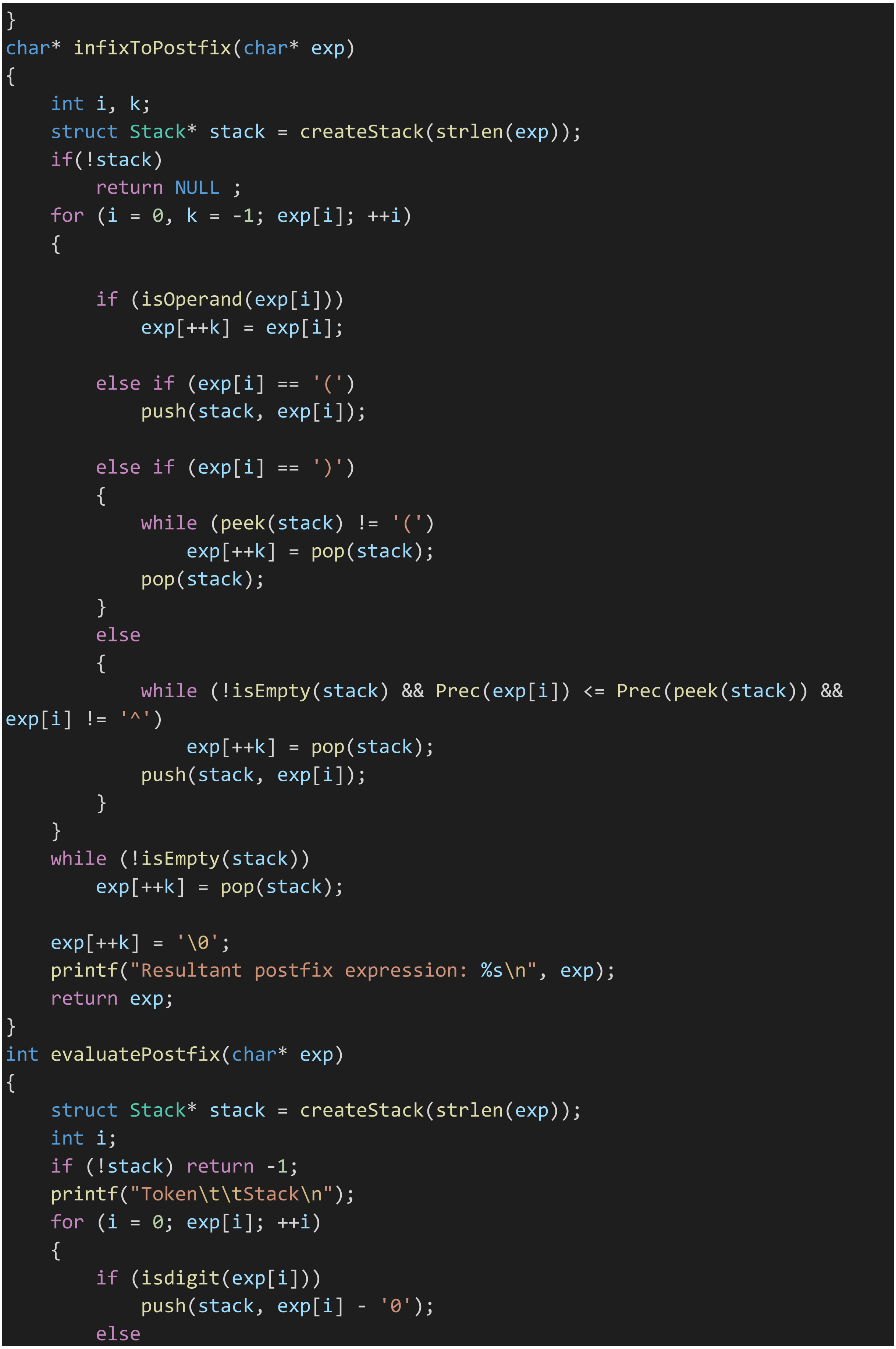


```
Enter the infix expression: (5+4)*2

Final postfix expression is: 54+2*
PS C:\Users\Saniha Kumar\Desktop\Anvita
```



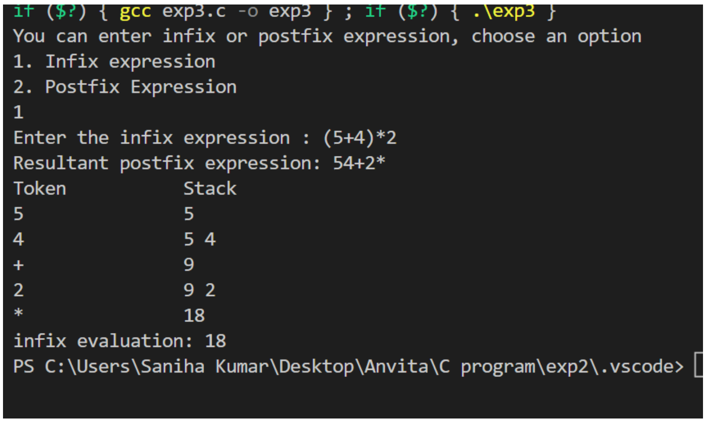
```
//WAP to evaluate postfix expression using Stack ADT (structure)
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <stdlib.h>
#include <math.h>
struct Stack
{
 int top;
 unsigned capacity;
 int* array;
};
struct Stack* createStack( unsigned capacity )
```

```
{
 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 isOperand(char ch)
{
    return (ch >= 'a' && ch <= 'z') || (ch >= 'A' && ch <= 'Z') || (ch >= '0'
&& ch <= '9');
}
int Prec(char ch)
{
    switch (ch)
    {
    case '+':
    case '-':
        return 1;

    case '*':
    case '/':
        return 2;

    case '^':
        return 3;
    }
    return -1;
```

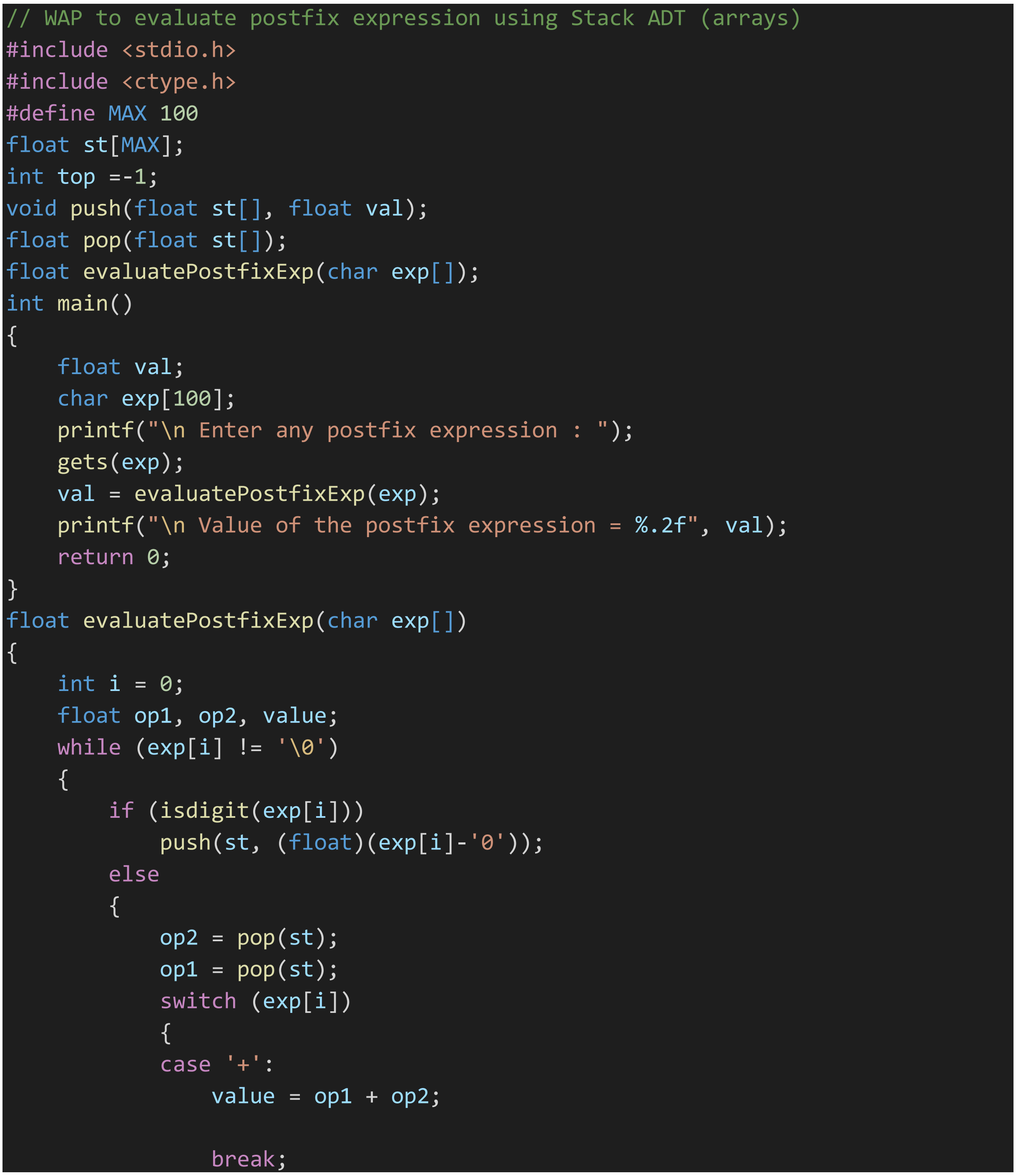


```
}
char* infixToPostfix(char* exp)
{
    int i, k;
    struct Stack* stack = createStack(strlen(exp));
    if(!stack)
        return NULL ;
    for (i = 0, k = -1; exp[i]; ++i)
    {

        if (isOperand(exp[i]))
            exp[++k] = exp[i];

        else if (exp[i] == '(')
            push(stack, exp[i]);

        else if (exp[i] == ')')
        {
            while (peek(stack) != '(')
                exp[++k] = pop(stack);
            pop(stack);
        }
        else
        {
            while (!isEmpty(stack) && Prec(exp[i]) <= Prec(peek(stack)) &&
exp[i] != '^')
                exp[++k] = pop(stack);
            push(stack, exp[i]);
        }
    }
    while (!isEmpty(stack))
        exp[++k] = pop(stack);

    exp[++k] = '\0';
    printf("Resultant postfix expression: %s\n", exp);
    return exp;
}
int evaluatePostfix(char* exp)
{
    struct Stack* stack = createStack(strlen(exp));
    int i;
    if (!stack) return -1;
    printf("Token\t\tStack\n");
    for (i = 0; exp[i]; ++i)
    {
        if (isdigit(exp[i]))
            push(stack, exp[i] - '0');
        else
```

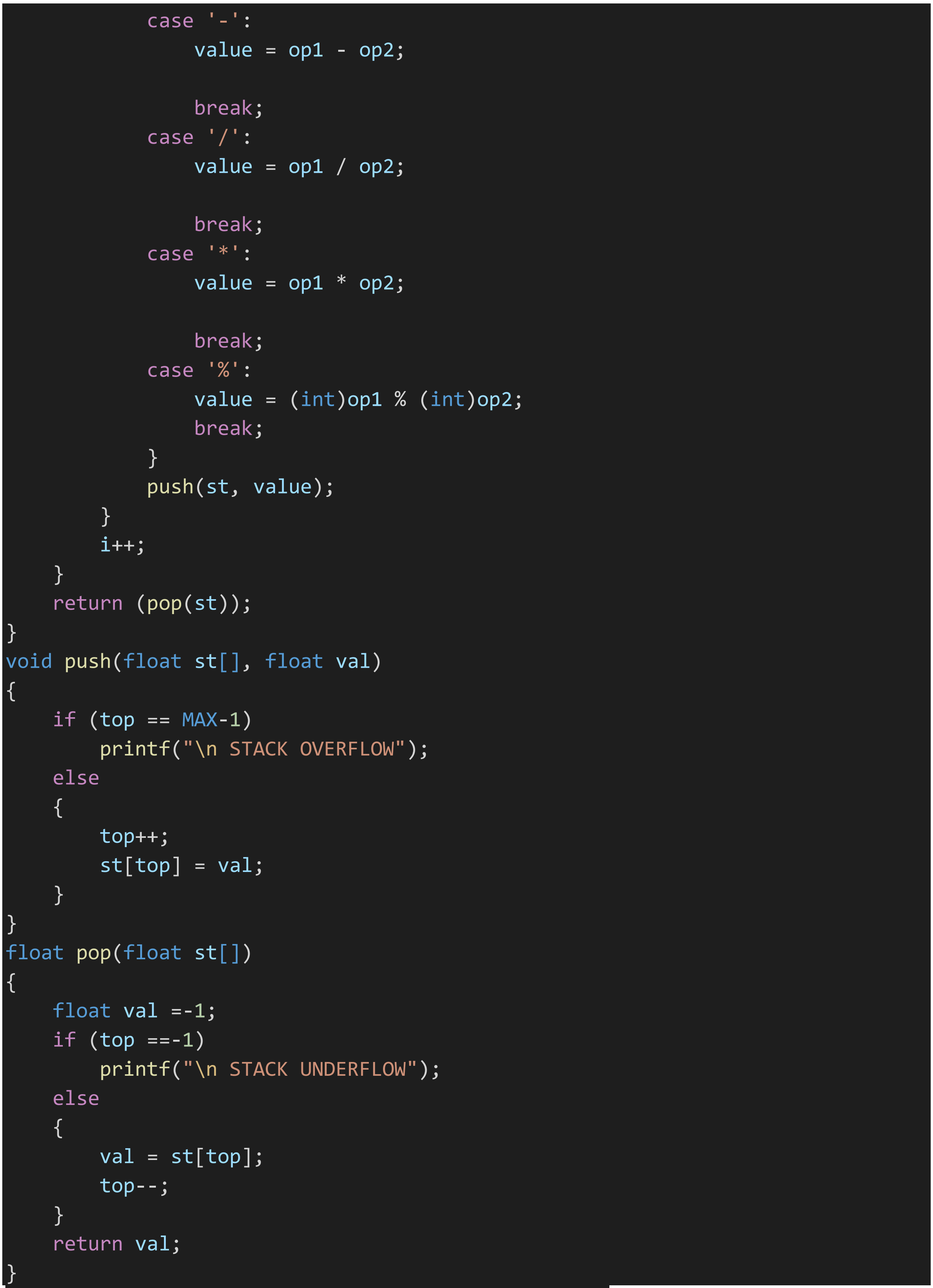
```c
        {
            int val1 = pop(stack);
            int val2 = pop(stack);
            switch (exp[i])
            {
            case '+': push(stack, val2 + val1); break;
            case '-': push(stack, val2 - val1); break;
            case '*': push(stack, val2 * val1); break;
            case '/': push(stack, val2/val1); break;
            case '^': push(stack, pow(val2, val1)); break;
            }
        }
        printf("%-16c", exp[i]);
        for (int i = 0; i <= stack->top; i++)
        {
            printf("%d ", stack->array[i]);
        }
        printf("\n");
    }
    return pop(stack);
}
int main()
{
    int c;
    here:
    printf("You can enter infix or postfix expression, choose an option\n1.
Infix expression\n2. Postfix Expression\n");
    scanf("%d", &c);
    char exp[20];
    switch(c) {
        case 1:
            printf("Enter the infix expression : ");
            scanf("%s", exp);
            printf ("infix evaluation: %d",
evaluatePostfix(infixToPostfix(exp)));
            break;
        case 2:
            printf("Enter the postfix expression : ");
            scanf("%s", exp);
            printf ("postfix evaluation: %d", evaluatePostfix(exp));
            break;
        default:
            goto here;
    }
 return 0;
}
```



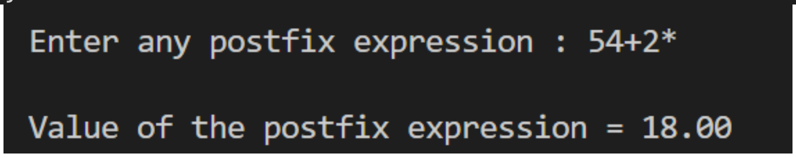
```
if ($?) { gcc exp3.c -o exp3 } ; if ($?) { .\exp3 }
You can enter infix or postfix expression, choose an option
1. Infix expression
2. Postfix Expression
1
Enter the infix expression : (5+4)*2
Resultant postfix expression: 54+2*
Token           Stack
5               5
4               5 4
+               9
2               9 2
*               18
infix evaluation: 18
PS C:\Users\Saniha Kumar\Desktop\Anvita\C program\exp2\.vscode> 
```



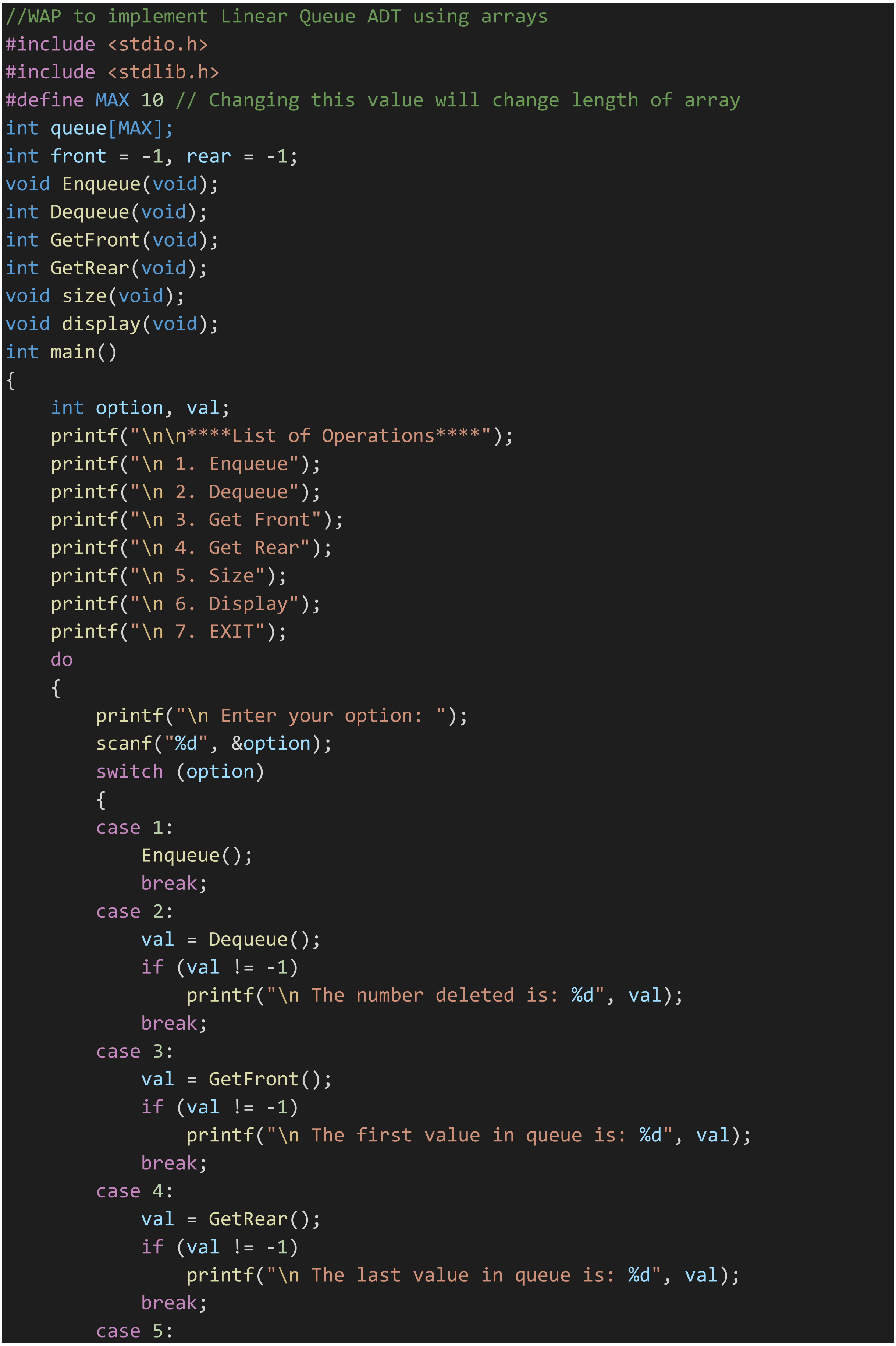
```c
// WAP to evaluate postfix expression using Stack ADT (arrays)
#include <stdio.h>
#include <ctype.h>
#define MAX 100
float st[MAX];
int top =-1;
void push(float st[], float val);
float pop(float st[]);
float evaluatePostfixExp(char exp[]);
int main()
{
    float val;
    char exp[100];
    printf("\n Enter any postfix expression : ");
    gets(exp);
    val = evaluatePostfixExp(exp);
    printf("\n Value of the postfix expression = %.2f", val);
    return 0;
}
float evaluatePostfixExp(char exp[])
{
    int i = 0;
    float op1, op2, value;
    while (exp[i] != '\0')
    {
        if (isdigit(exp[i]))
            push(st, (float)(exp[i]-'0'));
        else
        {
            op2 = pop(st);
            op1 = pop(st);
            switch (exp[i])
            {
            case '+':
                value = op1 + op2;

                break;
```



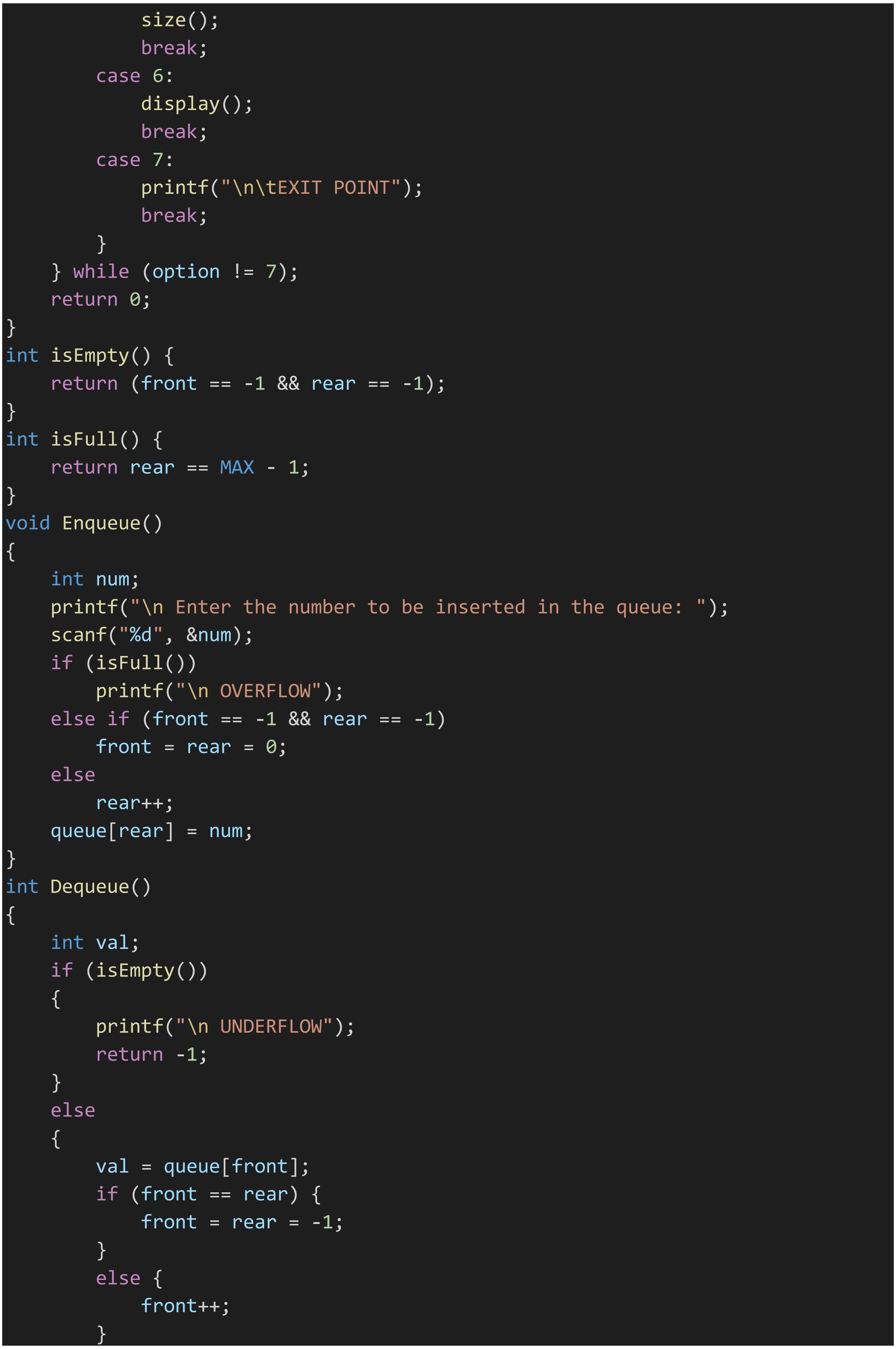
```
            case '-':
                value = op1 - op2;

                break;
            case '/':
                value = op1 / op2;

                break;
            case '*':
                value = op1 * op2;

                break;
            case '%':
                value = (int)op1 % (int)op2;
                break;
            }
            push(st, value);
        }
        i++;
    }
    return (pop(st));
}
void push(float st[], float val)
{
    if (top == MAX-1)
        printf("\n STACK OVERFLOW");
    else
    {
        top++;
        st[top] = val;
    }
}
float pop(float st[])
{
    float val =-1;
    if (top ==-1)
        printf("\n STACK UNDERFLOW");
    else
    {
        val = st[top];
        top--;
    }
    return val;
}
```



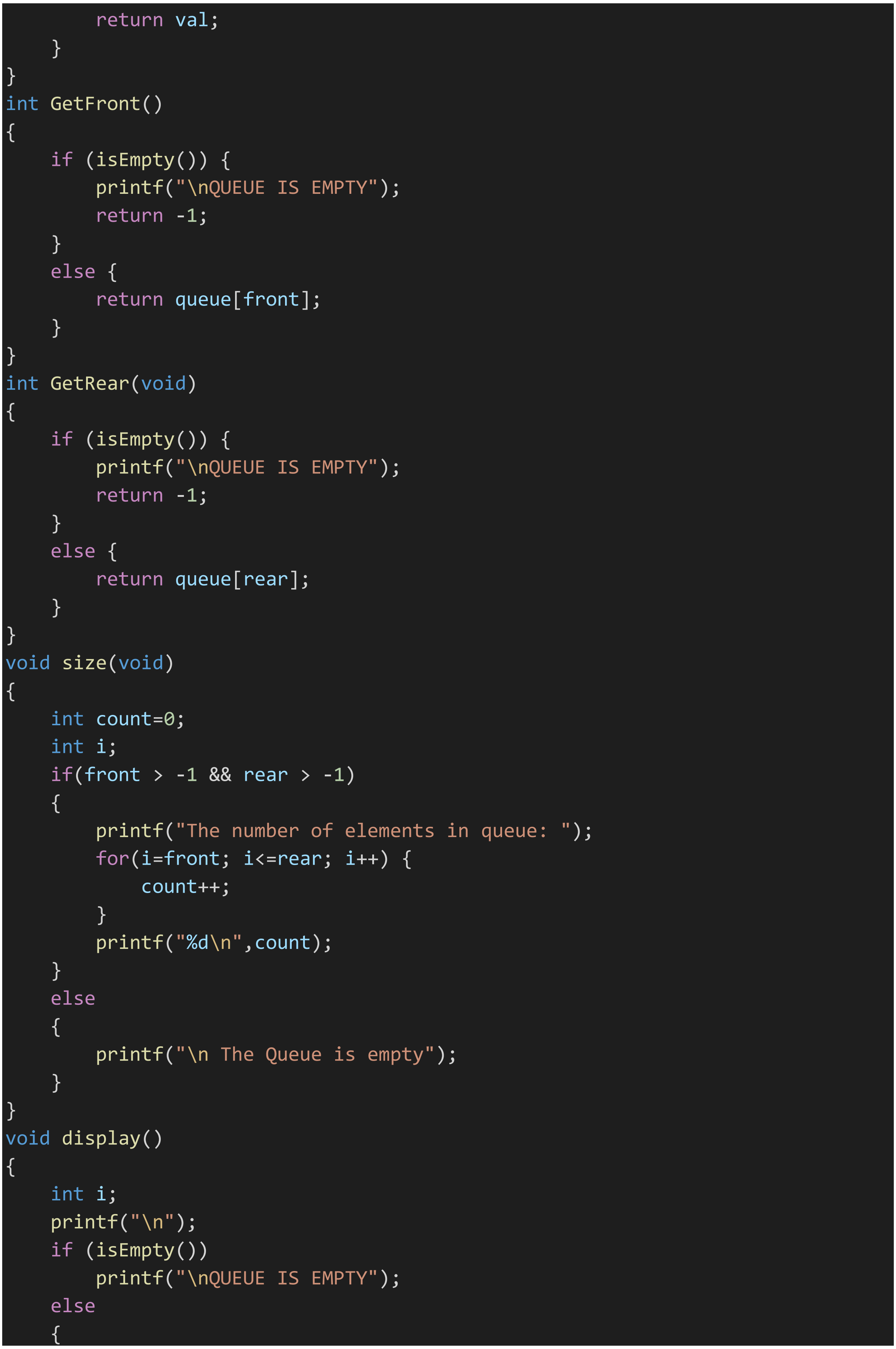
```
Enter any postfix expression : 54+2*

Value of the postfix expression = 18.00
```



```
//WAP to implement Linear Queue ADT using arrays
#include <stdio.h>
#include <stdlib.h>
#define MAX 10 // Changing this value will change length of array
int queue[MAX];
int front = -1, rear = -1;
void Enqueue(void);
int Dequeue(void);
int GetFront(void);
int GetRear(void);
void size(void);
void display(void);
int main()
{
    int option, val;
    printf("\n\n****List of Operations****");
    printf("\n 1. Enqueue");
    printf("\n 2. Dequeue");
    printf("\n 3. Get Front");
    printf("\n 4. Get Rear");
    printf("\n 5. Size");
    printf("\n 6. Display");
    printf("\n 7. EXIT");
    do
    {
        printf("\n Enter your option: ");
        scanf("%d", &option);
        switch (option)
        {
        case 1:
            Enqueue();
            break;
        case 2:
            val = Dequeue();
            if (val != -1)
                printf("\n The number deleted is: %d", val);
            break;
        case 3:
            val = GetFront();
            if (val != -1)
                printf("\n The first value in queue is: %d", val);
            break;
        case 4:
            val = GetRear();
            if (val != -1)
                printf("\n The last value in queue is: %d", val);
            break;
        case 5:
```

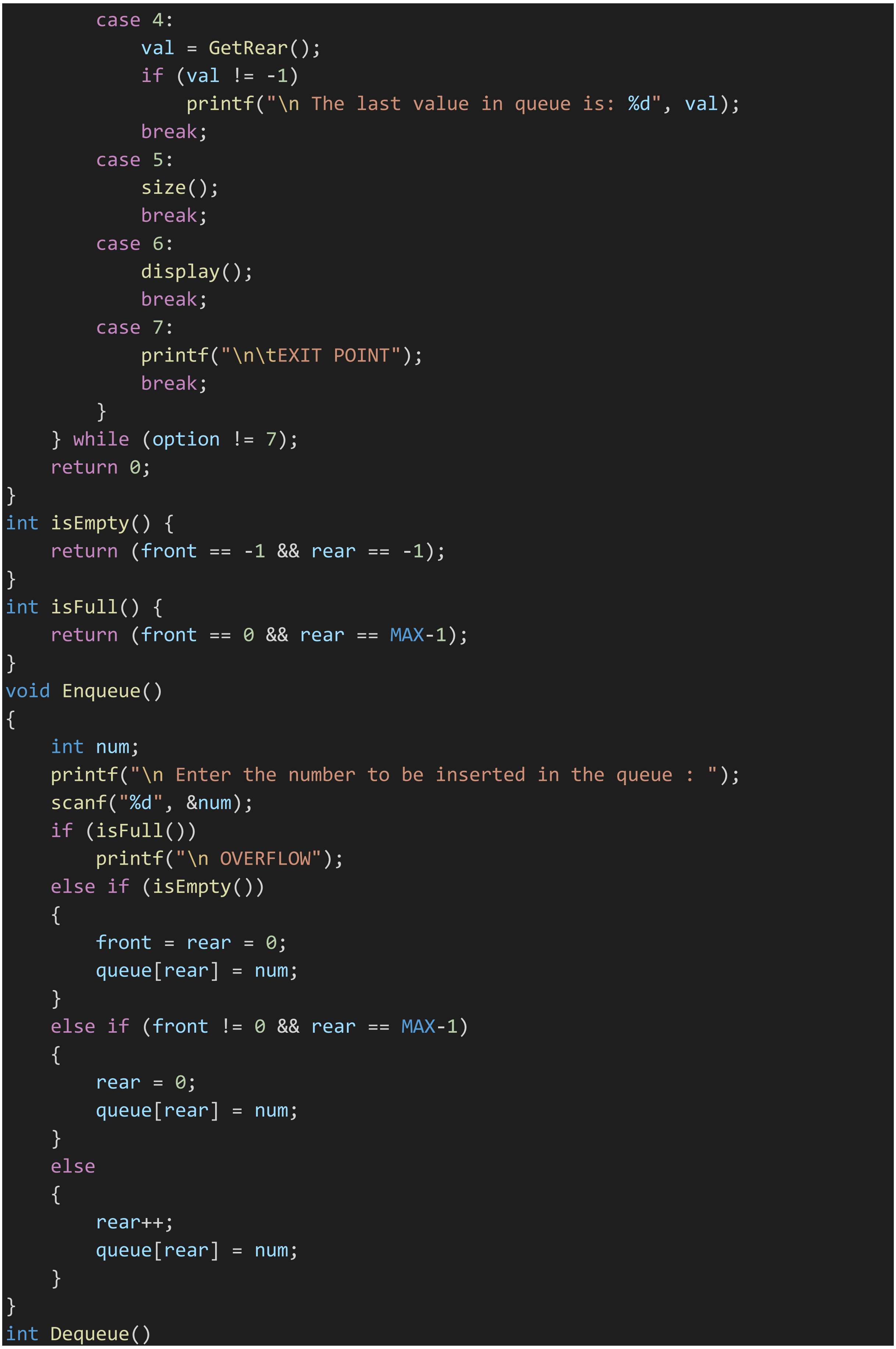


```
            size();
            break;
        case 6:
            display();
            break;
        case 7:
            printf("\n\tEXIT POINT");
            break;
        }
    } while (option != 7);
    return 0;
}
int isEmpty() {
    return (front == -1 && rear == -1);
}
int isFull() {
    return rear == MAX - 1;
}
void Enqueue()
{
    int num;
    printf("\n Enter the number to be inserted in the queue: ");
    scanf("%d", &num);
    if (isFull())
        printf("\n OVERFLOW");
    else if (front == -1 && rear == -1)
        front = rear = 0;
    else
        rear++;
    queue[rear] = num;
}
int Dequeue()
{
    int val;
    if (isEmpty())
    {
        printf("\n UNDERFLOW");
        return -1;
    }
    else
    {
        val = queue[front];
        if (front == rear) {
            front = rear = -1;
        }
        else {
            front++;
        }
```

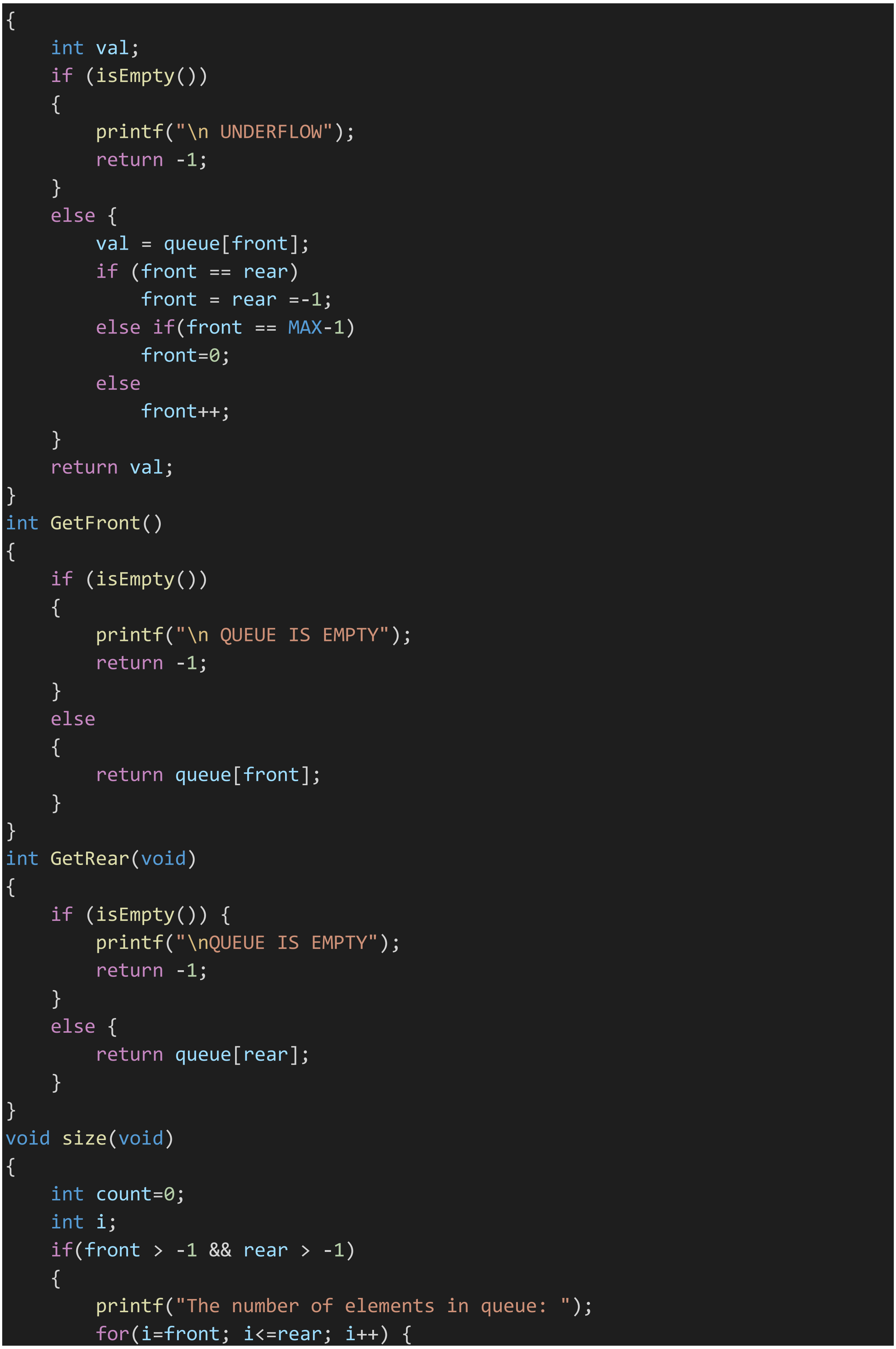


```
        return val;
    }
}
int GetFront()
{
    if (isEmpty()) {
        printf("\nQUEUE IS EMPTY");
        return -1;
    }
    else {
        return queue[front];
    }
}
int GetRear(void)
{
    if (isEmpty()) {
        printf("\nQUEUE IS EMPTY");
        return -1;
    }
    else {
        return queue[rear];
    }
}
void size(void)
{
    int count=0;
    int i;
    if(front > -1 && rear > -1)
    {
        printf("The number of elements in queue: ");
        for(i=front; i<=rear; i++) {
            count++;
        }
        printf("%d\n",count);
    }
    else
    {
        printf("\n The Queue is empty");
    }
}
void display()
{
    int i;
    printf("\n");
    if (isEmpty())
        printf("\nQUEUE IS EMPTY");
    else
    {
        printf("\nThe Linear Queue is: ");
        for (i=front; i<=rear; i++)
            printf("\t%d", queue[i]);
    }
}
```

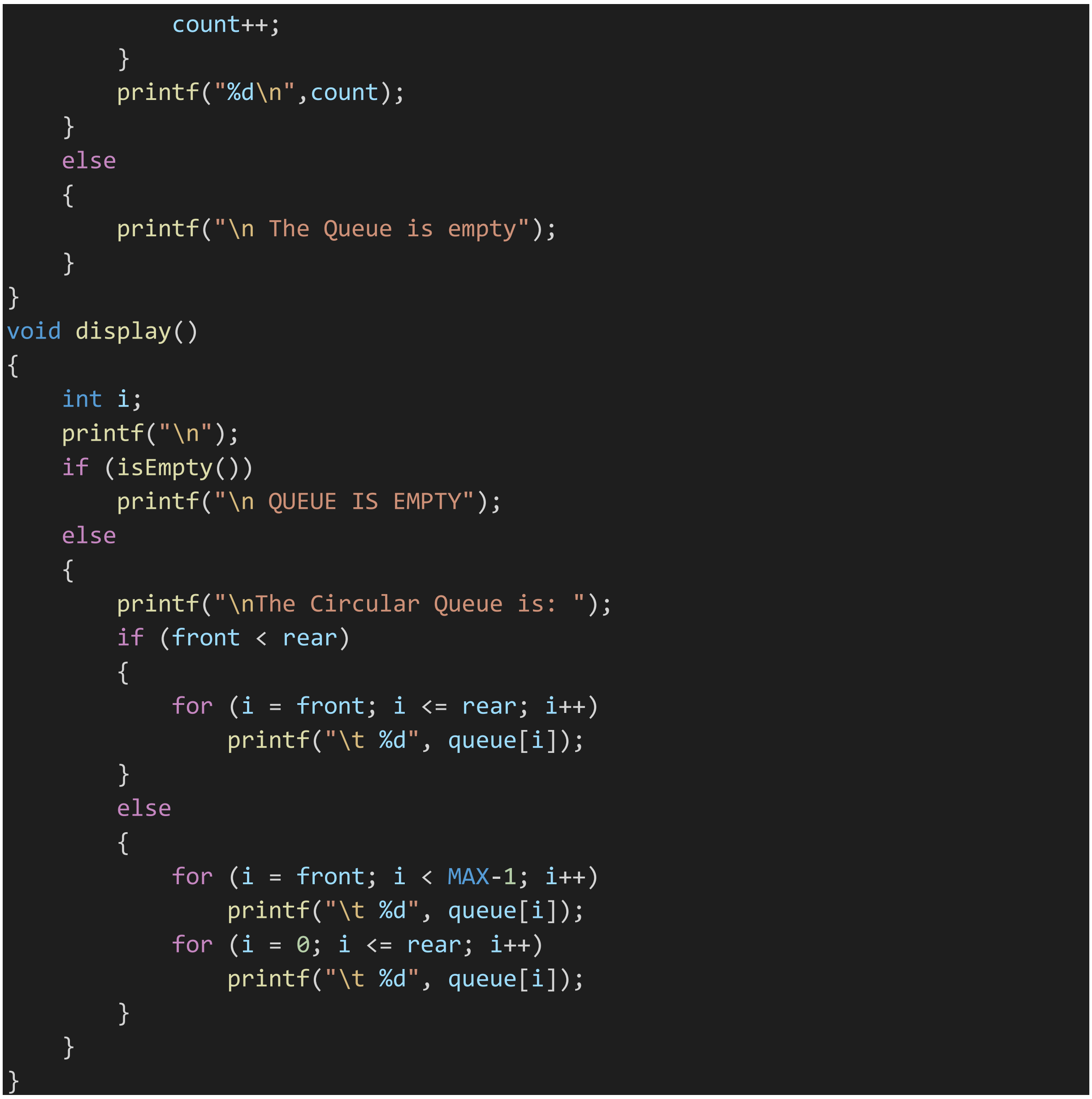
```
//WAP to implement Ciruclar Queue ADT using arrays
#include <stdio.h>
#include <stdlib.h>
#define MAX 10
int queue[MAX];
int front = -1, rear = -1;
void Enqueue(void);
int Dequeue(void);
int GetFront(void);
int GetRear(void);
void size(void);
void display(void);
int main()
{
    int option, val;
    printf("\n\n****List of Operations****");
    printf("\n 1. Enqueue");
    printf("\n 2. Dequeue");
    printf("\n 3. Get Front");
    printf("\n 4. Get Rear");
    printf("\n 5. Size");
    printf("\n 6. Display");
    printf("\n 7. EXIT");
    do
    {
        printf("\n Enter your option: ");
        scanf("%d", &option);
        switch (option)
        {
        case 1:
            Enqueue();
            break;
        case 2:
            val = Dequeue();
            if (val != -1)
                printf("\n The number deleted is: %d", val);
            break;
        case 3:
            val = GetFront();
            if (val != -1)
                printf("\n The first value in queue is: %d", val);
            break;
```



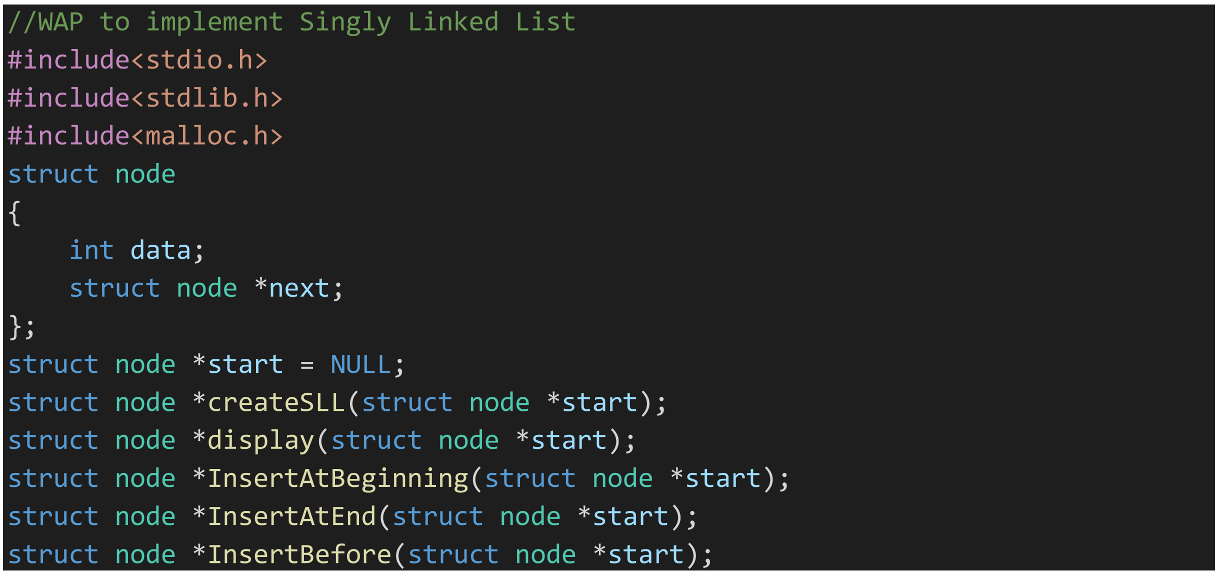
```
        case 4:
            val = GetRear();
            if (val != -1)
                printf("\n The last value in queue is: %d", val);
            break;
        case 5:
            size();
            break;
        case 6:
            display();
            break;
        case 7:
            printf("\n\tEXIT POINT");
            break;
        }
    } while (option != 7);
    return 0;
}
int isEmpty() {
    return (front == -1 && rear == -1);
}
int isFull() {
    return (front == 0 && rear == MAX-1);
}
void Enqueue()
{
    int num;
    printf("\n Enter the number to be inserted in the queue : ");
    scanf("%d", &num);
    if (isFull())
        printf("\n OVERFLOW");
    else if (isEmpty())
    {
        front = rear = 0;
        queue[rear] = num;
    }
    else if (front != 0 && rear == MAX-1)
    {
        rear = 0;
        queue[rear] = num;
    }
    else
    {
        rear++;
        queue[rear] = num;
    }
}
int Dequeue()
```



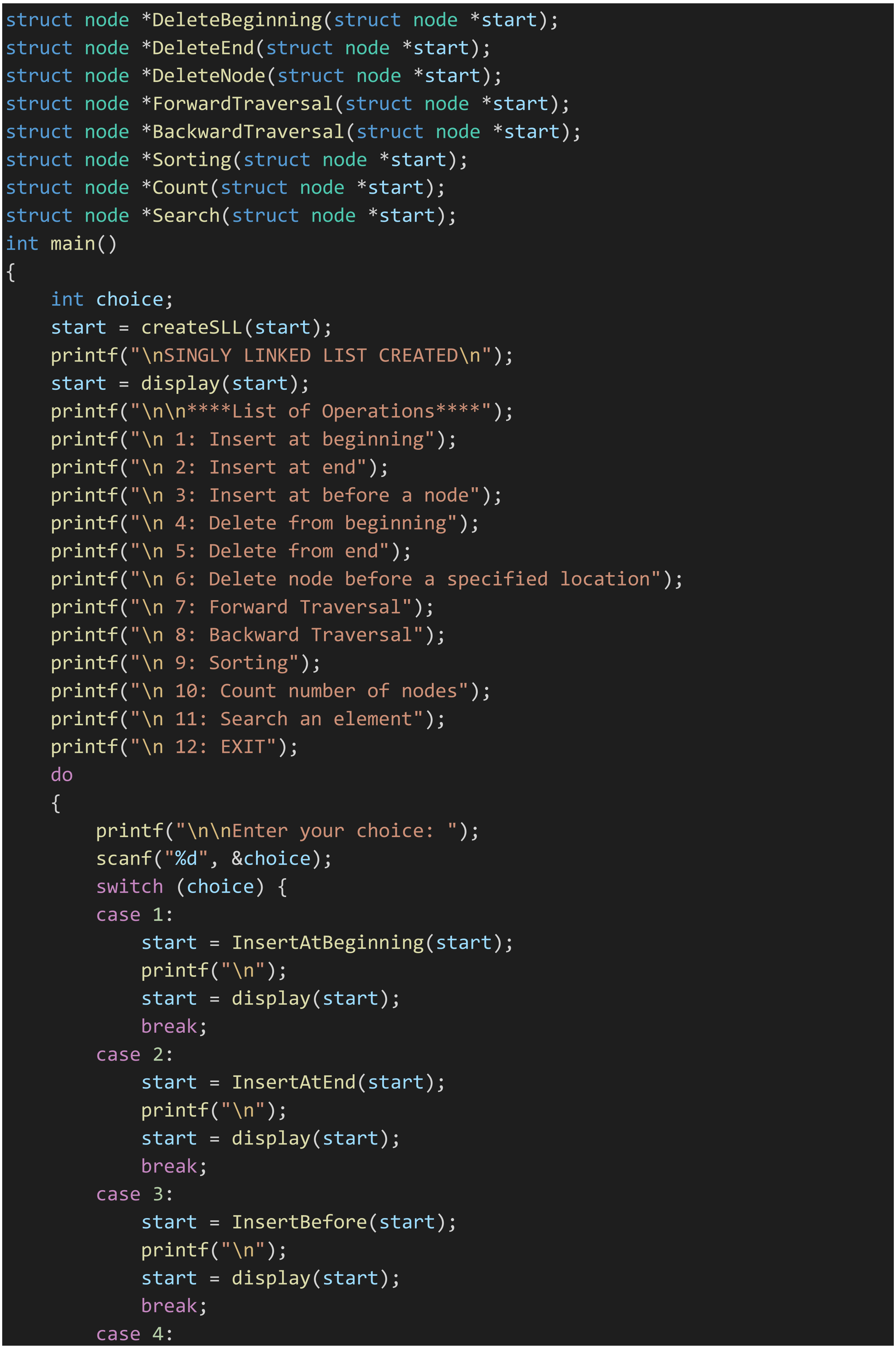
```
{
    int val;
    if (isEmpty())
    {
        printf("\n UNDERFLOW");
        return -1;
    }
    else {
        val = queue[front];
        if (front == rear)
            front = rear =-1;
        else if(front == MAX-1)
            front=0;
        else
            front++;
    }
    return val;
}
int GetFront()
{
    if (isEmpty())
    {
        printf("\n QUEUE IS EMPTY");
        return -1;
    }
    else
    {
        return queue[front];
    }
}
int GetRear(void)
{
    if (isEmpty()) {
        printf("\nQUEUE IS EMPTY");
        return -1;
    }
    else {
        return queue[rear];
    }
}
void size(void)
{
    int count=0;
    int i;
    if(front > -1 && rear > -1)
    {
        printf("The number of elements in queue: ");
        for(i=front; i<=rear; i++) {
```



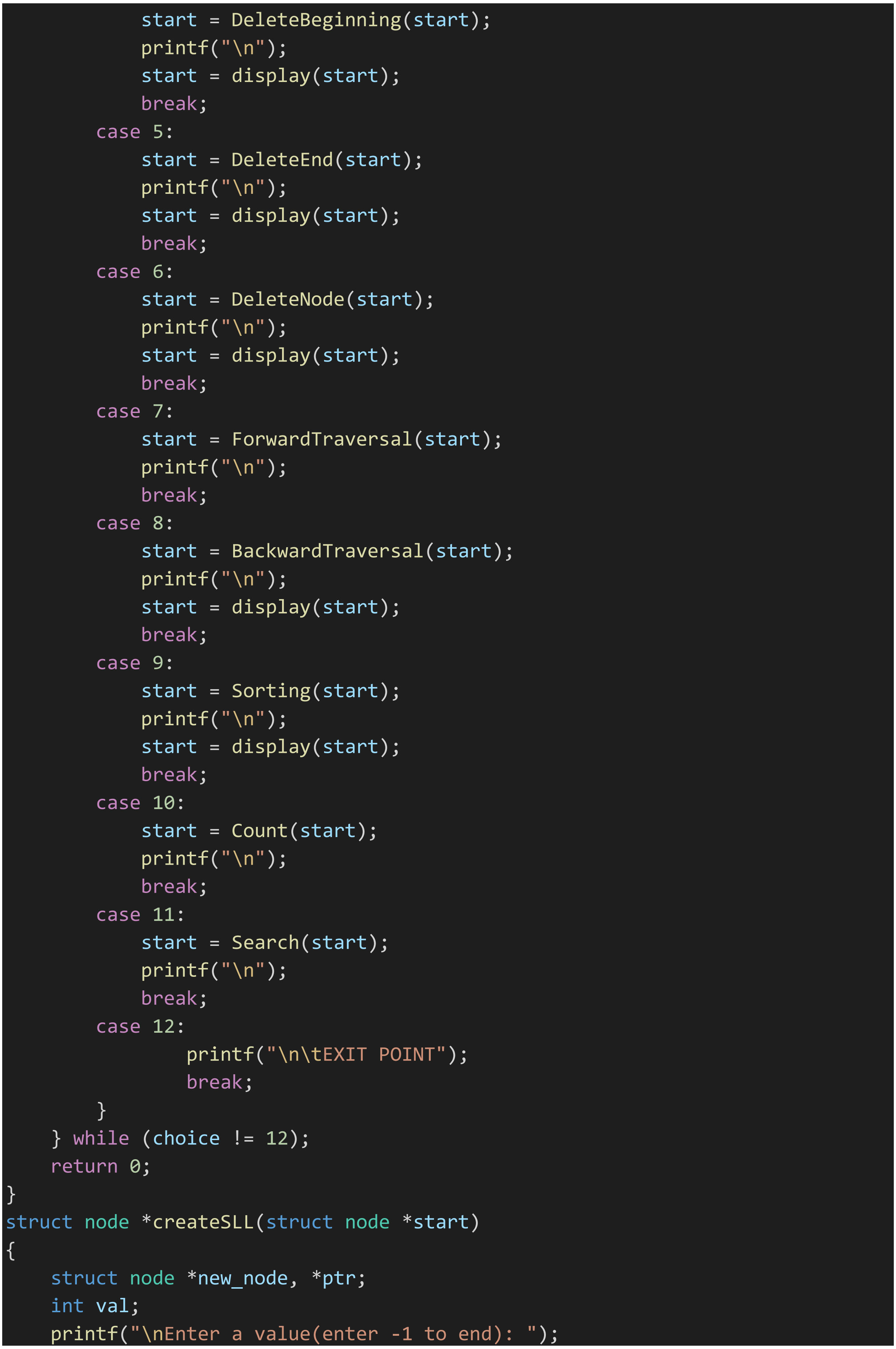
```
            count++;
        }
        printf("%d\n",count);
    }
    else
    {
        printf("\n The Queue is empty");
    }
}
void display()
{
    int i;
    printf("\n");
    if (isEmpty())
        printf("\n QUEUE IS EMPTY");
    else
    {
        printf("\nThe Circular Queue is: ");
        if (front < rear)
        {
            for (i = front; i <= rear; i++)
                printf("\t %d", queue[i]);
        }
        else
        {
            for (i = front; i < MAX-1; i++)
                printf("\t %d", queue[i]);
            for (i = 0; i <= rear; i++)
                printf("\t %d", queue[i]);
        }
    }
}
```



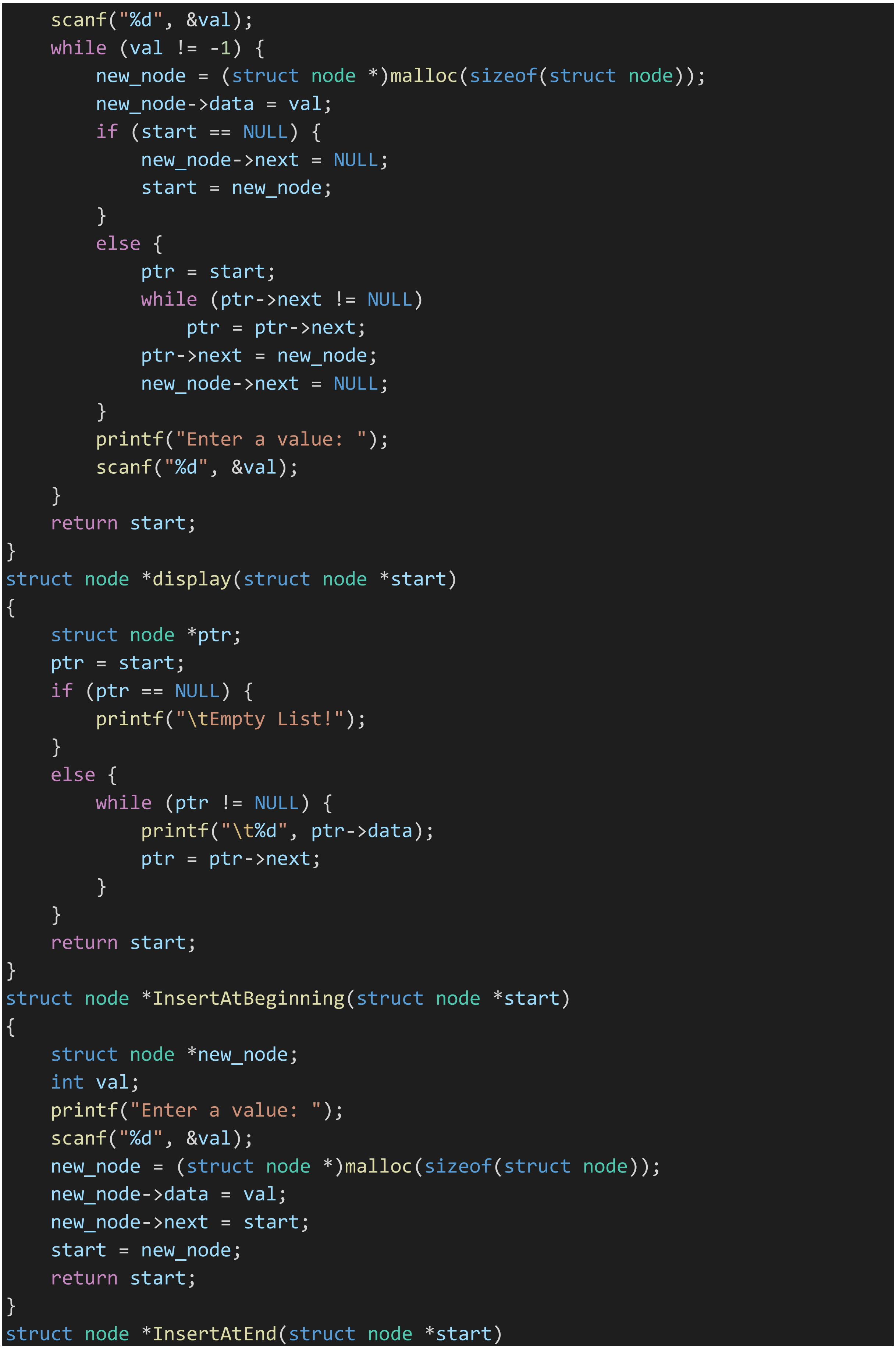
```
//WAP to implement Singly Linked List
#include<stdio.h>
#include<stdlib.h>
#include<malloc.h>
struct node
{
    int data;
    struct node *next;
};
struct node *start = NULL;
struct node *createSLL(struct node *start);
struct node *display(struct node *start);
struct node *InsertAtBeginning(struct node *start);
struct node *InsertAtEnd(struct node *start);
struct node *InsertBefore(struct node *start);
```



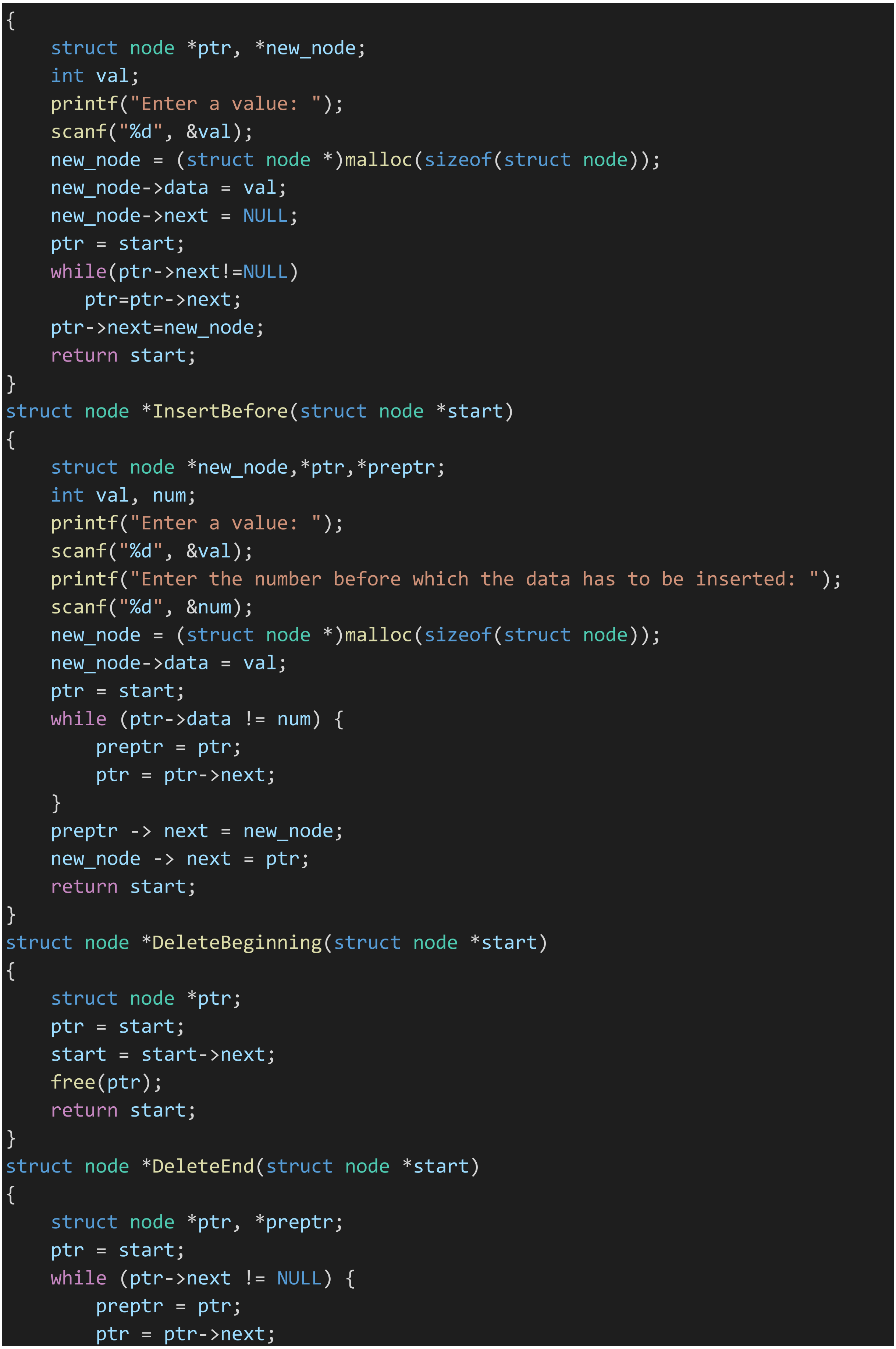
```
struct node *DeleteBeginning(struct node *start);
struct node *DeleteEnd(struct node *start);
struct node *DeleteNode(struct node *start);
struct node *ForwardTraversal(struct node *start);
struct node *BackwardTraversal(struct node *start);
struct node *Sorting(struct node *start);
struct node *Count(struct node *start);
struct node *Search(struct node *start);
int main()
{
    int choice;
    start = createSLL(start);
    printf("\nSINGLY LINKED LIST CREATED\n");
    start = display(start);
    printf("\n\n****List of Operations****");
    printf("\n 1: Insert at beginning");
    printf("\n 2: Insert at end");
    printf("\n 3: Insert at before a node");
    printf("\n 4: Delete from beginning");
    printf("\n 5: Delete from end");
    printf("\n 6: Delete node before a specified location");
    printf("\n 7: Forward Traversal");
    printf("\n 8: Backward Traversal");
    printf("\n 9: Sorting");
    printf("\n 10: Count number of nodes");
    printf("\n 11: Search an element");
    printf("\n 12: EXIT");
    do
    {
        printf("\n\nEnter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
        case 1:
            start = InsertAtBeginning(start);
            printf("\n");
            start = display(start);
            break;
        case 2:
            start = InsertAtEnd(start);
            printf("\n");
            start = display(start);
            break;
        case 3:
            start = InsertBefore(start);
            printf("\n");
            start = display(start);
            break;
        case 4:
```



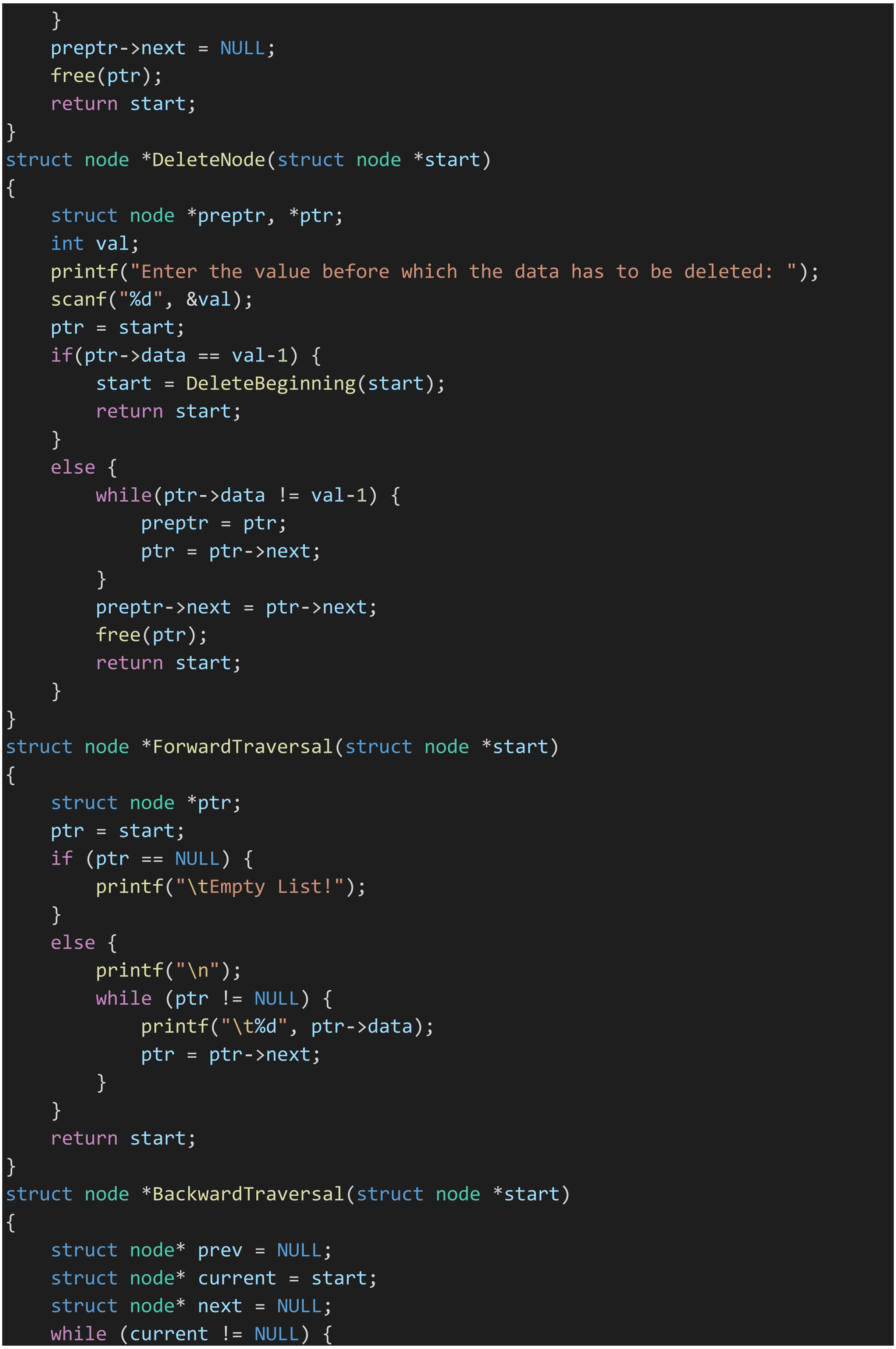
```
            start = DeleteBeginning(start);
            printf("\n");
            start = display(start);
            break;
        case 5:
            start = DeleteEnd(start);
            printf("\n");
            start = display(start);
            break;
        case 6:
            start = DeleteNode(start);
            printf("\n");
            start = display(start);
            break;
        case 7:
            start = ForwardTraversal(start);
            printf("\n");
            break;
        case 8:
            start = BackwardTraversal(start);
            printf("\n");
            start = display(start);
            break;
        case 9:
            start = Sorting(start);
            printf("\n");
            start = display(start);
            break;
        case 10:
            start = Count(start);
            printf("\n");
            break;
        case 11:
            start = Search(start);
            printf("\n");
            break;
        case 12:
                printf("\n\tEXIT POINT");
                break;
        }
    } while (choice != 12);
    return 0;
}
struct node *createSLL(struct node *start)
{
    struct node *new_node, *ptr;
    int val;
    printf("\nEnter a value(enter -1 to end): ");
```



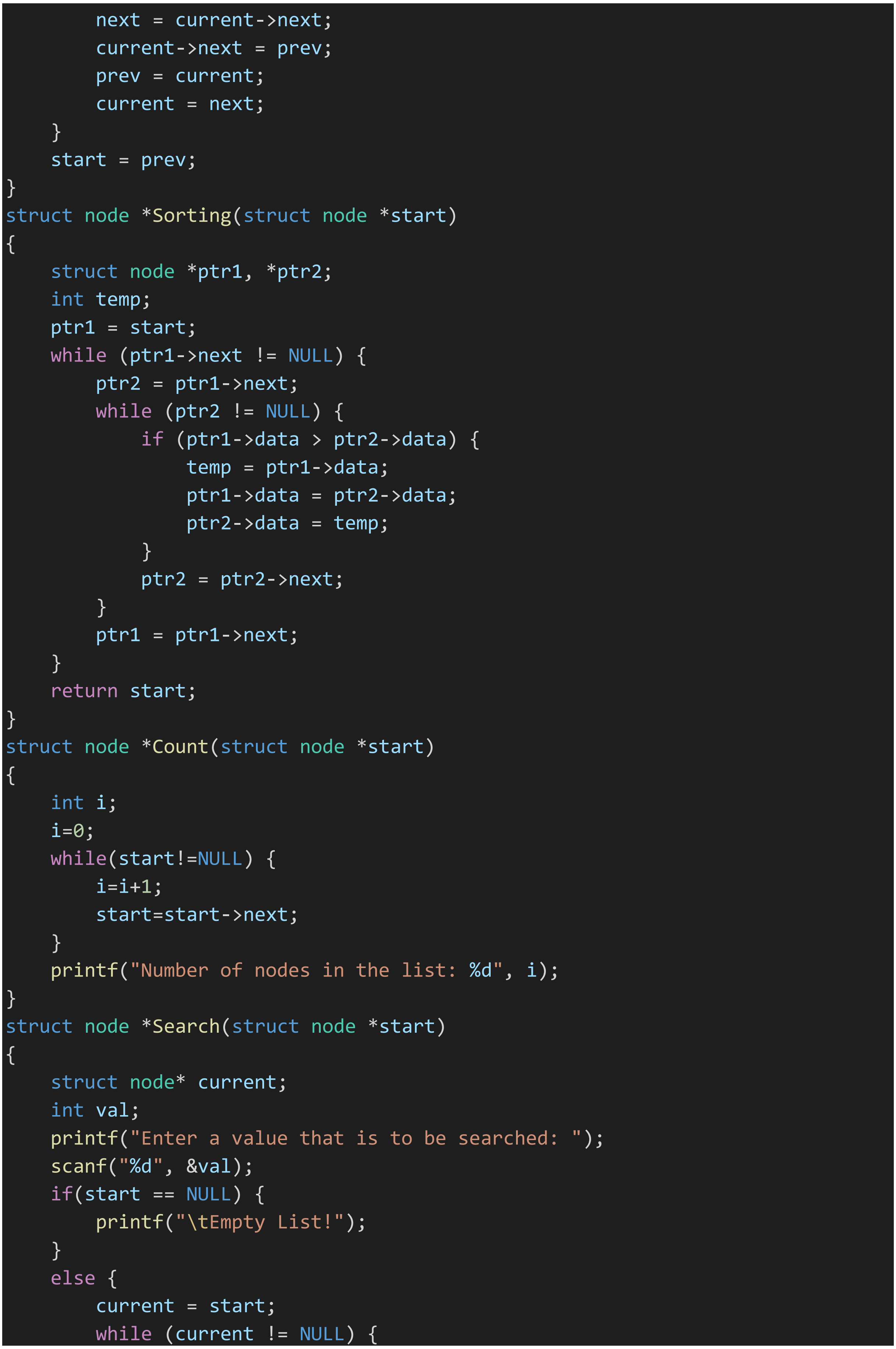
```
    scanf("%d", &val);
    while (val != -1) {
        new_node = (struct node *)malloc(sizeof(struct node));
        new_node->data = val;
        if (start == NULL) {
            new_node->next = NULL;
            start = new_node;
        }
        else {
            ptr = start;
            while (ptr->next != NULL)
                ptr = ptr->next;
            ptr->next = new_node;
            new_node->next = NULL;
        }
        printf("Enter a value: ");
        scanf("%d", &val);
    }
    return start;
}
struct node *display(struct node *start)
{
    struct node *ptr;
    ptr = start;
    if (ptr == NULL) {
        printf("\tEmpty List!");
    }
    else {
        while (ptr != NULL) {
            printf("\t%d", ptr->data);
            ptr = ptr->next;
        }
    }
    return start;
}
struct node *InsertAtBeginning(struct node *start)
{
    struct node *new_node;
    int val;
    printf("Enter a value: ");
    scanf("%d", &val);
    new_node = (struct node *)malloc(sizeof(struct node));
    new_node->data = val;
    new_node->next = start;
    start = new_node;
    return start;
}
struct node *InsertAtEnd(struct node *start)
```



```
{
    struct node *ptr, *new_node;
    int val;
    printf("Enter a value: ");
    scanf("%d", &val);
    new_node = (struct node *)malloc(sizeof(struct node));
    new_node->data = val;
    new_node->next = NULL;
    ptr = start;
    while(ptr->next!=NULL)
       ptr=ptr->next;
    ptr->next=new_node;
    return start;
}
struct node *InsertBefore(struct node *start)
{
    struct node *new_node,*ptr,*preptr;
    int val, num;
    printf("Enter a value: ");
    scanf("%d", &val);
    printf("Enter the number before which the data has to be inserted: ");
    scanf("%d", &num);
    new_node = (struct node *)malloc(sizeof(struct node));
    new_node->data = val;
    ptr = start;
    while (ptr->data != num) {
        preptr = ptr;
        ptr = ptr->next;
    }
    preptr -> next = new_node;
    new_node -> next = ptr;
    return start;
}
struct node *DeleteBeginning(struct node *start)
{
    struct node *ptr;
    ptr = start;
    start = start->next;
    free(ptr);
    return start;
}
struct node *DeleteEnd(struct node *start)
{
    struct node *ptr, *preptr;
    ptr = start;
    while (ptr->next != NULL) {
        preptr = ptr;
        ptr = ptr->next;
```

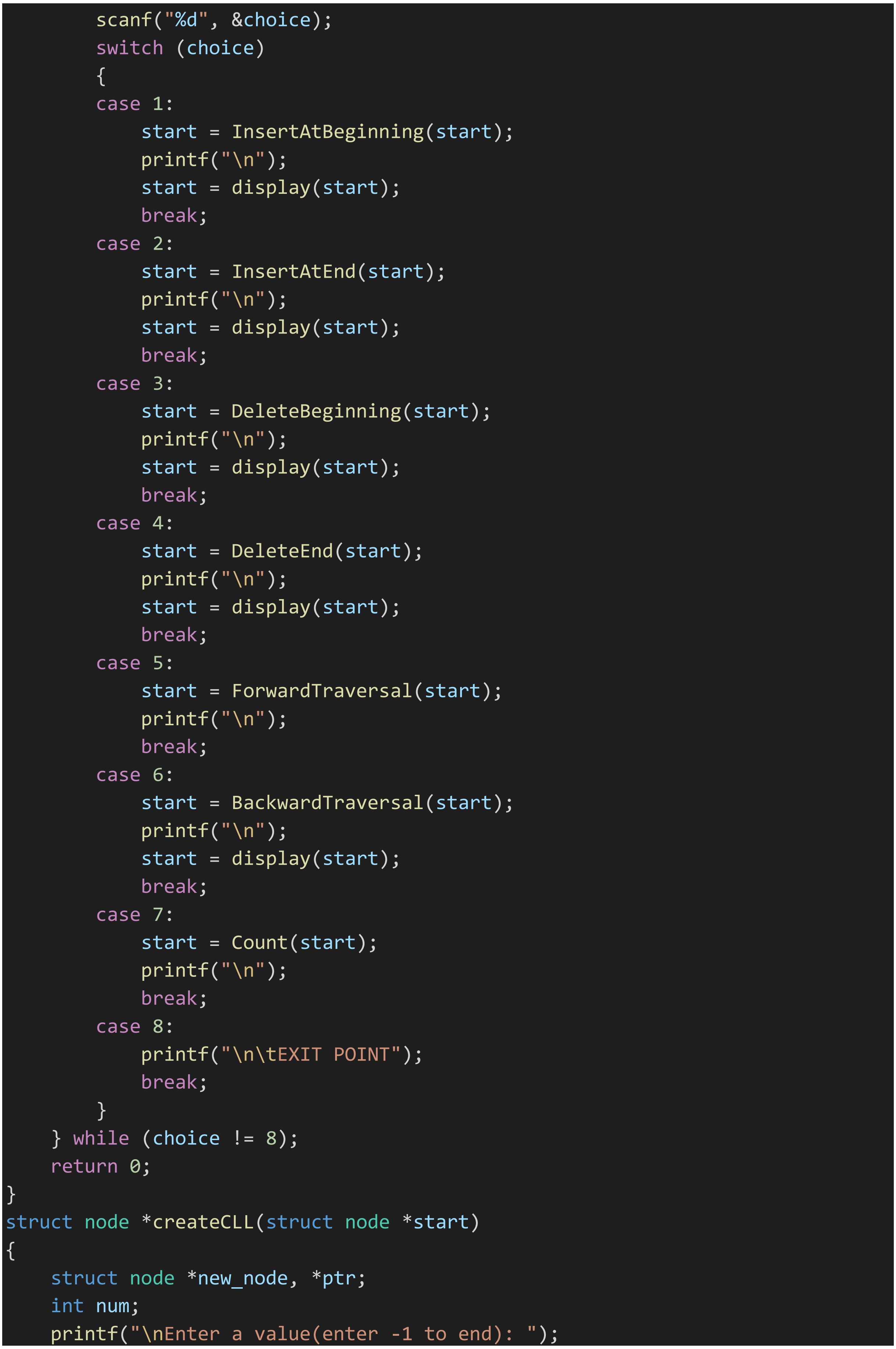


```
    }
    preptr->next = NULL;
    free(ptr);
    return start;
}
struct node *DeleteNode(struct node *start)
{
    struct node *preptr, *ptr;
    int val;
    printf("Enter the value before which the data has to be deleted: ");
    scanf("%d", &val);
    ptr = start;
    if(ptr->data == val-1) {
        start = DeleteBeginning(start);
        return start;
    }
    else {
        while(ptr->data != val-1) {
            preptr = ptr;
            ptr = ptr->next;
        }
        preptr->next = ptr->next;
        free(ptr);
        return start;
    }
}
struct node *ForwardTraversal(struct node *start)
{
    struct node *ptr;
    ptr = start;
    if (ptr == NULL) {
        printf("\tEmpty List!");
    }
    else {
        printf("\n");
        while (ptr != NULL) {
            printf("\t%d", ptr->data);
            ptr = ptr->next;
        }
    }
    return start;
}
struct node *BackwardTraversal(struct node *start)
{
    struct node* prev = NULL;
    struct node* current = start;
    struct node* next = NULL;
    while (current != NULL) {
```

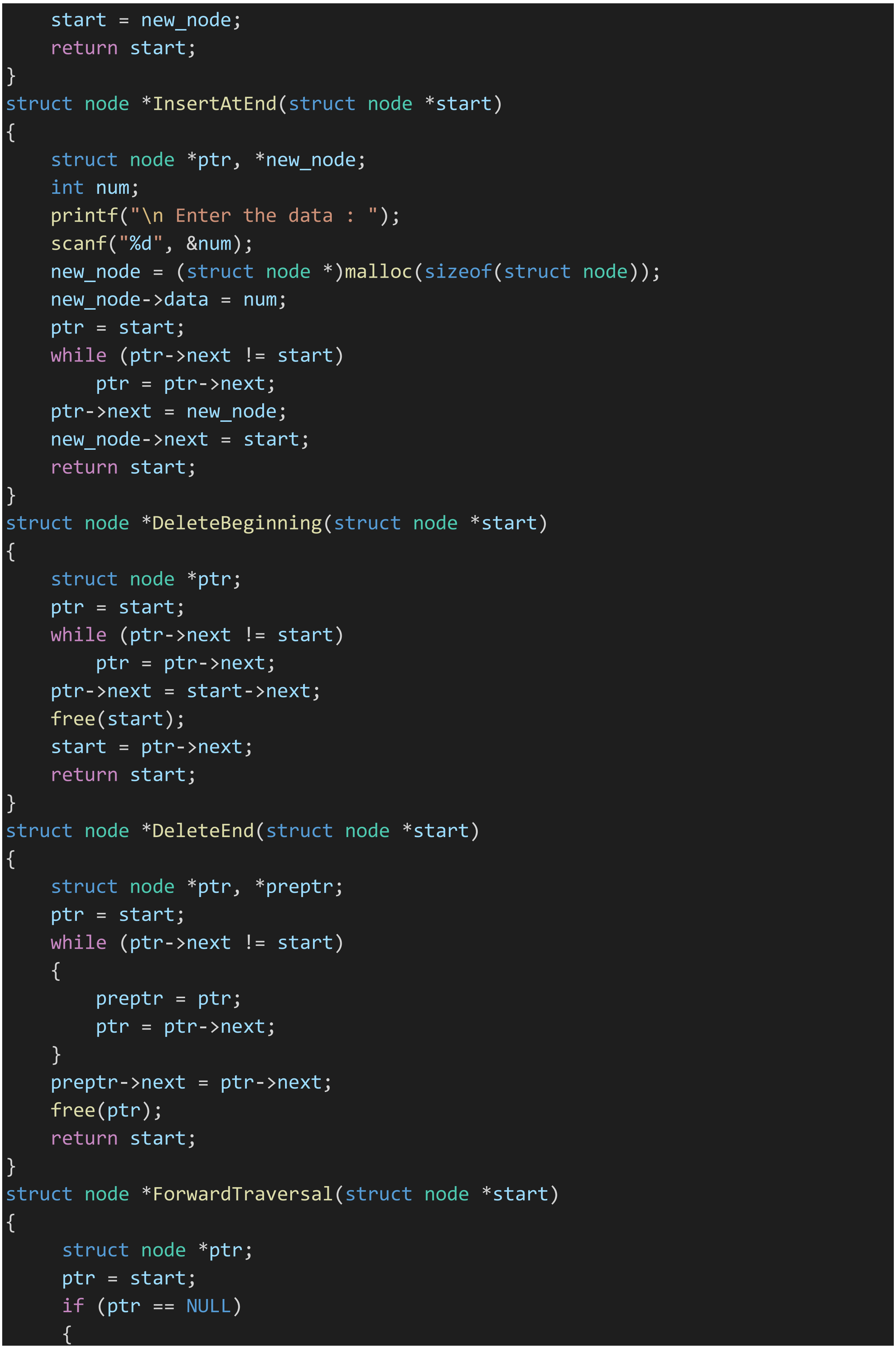


```
        next = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }
    start = prev;
}
struct node *Sorting(struct node *start)
{
    struct node *ptr1, *ptr2;
    int temp;
    ptr1 = start;
    while (ptr1->next != NULL) {
        ptr2 = ptr1->next;
        while (ptr2 != NULL) {
            if (ptr1->data > ptr2->data) {
                temp = ptr1->data;
                ptr1->data = ptr2->data;
                ptr2->data = temp;
            }
            ptr2 = ptr2->next;
        }
        ptr1 = ptr1->next;
    }
    return start;
}
struct node *Count(struct node *start)
{
    int i;
    i=0;
    while(start!=NULL) {
        i=i+1;
        start=start->next;
    }
    printf("Number of nodes in the list: %d", i);
}
struct node *Search(struct node *start)
{
    struct node* current;
    int val;
    printf("Enter a value that is to be searched: ");
    scanf("%d", &val);
    if(start == NULL) {
        printf("\tEmpty List!");
    }
    else {
        current = start;
        while (current != NULL) {
            if (current -> data == val)
            printf("\tElement found");
            break;
        }
        current = current->next;
    }
    if(current == NULL) {
        printf("\tElement not found");
    }
}
```

```
//WAP to implement Circular Linked List
#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
struct node
{
    int data;
    struct node *next;
};
struct node *start = NULL;
struct node *createCLL(struct node *start);
struct node *display(struct node *start);
struct node *InsertAtBeginning(struct node *start);
struct node *InsertAtEnd(struct node *start);
struct node *DeleteBeginning(struct node *start);
struct node *DeleteEnd(struct node *start);
struct node *ForwardTraversal(struct node *start);
struct node *BackwardTraversal(struct node *start);
struct node *Count(struct node *start);
int main()
{
    int choice;
    start = createCLL(start);
    printf("\nCIRCULAR LINKED LIST CREATED\n");
    start = display(start);
    printf("\n\n****List of Operations****");
    printf("\n 1: Insert at beginning");
    printf("\n 2: Insert at end");
    printf("\n 3: Delete from beginning");
    printf("\n 4: Delete from end");
    printf("\n 5: Forward Traversal");
    printf("\n 6: Backward Traversal");
    printf("\n 7: Count number of nodes");
    printf("\n 8: EXIT");
    do
    {
        printf("\n\nEnter your choice: ");
```

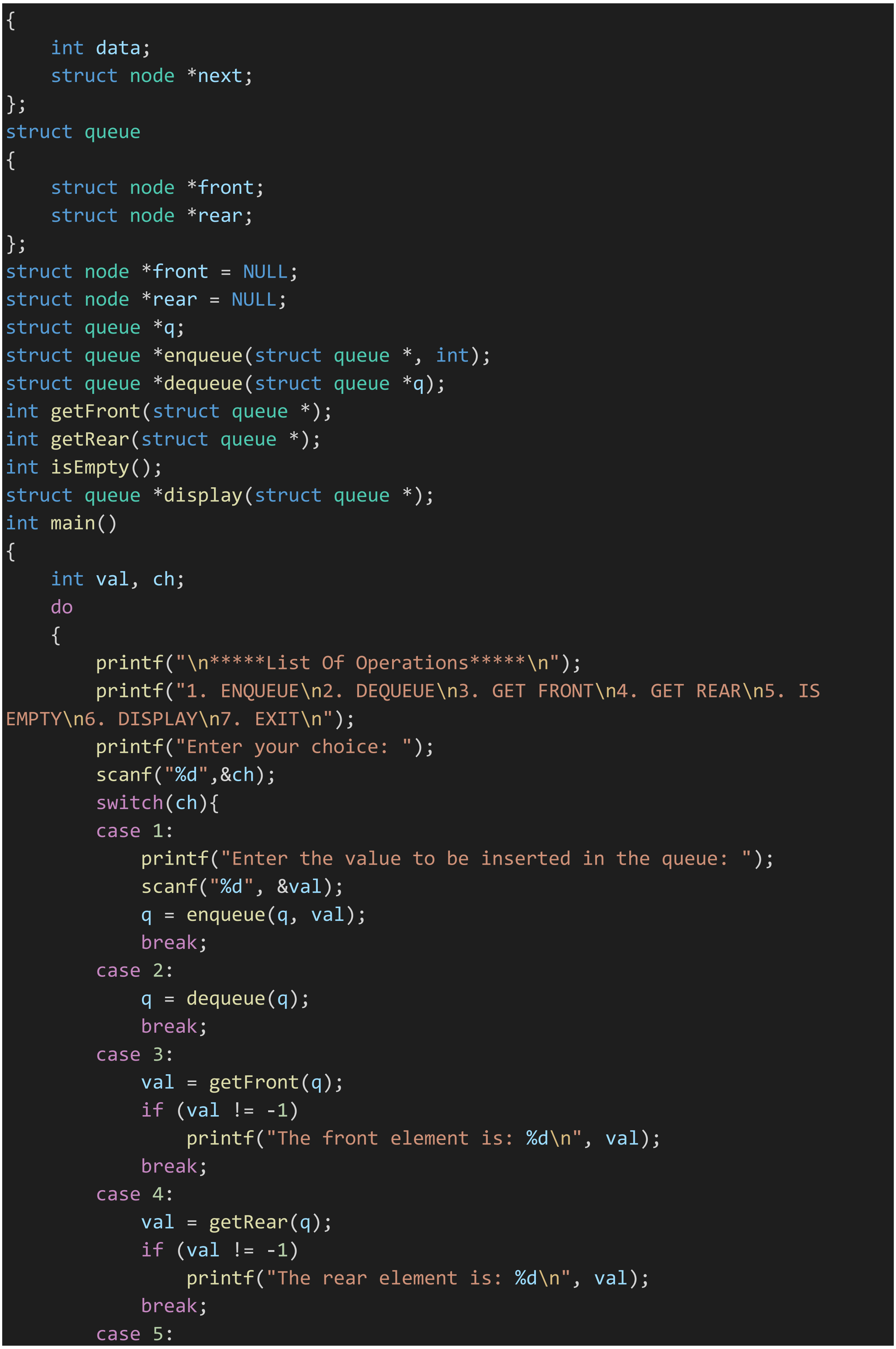


```
        scanf("%d", &choice);
        switch (choice)
        {
        case 1:
            start = InsertAtBeginning(start);
            printf("\n");
            start = display(start);
            break;
        case 2:
            start = InsertAtEnd(start);
            printf("\n");
            start = display(start);
            break;
        case 3:
            start = DeleteBeginning(start);
            printf("\n");
            start = display(start);
            break;
        case 4:
            start = DeleteEnd(start);
            printf("\n");
            start = display(start);
            break;
        case 5:
            start = ForwardTraversal(start);
            printf("\n");
            break;
        case 6:
            start = BackwardTraversal(start);
            printf("\n");
            start = display(start);
            break;
        case 7:
            start = Count(start);
            printf("\n");
            break;
        case 8:
            printf("\n\tEXIT POINT");
            break;
        }
    } while (choice != 8);
    return 0;
}
struct node *createCLL(struct node *start)
{
    struct node *new_node, *ptr;
    int num;
    printf("\nEnter a value(enter -1 to end): ");
    scanf("%d", &num);
    while (num != -1)
    {
        new_node = (struct node *)malloc(sizeof(struct node));
        new_node->data = num;
        if (start == NULL)
        {
            new_node->next = new_node;
            start = new_node;
        }
        else
        {
            ptr = start;
            while (ptr->next != start)
                ptr = ptr->next;
            ptr->next = new_node;
            new_node->next = start;
        }
        printf("Enter a value: ");
        scanf("%d", &num);
    }
    return start;
}
struct node *display(struct node *start)
{
    struct node *ptr;
    ptr = start;
    while (ptr->next != start)
    {
        printf("\t%d", ptr->data);
        ptr = ptr->next;
    }
    printf("\t%d", ptr->data);
    return start;
}
struct node *InsertAtBeginning(struct node *start)
{
    struct node *new_node, *ptr;
    int num;
    printf("Enter a value: ");
    scanf("%d", &num);
    new_node = (struct node *)malloc(sizeof(struct node));
    new_node->data = num;
    ptr = start;
    while (ptr->next != start)
        ptr = ptr->next;
    ptr->next = new_node;
    new_node->next = start;
```

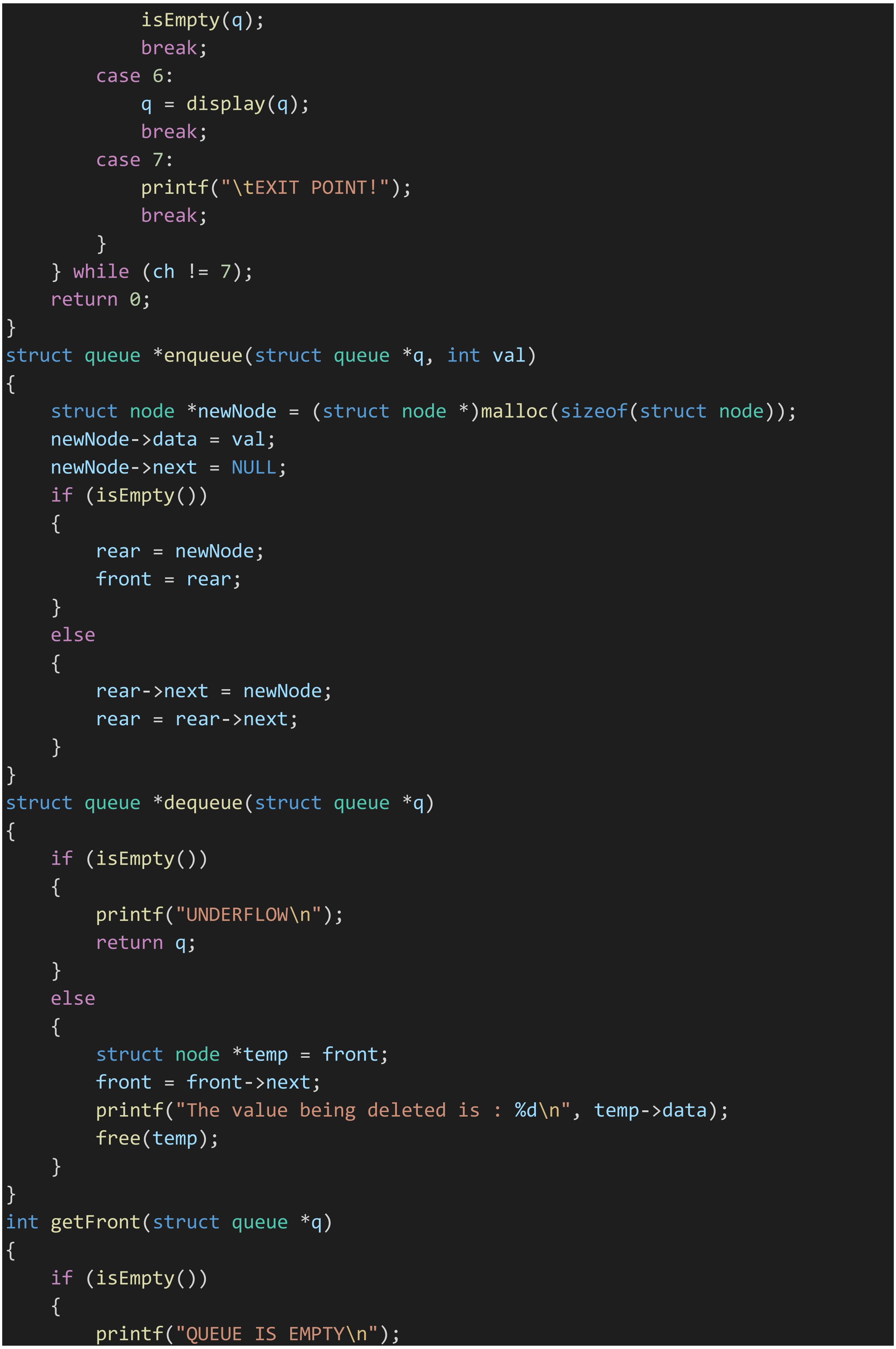


```
    start = new_node;
    return start;
}
struct node *InsertAtEnd(struct node *start)
{
    struct node *ptr, *new_node;
    int num;
    printf("\n Enter the data : ");
    scanf("%d", &num);
    new_node = (struct node *)malloc(sizeof(struct node));
    new_node->data = num;
    ptr = start;
    while (ptr->next != start)
        ptr = ptr->next;
    ptr->next = new_node;
    new_node->next = start;
    return start;
}
struct node *DeleteBeginning(struct node *start)
{
    struct node *ptr;
    ptr = start;
    while (ptr->next != start)
        ptr = ptr->next;
    ptr->next = start->next;
    free(start);
    start = ptr->next;
    return start;
}
struct node *DeleteEnd(struct node *start)
{
    struct node *ptr, *preptr;
    ptr = start;
    while (ptr->next != start)
    {
        preptr = ptr;
        ptr = ptr->next;
    }
    preptr->next = ptr->next;
    free(ptr);
    return start;
}
struct node *ForwardTraversal(struct node *start)
{
     struct node *ptr;
     ptr = start;
     if (ptr == NULL)
     {
         printf("\tEmpty List!");
     }
     else
     {
         printf("\n");
         while (ptr->next != start)
         {
             printf("\t%d", ptr->data);
             ptr = ptr->next;
         }
         printf("\t%d", ptr->data);
     }
     return start;
}
struct node *BackwardTraversal(struct node *start)
{
    struct node* prev = start;
    struct node *current = start;
    struct node *temp = start;
    current=current->next;
    temp=temp->next->next;
    while (current != start)
    {

        current->next = prev;
        prev = current;
        current = temp;
        temp = current->next;
    }
    start = prev;
    current->next = start;
}
struct node *Count(struct node *start)
{
    int i=0;
    struct node *current = start;
    do
    {
        start = start->next;
        i++;
    } while (current != start);
    printf("Number of nodes in the list: %d", i);
}
```

```
//WAP to implement Linear Queue ADT using Linked List
#include <stdio.h>
#include <stdlib.h>
struct node
```



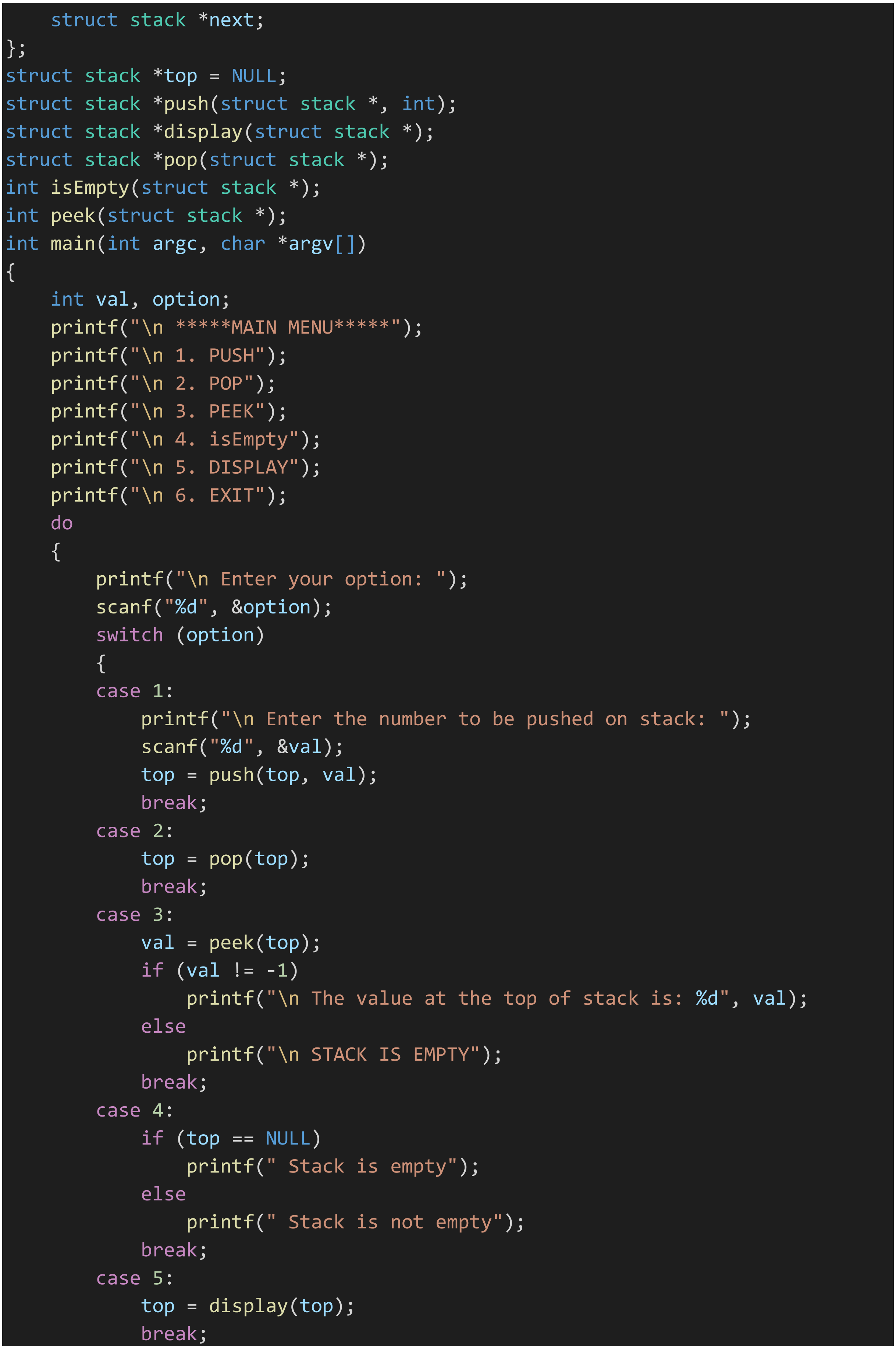
```c
{
    int data;
    struct node *next;
};
struct queue
{
    struct node *front;
    struct node *rear;
};
struct node *front = NULL;
struct node *rear = NULL;
struct queue *q;
struct queue *enqueue(struct queue *, int);
struct queue *dequeue(struct queue *q);
int getFront(struct queue *);
int getRear(struct queue *);
int isEmpty();
struct queue *display(struct queue *);
int main()
{
    int val, ch;
    do
    {
        printf("\n*****List Of Operations*****\n");
        printf("1. ENQUEUE\n2. DEQUEUE\n3. GET FRONT\n4. GET REAR\n5. IS
EMPTY\n6. DISPLAY\n7. EXIT\n");
        printf("Enter your choice: ");
        scanf("%d",&ch);
        switch(ch){
        case 1:
            printf("Enter the value to be inserted in the queue: ");
            scanf("%d", &val);
            q = enqueue(q, val);
            break;
        case 2:
            q = dequeue(q);
            break;
        case 3:
            val = getFront(q);
            if (val != -1)
                printf("The front element is: %d\n", val);
            break;
        case 4:
            val = getRear(q);
            if (val != -1)
                printf("The rear element is: %d\n", val);
            break;
        case 5:
```



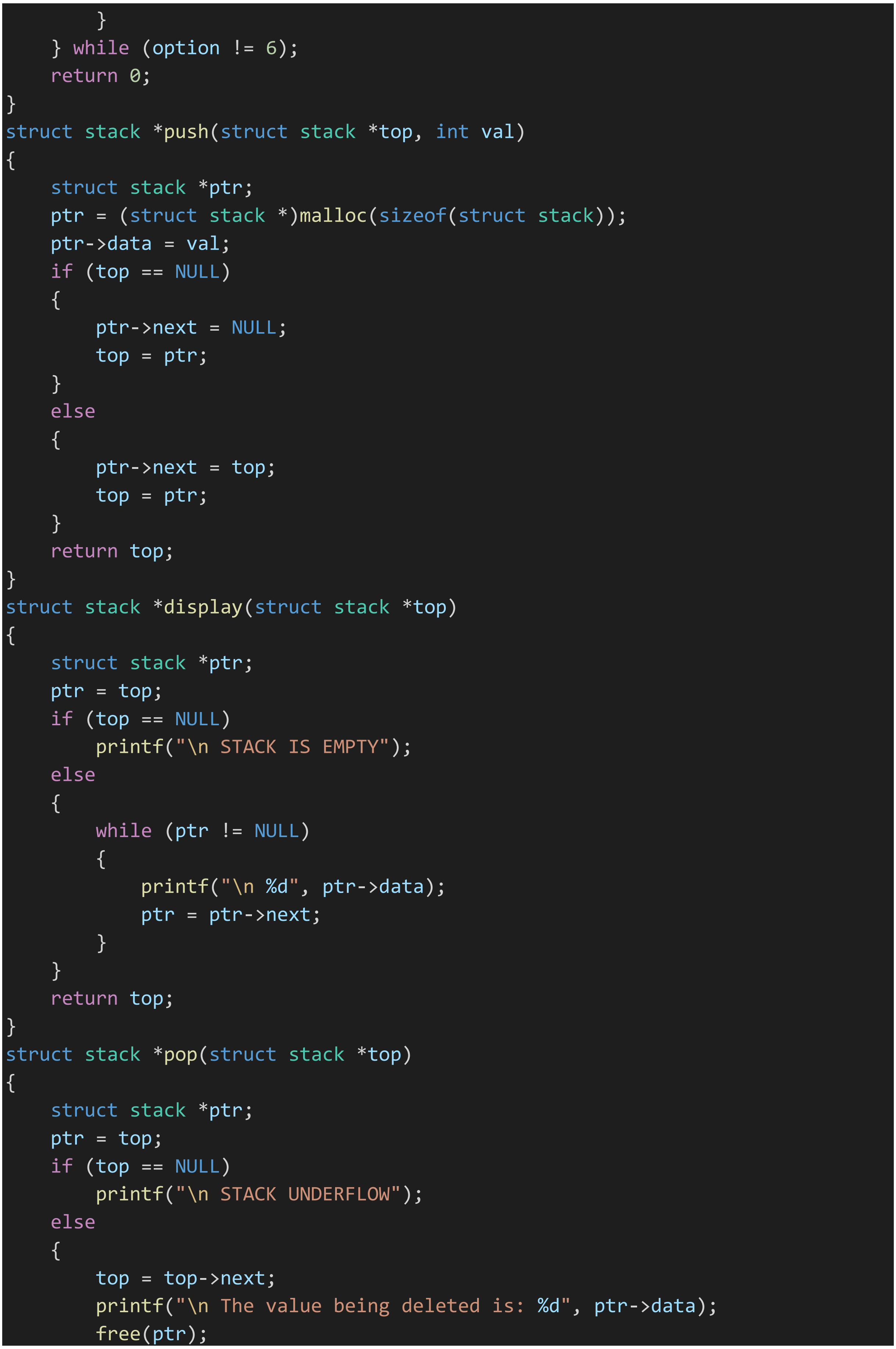
```
            isEmpty(q);
            break;
        case 6:
            q = display(q);
            break;
        case 7:
            printf("\tEXIT POINT!");
            break;
        }
    } while (ch != 7);
    return 0;
}
struct queue *enqueue(struct queue *q, int val)
{
    struct node *newNode = (struct node *)malloc(sizeof(struct node));
    newNode->data = val;
    newNode->next = NULL;
    if (isEmpty())
    {
        rear = newNode;
        front = rear;
    }
    else
    {
        rear->next = newNode;
        rear = rear->next;
    }
}
struct queue *dequeue(struct queue *q)
{
    if (isEmpty())
    {
        printf("UNDERFLOW\n");
        return q;
    }
    else
    {
        struct node *temp = front;
        front = front->next;
        printf("The value being deleted is : %d\n", temp->data);
        free(temp);
    }
}
int getFront(struct queue *q)
{
    if (isEmpty())
    {
        printf("QUEUE IS EMPTY\n");
```

```c
        return -1;
    }
    int val = front->data;
    return val;
}
int getRear(struct queue *q)
{
    if (isEmpty())
    {
        printf("QUEUE IS EMPTY\n");
        return -1;
    }
    int val = rear->data;
    return val;
}
int isEmpty()
{
    if (front == NULL && rear == NULL)
    {
        return -1;
    }
    return 0;
}
struct queue *display(struct queue *q)
{
    if (isEmpty())
    {
        printf("QUEUE IS EMPTY\n");
        return q;
    }
    struct node *temp = front;
    printf("The Queue is: ");
    while (temp != NULL)
    {
        printf("\t%d", temp->data);
        temp = temp->next;
    }
    printf("\tNULL\n");
}
```

```c
// WAP to implement Stack ADT using Linked List
#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
struct stack
{
    int data;
```

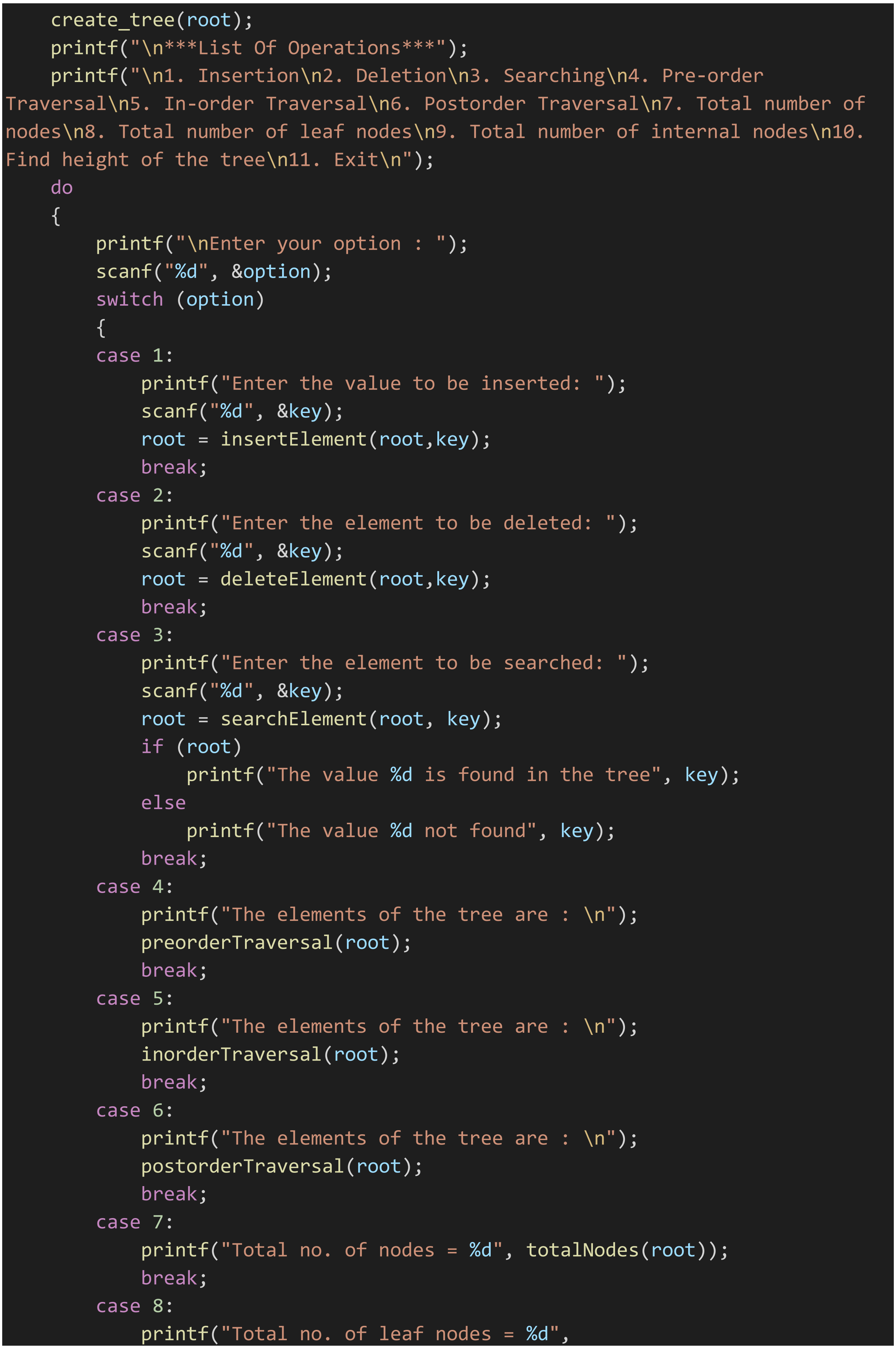


```
    struct stack *next;
};
struct stack *top = NULL;
struct stack *push(struct stack *, int);
struct stack *display(struct stack *);
struct stack *pop(struct stack *);
int isEmpty(struct stack *);
int peek(struct stack *);
int main(int argc, char *argv[])
{
    int val, option;
    printf("\n *****MAIN MENU*****");
    printf("\n 1. PUSH");
    printf("\n 2. POP");
    printf("\n 3. PEEK");
    printf("\n 4. isEmpty");
    printf("\n 5. DISPLAY");
    printf("\n 6. EXIT");
    do
    {
        printf("\n Enter your option: ");
        scanf("%d", &option);
        switch (option)
        {
        case 1:
            printf("\n Enter the number to be pushed on stack: ");
            scanf("%d", &val);
            top = push(top, val);
            break;
        case 2:
            top = pop(top);
            break;
        case 3:
            val = peek(top);
            if (val != -1)
                printf("\n The value at the top of stack is: %d", val);
            else
                printf("\n STACK IS EMPTY");
            break;
        case 4:
            if (top == NULL)
                printf(" Stack is empty");
            else
                printf(" Stack is not empty");
            break;
        case 5:
            top = display(top);
            break;
```

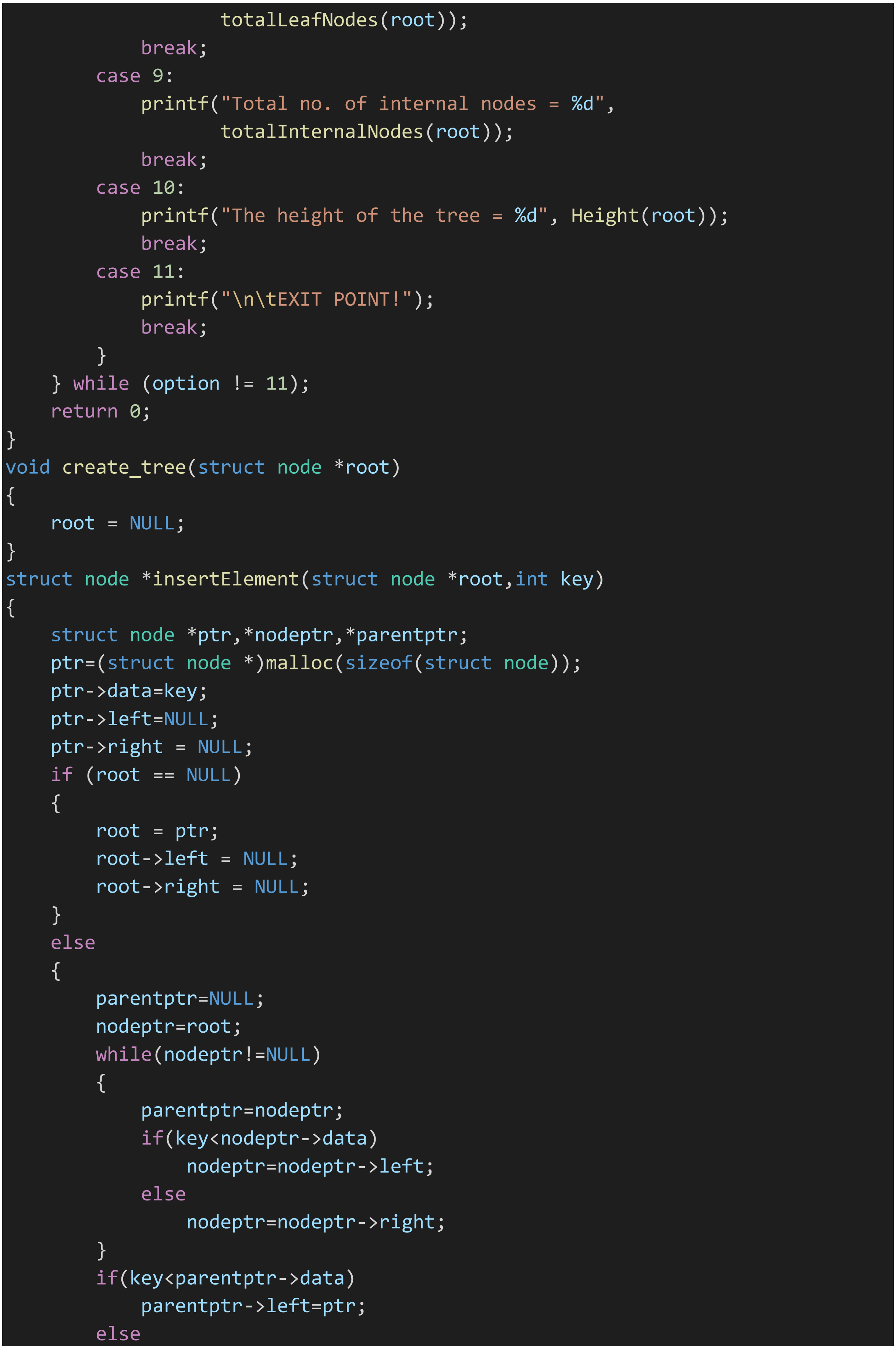


```
        }
    } while (option != 6);
    return 0;
}
struct stack *push(struct stack *top, int val)
{
    struct stack *ptr;
    ptr = (struct stack *)malloc(sizeof(struct stack));
    ptr->data = val;
    if (top == NULL)
    {
        ptr->next = NULL;
        top = ptr;
    }
    else
    {
        ptr->next = top;
        top = ptr;
    }
    return top;
}
struct stack *display(struct stack *top)
{
    struct stack *ptr;
    ptr = top;
    if (top == NULL)
        printf("\n STACK IS EMPTY");
    else
    {
        while (ptr != NULL)
        {
            printf("\n %d", ptr->data);
            ptr = ptr->next;
        }
    }
    return top;
}
struct stack *pop(struct stack *top)
{
    struct stack *ptr;
    ptr = top;
    if (top == NULL)
        printf("\n STACK UNDERFLOW");
    else
    {
        top = top->next;
        printf("\n The value being deleted is: %d", ptr->data);
        free(ptr);
    }
    return top;
}
int isEmpty(struct stack *top)
{
    if (top == NULL)

    {
        printf("Underflow");
        return 0;
    }
    else
        printf("SStack is not empty");
    return 1;
}
int peek(struct stack *top)
{
    if (top == NULL)
        return -1;
    else
        return top->data;
}
```

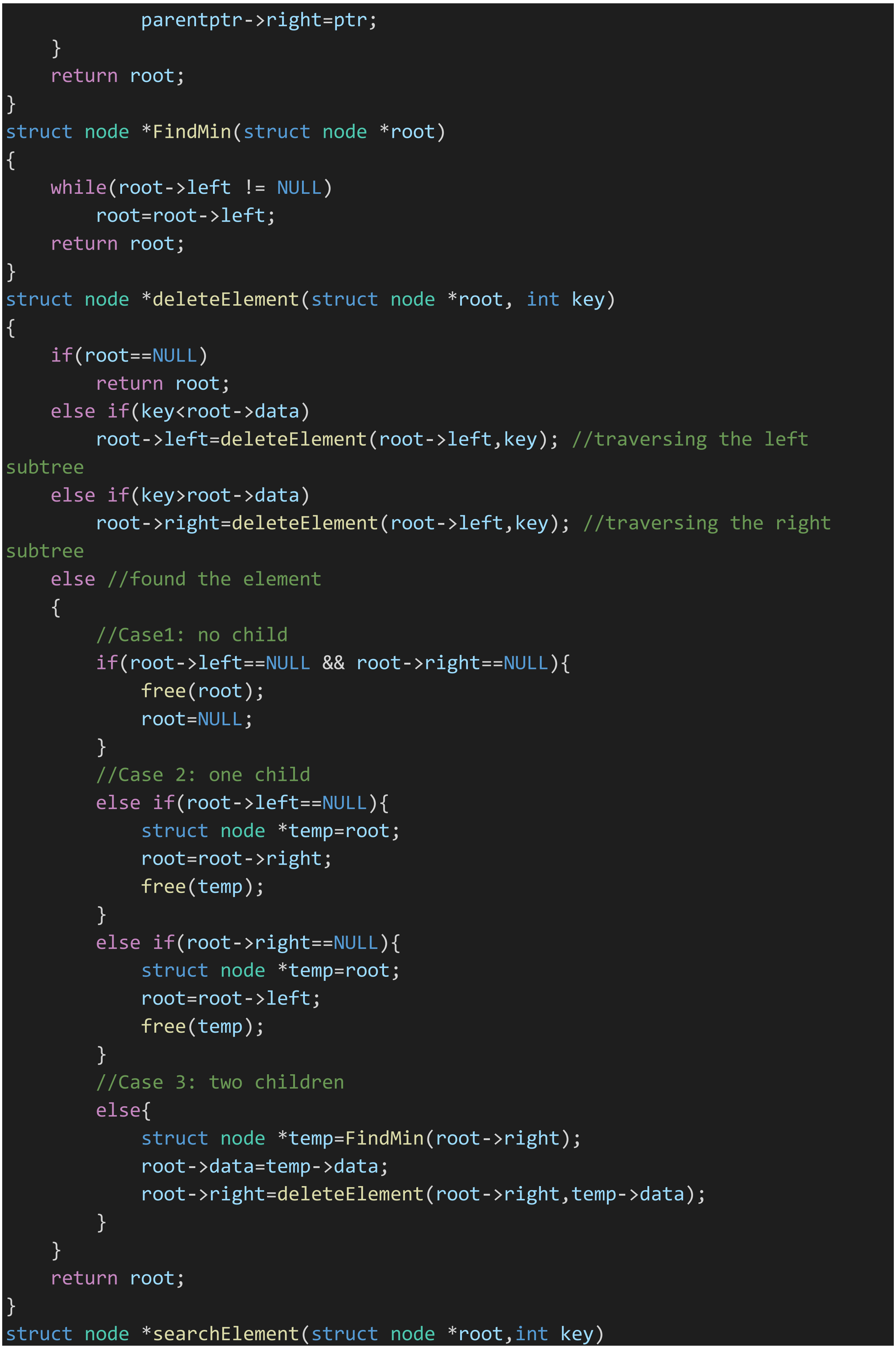
```
// Write a menu driven code to implement Binary Search Tree (1)
#include<stdio.h>
#include<stdlib.h>
#include<malloc.h>
struct node{
    int data;
    struct node *left;
    struct node *right;
};
struct node *root;
void create_tree(struct node *);
struct node *insertElement(struct node *, int);
struct node *FindMin(struct node *);
struct node *deleteElement(struct node *, int);
struct node *searchElement(struct node *, int);
void preorderTraversal(struct node *);
void inorderTraversal(struct node *);
void postorderTraversal(struct node *);
int totalNodes(struct node *);
int totalLeafNodes(struct node *);
int totalInternalNodes(struct node *);
int Height(struct node *);
int main()
{
    int option, key;
```



```c
    create_tree(root);
    printf("\n***List Of Operations***");
    printf("\n1. Insertion\n2. Deletion\n3. Searching\n4. Pre-order
Traversal\n5. In-order Traversal\n6. Postorder Traversal\n7. Total number of
nodes\n8. Total number of leaf nodes\n9. Total number of internal nodes\n10.
Find height of the tree\n11. Exit\n");
    do
    {
        printf("\nEnter your option : ");
        scanf("%d", &option);
        switch (option)
        {
        case 1:
            printf("Enter the value to be inserted: ");
            scanf("%d", &key);
            root = insertElement(root,key);
            break;
        case 2:
            printf("Enter the element to be deleted: ");
            scanf("%d", &key);
            root = deleteElement(root,key);
            break;
        case 3:
            printf("Enter the element to be searched: ");
            scanf("%d", &key);
            root = searchElement(root, key);
            if (root)
                printf("The value %d is found in the tree", key);
            else
                printf("The value %d not found", key);
            break;
        case 4:
            printf("The elements of the tree are : \n");
            preorderTraversal(root);
            break;
        case 5:
            printf("The elements of the tree are : \n");
            inorderTraversal(root);
            break;
        case 6:
            printf("The elements of the tree are : \n");
            postorderTraversal(root);
            break;
        case 7:
            printf("Total no. of nodes = %d", totalNodes(root));
            break;
        case 8:
            printf("Total no. of leaf nodes = %d",
```



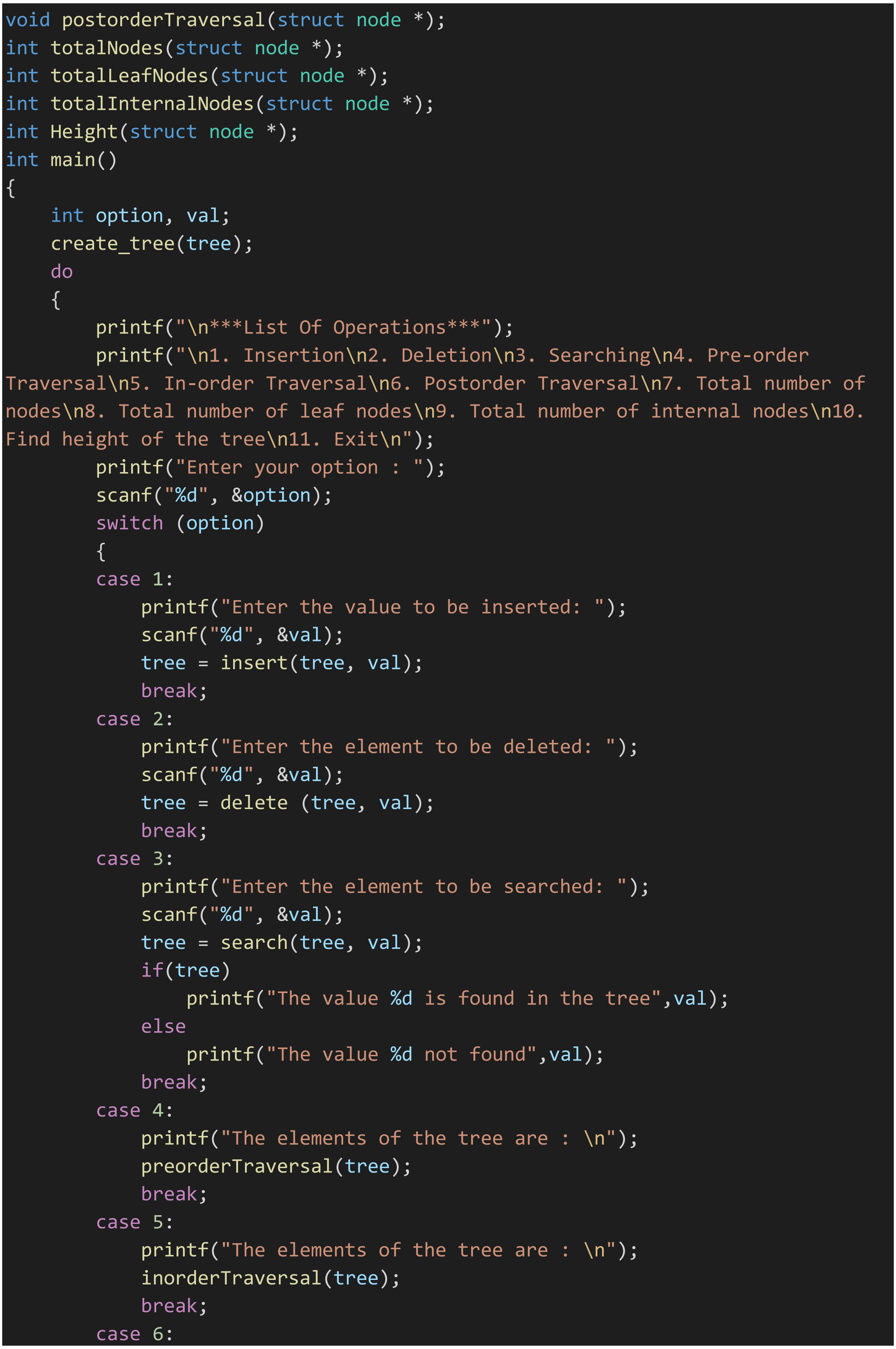
```
                   totalLeafNodes(root));
            break;
        case 9:
            printf("Total no. of internal nodes = %d",
                   totalInternalNodes(root));
            break;
        case 10:
            printf("The height of the tree = %d", Height(root));
            break;
        case 11:
            printf("\n\tEXIT POINT!");
            break;
        }
    } while (option != 11);
    return 0;
}
void create_tree(struct node *root)
{
    root = NULL;
}
struct node *insertElement(struct node *root,int key)
{
    struct node *ptr,*nodeptr,*parentptr;
    ptr=(struct node *)malloc(sizeof(struct node));
    ptr->data=key;
    ptr->left=NULL;
    ptr->right = NULL;
    if (root == NULL)
    {
        root = ptr;
        root->left = NULL;
        root->right = NULL;
    }
    else
    {
        parentptr=NULL;
        nodeptr=root;
        while(nodeptr!=NULL)
        {
            parentptr=nodeptr;
            if(key<nodeptr->data)
                nodeptr=nodeptr->left;
            else
                nodeptr=nodeptr->right;
        }
        if(key<parentptr->data)
            parentptr->left=ptr;
        else
```



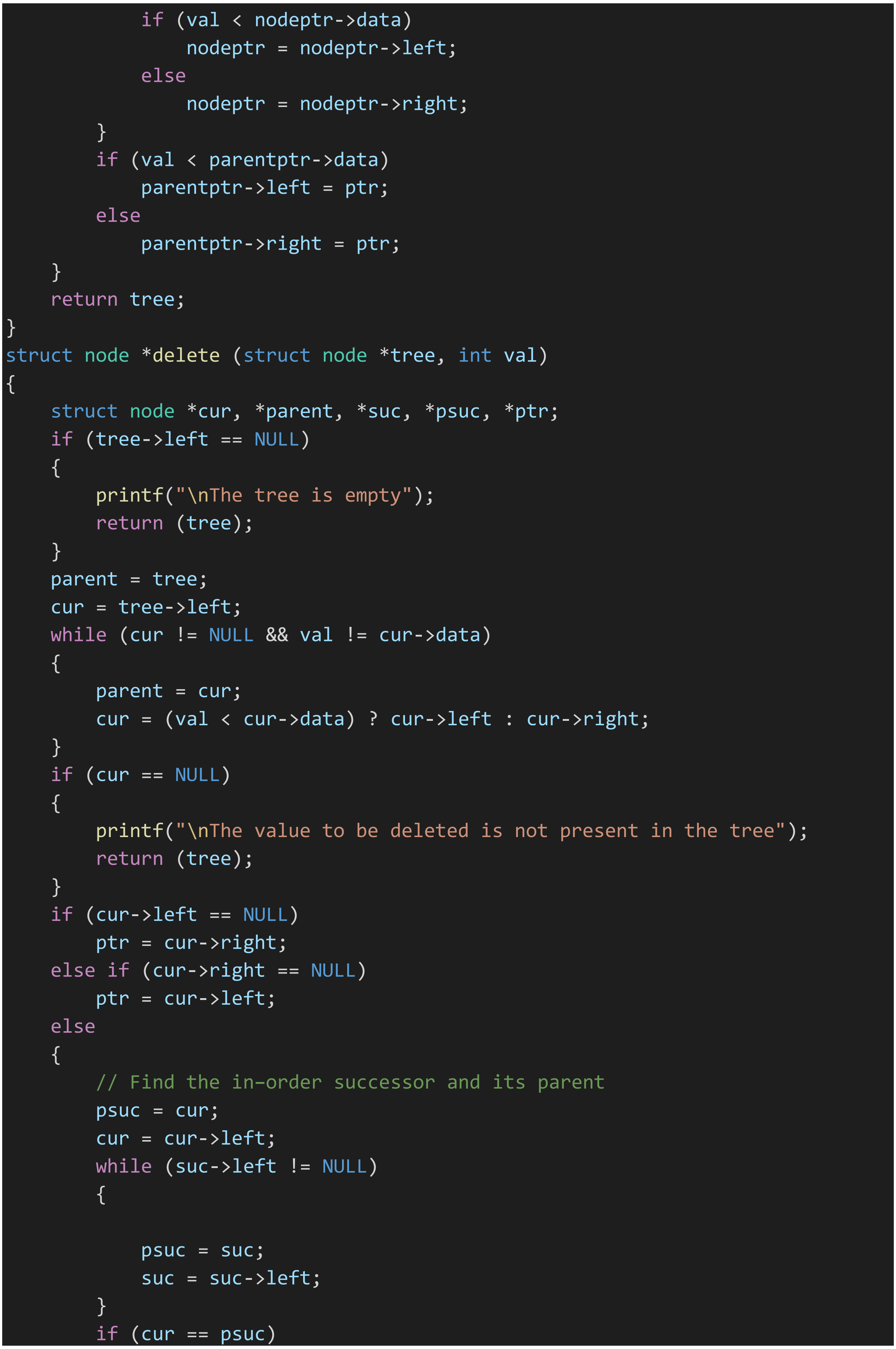
```
            parentptr->right=ptr;
    }
    return root;
}
struct node *FindMin(struct node *root)
{
    while(root->left != NULL)
        root=root->left;
    return root;
}
struct node *deleteElement(struct node *root, int key)
{
    if(root==NULL)
        return root;
    else if(key<root->data)
        root->left=deleteElement(root->left,key); //traversing the left 
subtree
    else if(key>root->data)
        root->right=deleteElement(root->left,key); //traversing the right 
subtree
    else //found the element
    {
        //Case1: no child
        if(root->left==NULL && root->right==NULL){
            free(root);
            root=NULL;
        }
        //Case 2: one child
        else if(root->left==NULL){
            struct node *temp=root;
            root=root->right;
            free(temp);
        }
        else if(root->right==NULL){
            struct node *temp=root;
            root=root->left;
            free(temp);
        }
        //Case 3: two children
        else{
            struct node *temp=FindMin(root->right);
            root->data=temp->data;
            root->right=deleteElement(root->right,temp->data);
        }
    }
    return root;
}
struct node *searchElement(struct node *root,int key)
```

```
{
    if(root==NULL)
    {
        printf("\nThe tree is empty");
    }
    else if(key>root->data)
        return searchElement(root->right,key);
    else if(key<root->data)
        return searchElement(root->left,key);
    else
        return root;
}
void preorderTraversal(struct node *root)
{
    if (root!= NULL)
    {
        printf("%d\t", root->data);
        preorderTraversal(root->left);
        preorderTraversal(root->right);
    }
}
void inorderTraversal(struct node *root)
{
    if (root!= NULL)
    {
        inorderTraversal(root->left);
        printf("%d\t", root->data);
        inorderTraversal(root->right);
    }
}
void postorderTraversal(struct node *root)
{
    if (root!= NULL)
    {
        postorderTraversal(root->left);
        postorderTraversal(root->right);
        printf("%d\t", root->data);
    }
}
int totalNodes(struct node *root)
{
    if (root == NULL)
        return 0;
    else
        return (totalNodes(root->left) + totalNodes(root->right) + 1);
}
int totalLeafNodes(struct node *root)
{
    if (root==NULL)
        return 0;
    else if ((root->left == NULL) && (root->right == NULL))
        return 1;
    else
        return (totalLeafNodes(root->left) + totalLeafNodes(root->right));
}
int totalInternalNodes(struct node *root)
{
    if ((root == NULL) || ((root->left == NULL) && (root->right == NULL)))
        return 0;
    else
        return (totalInternalNodes(root->left) + totalInternalNodes(root-
>right) + 1);
}
int Height(struct node *root)
{
    int leftheight, rightheight;
    if (root == NULL) return 0;
    else
    {
        leftheight = Height(root->left);
        rightheight = Height(root->right);
        if (leftheight > rightheight)
            return (leftheight + 1);
        else
            return (rightheight + 1);
    }
}
```

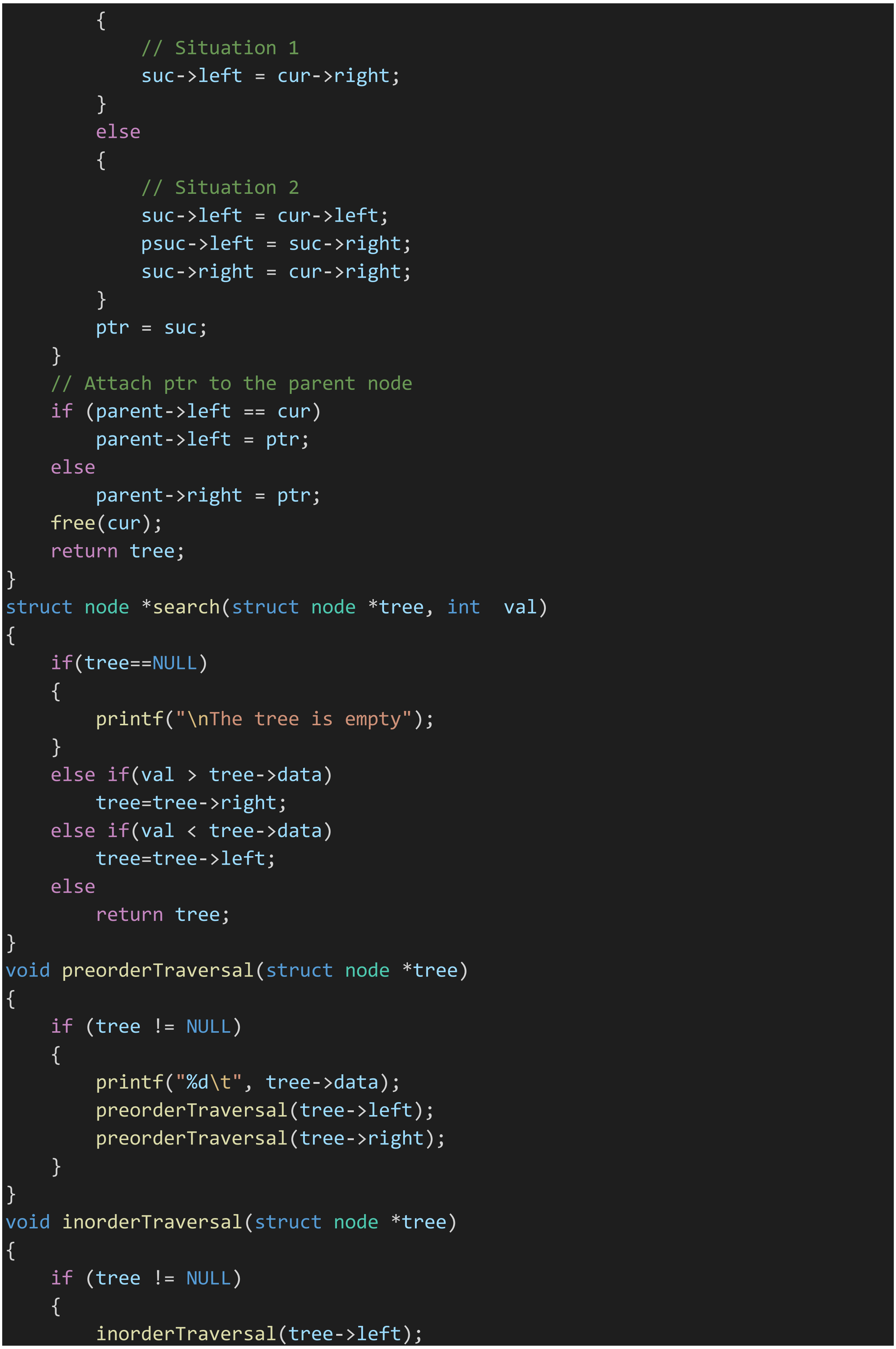
```
// Write a menu driven code to implement Binary Search Tree (2)
#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
struct node
{
    int data;
    struct node *left;
    struct node *right;
};
struct node *tree;
void create_tree(struct node *);
struct node *insert(struct node *, int);
struct node *delete (struct node *, int);
struct node *search(struct node *, int);
void preorderTraversal(struct node *);
void inorderTraversal(struct node *);
```



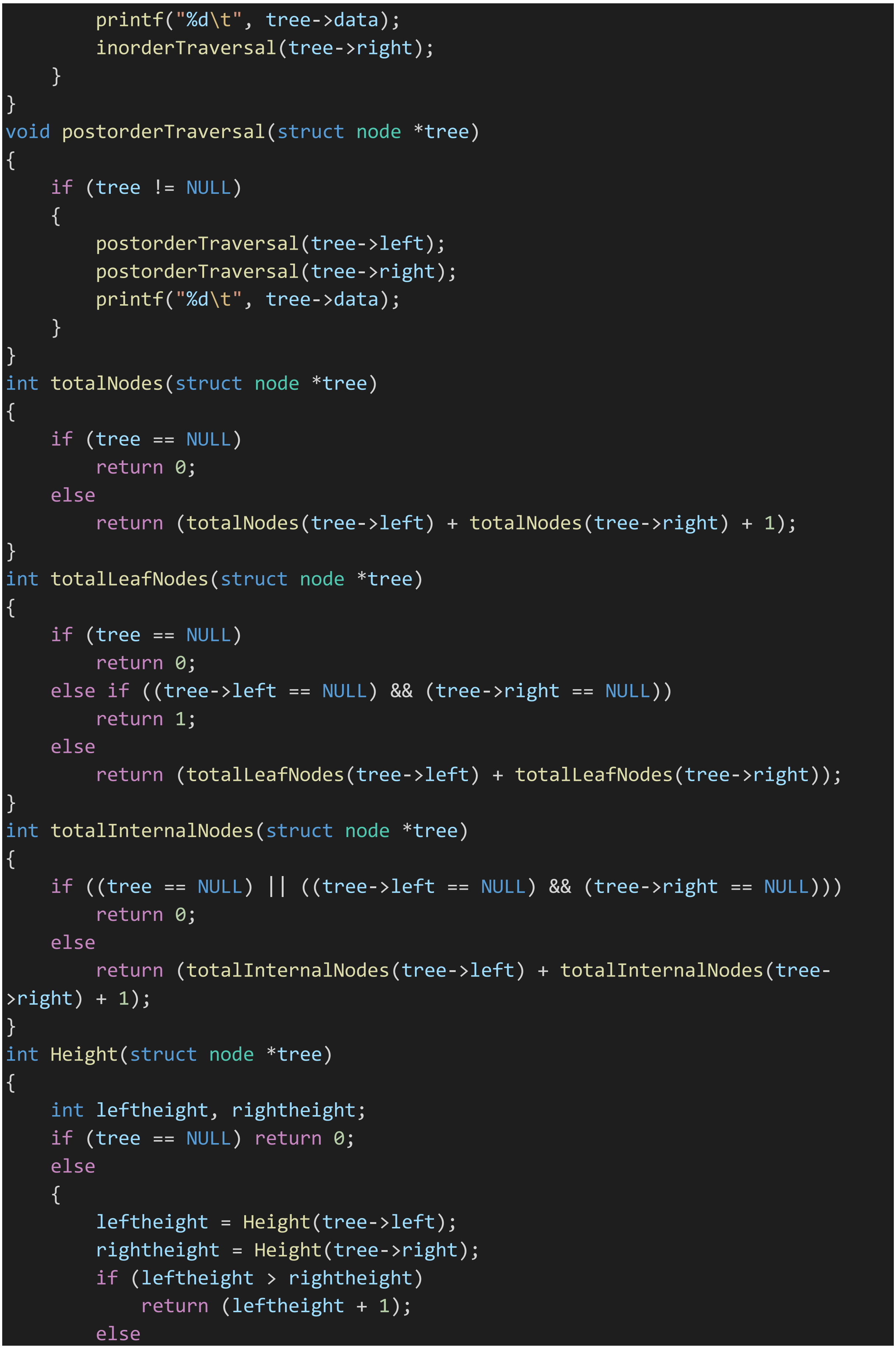
```
void postorderTraversal(struct node *);
int totalNodes(struct node *);
int totalLeafNodes(struct node *);
int totalInternalNodes(struct node *);
int Height(struct node *);
int main()
{
    int option, val;
    create_tree(tree);
    do
    {
        printf("\n***List Of Operations***");
        printf("\n1. Insertion\n2. Deletion\n3. Searching\n4. Pre-order
Traversal\n5. In-order Traversal\n6. Postorder Traversal\n7. Total number of
nodes\n8. Total number of leaf nodes\n9. Total number of internal nodes\n10.
Find height of the tree\n11. Exit\n");
        printf("Enter your option : ");
        scanf("%d", &option);
        switch (option)
        {
        case 1:
            printf("Enter the value to be inserted: ");
            scanf("%d", &val);
            tree = insert(tree, val);
            break;
        case 2:
            printf("Enter the element to be deleted: ");
            scanf("%d", &val);
            tree = delete (tree, val);
            break;
        case 3:
            printf("Enter the element to be searched: ");
            scanf("%d", &val);
            tree = search(tree, val);
            if(tree)
                printf("The value %d is found in the tree",val);
            else
                printf("The value %d not found",val);
            break;
        case 4:
            printf("The elements of the tree are : \n");
            preorderTraversal(tree);
            break;
        case 5:
            printf("The elements of the tree are : \n");
            inorderTraversal(tree);
            break;
        case 6:
            printf("The elements of the tree are : \n");
            postorderTraversal(tree);
            break;
        case 7:
            printf("Total no. of nodes = %d", totalNodes(tree));
            break;
        case 8:
            printf("Total no. of leaf nodes = %d",
                   totalLeafNodes(tree));
            break;
        case 9:
            printf("Total no. of internal nodes = %d",
                   totalInternalNodes(tree));
            break;
        case 10:
            printf("The height of the tree = %d", Height(tree));
            break;
        case 11:
            printf("\n\tEXIT POINT!");
            break;
        }
    } while (option != 11);
    return 0;
}
void create_tree(struct node *tree)
{
    tree = NULL;
}
struct node *insert(struct node *tree, int val)
{
    struct node *ptr, *nodeptr, *parentptr;
    ptr = (struct node *)malloc(sizeof(struct node));
    ptr->data = val;
    ptr->left = NULL;
    ptr->right = NULL;
    if (tree == NULL)
    {
        tree = ptr;
        tree->left = NULL;
        tree->right = NULL;
    }
    else
    {
        parentptr = NULL;
        nodeptr = tree;
        while (nodeptr != NULL)
        {
            parentptr = nodeptr;
```



```
            if (val < nodeptr->data)
                nodeptr = nodeptr->left;
            else
                nodeptr = nodeptr->right;
        }
        if (val < parentptr->data)
            parentptr->left = ptr;
        else
            parentptr->right = ptr;
    }
    return tree;
}
struct node *delete (struct node *tree, int val)
{
    struct node *cur, *parent, *suc, *psuc, *ptr;
    if (tree->left == NULL)
    {
        printf("\nThe tree is empty");
        return (tree);
    }
    parent = tree;
    cur = tree->left;
    while (cur != NULL && val != cur->data)
    {
        parent = cur;
        cur = (val < cur->data) ? cur->left : cur->right;
    }
    if (cur == NULL)
    {
        printf("\nThe value to be deleted is not present in the tree");
        return (tree);
    }
    if (cur->left == NULL)
        ptr = cur->right;
    else if (cur->right == NULL)
        ptr = cur->left;
    else
    {
        // Find the in-order successor and its parent
        psuc = cur;
        cur = cur->left;
        while (suc->left != NULL)
        {

            psuc = suc;
            suc = suc->left;
        }
        if (cur == psuc)
```



```
        {
            // Situation 1
            suc->left = cur->right;
        }
        else
        {
            // Situation 2
            suc->left = cur->left;
            psuc->left = suc->right;
            suc->right = cur->right;
        }
        ptr = suc;
    }
    // Attach ptr to the parent node
    if (parent->left == cur)
        parent->left = ptr;
    else
        parent->right = ptr;
    free(cur);
    return tree;
}
struct node *search(struct node *tree, int  val)
{
    if(tree==NULL)
    {
        printf("\nThe tree is empty");
    }
    else if(val > tree->data)
        tree=tree->right;
    else if(val < tree->data)
        tree=tree->left;
    else
        return tree;
}
void preorderTraversal(struct node *tree)
{
    if (tree != NULL)
    {
        printf("%d\t", tree->data);
        preorderTraversal(tree->left);
        preorderTraversal(tree->right);
    }
}
void inorderTraversal(struct node *tree)
{
    if (tree != NULL)
    {
        inorderTraversal(tree->left);
```

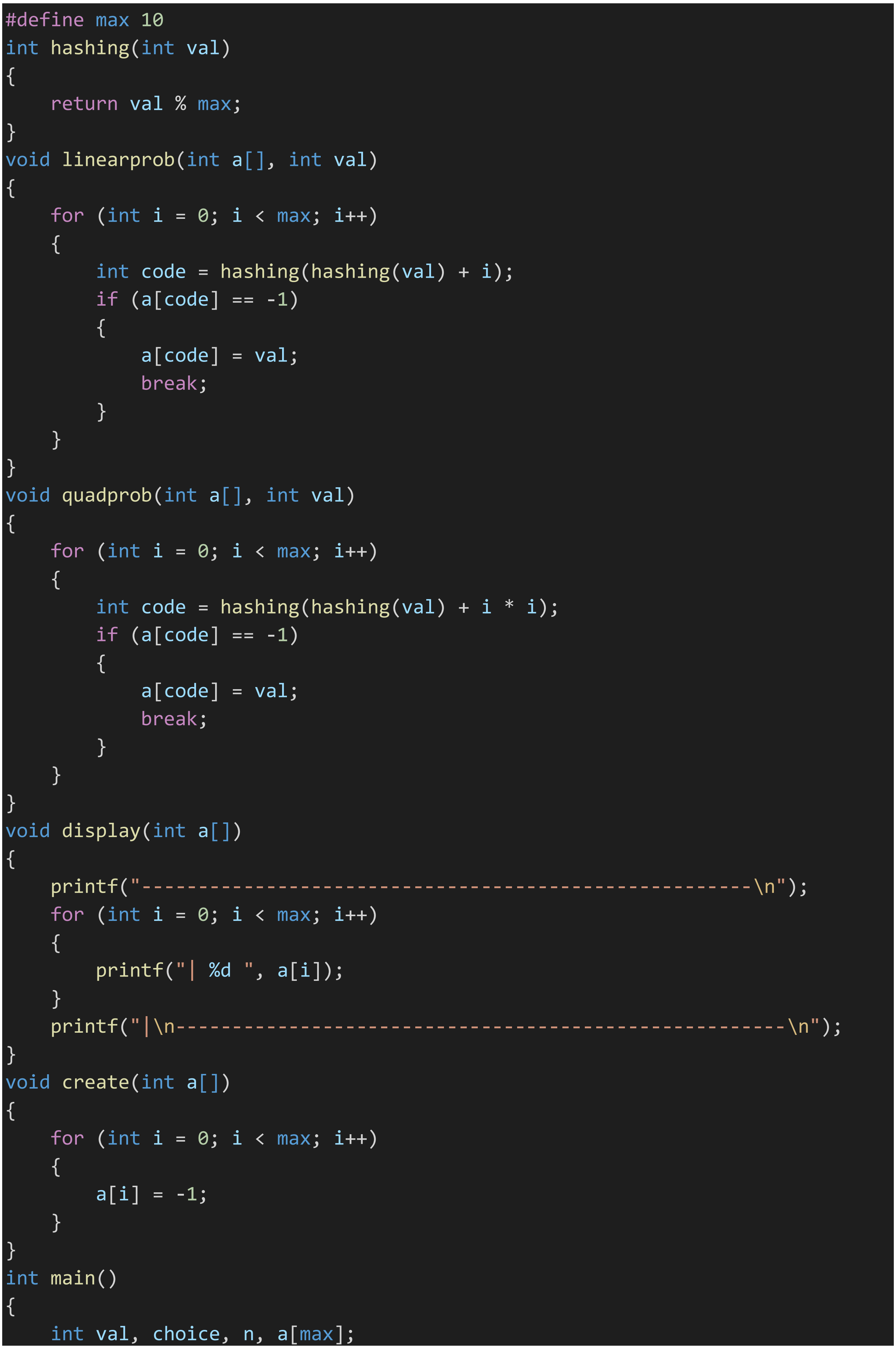


```
        printf("%d\t", tree->data);
        inorderTraversal(tree->right);
    }
}
void postorderTraversal(struct node *tree)
{
    if (tree != NULL)
    {
        postorderTraversal(tree->left);
        postorderTraversal(tree->right);
        printf("%d\t", tree->data);
    }
}
int totalNodes(struct node *tree)
{
    if (tree == NULL)
        return 0;
    else
        return (totalNodes(tree->left) + totalNodes(tree->right) + 1);
}
int totalLeafNodes(struct node *tree)
{
    if (tree == NULL)
        return 0;
    else if ((tree->left == NULL) && (tree->right == NULL))
        return 1;
    else
        return (totalLeafNodes(tree->left) + totalLeafNodes(tree->right));
}
int totalInternalNodes(struct node *tree)
{
    if ((tree == NULL) || ((tree->left == NULL) && (tree->right == NULL)))
        return 0;
    else
        return (totalInternalNodes(tree->left) + totalInternalNodes(tree-
>right) + 1);
}
int Height(struct node *tree)
{
    int leftheight, rightheight;
    if (tree == NULL) return 0;
    else
    {
        leftheight = Height(tree->left);
        rightheight = Height(tree->right);
        if (leftheight > rightheight)
            return (leftheight + 1);
        else
```

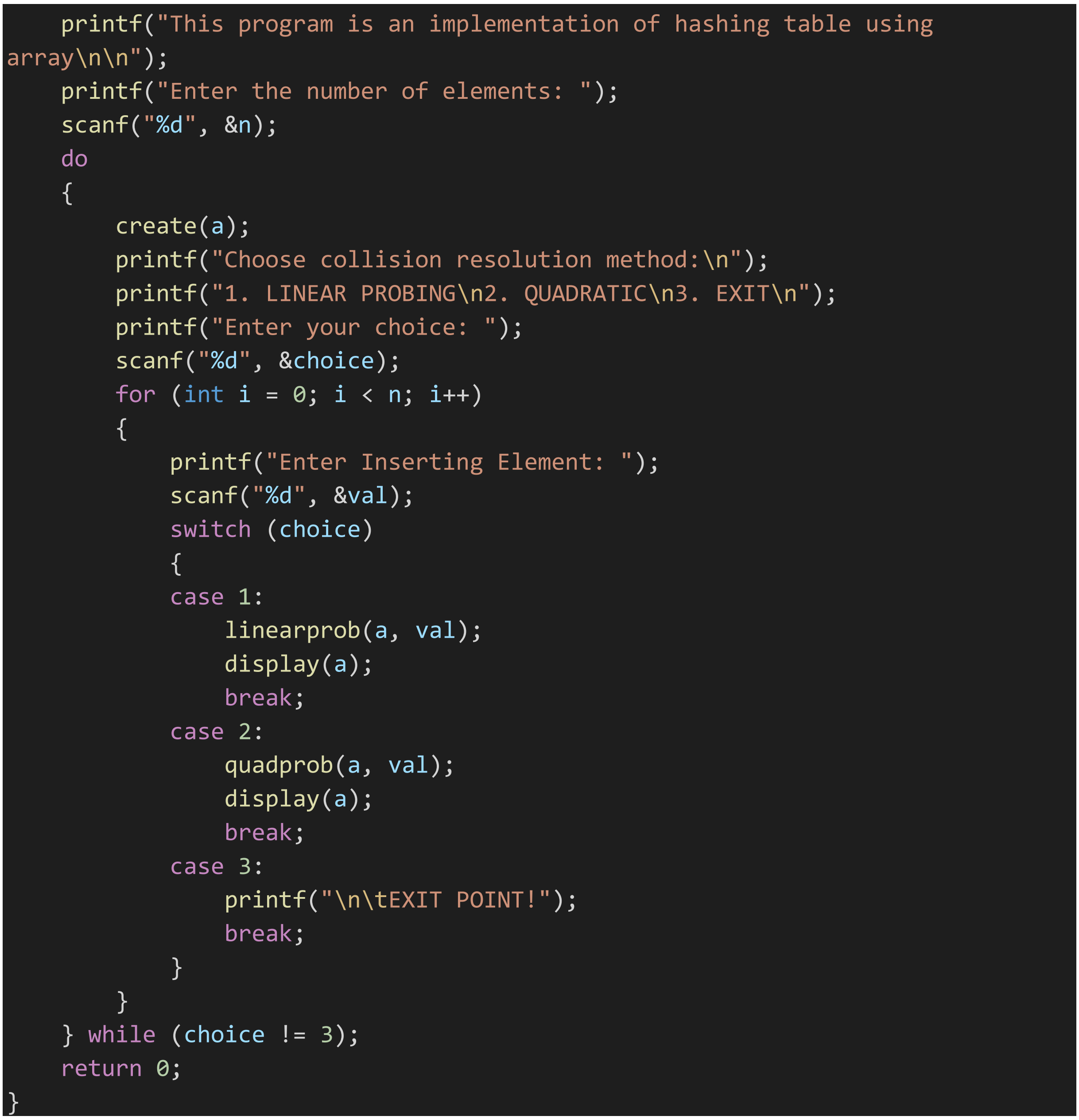
```
            return (rightheight + 1);
    }
}
```

```
//WAP to implement BFS and DFS
#include <stdio.h>
#include <conio.h>
int adj[30][30], n;
void BFS(int front, int rear, int vis[], int queue[], int start)
{
    int i;
    for (i = 0; i < n; i++)
    {
        if (adj[start][i] != 0 && vis[i] != 1)
        {
            rear = rear + 1;
            queue[rear] = i;
            vis[i] = 1;
            printf("%d ", i);
        }
    }
    front = front + 1;
    if (front <= rear)
        BFS(front, rear, vis, queue, queue[front]);
}
void DFS(int vis[], int start)
{
    int j;
    for (j = 0; j < n; j++)
    {
        if (vis[j] == 0 && adj[start][j] != 0)
        {
            vis[j] = 1;
            printf("%d ", j);
            DFS(vis, j);
        }
    }
}
int main()
{
    int choice, v;
    int front = -1, rear = -1;
    int queue[10], vis1[10], vis2[10] = {0};
    printf("Enter no. of vertices of adjaceny matrix: ");
    scanf("%d", &n);
    printf("Enter the Adjacency Matrix:\n");
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
            scanf("%d", &adj[i][j]);
    }
    for (int i = 0; i < n; i++)
    {
        vis1[i] = 0;
    }
    printf("Press 1.BFS\n");
    printf("Press 2.DFS\n");
    printf("Press 3.Exit\n");
    do
    {
        printf("\nEnter your choice: ");
        scanf("%d", &choice);
        switch (choice)
        {
        case 1:
            printf("Enter the starting vertex: ");
            scanf("%d", &v);
            front = 0;
            rear = 0;
            queue[rear] = v;
            vis1[v] = 1;
            printf("BFS Traversal: ");
            printf("%d ", v);
            BFS(front, rear, vis1, queue, v);
            break;
        case 2:
            printf("Enter the starting vertex: ");
            scanf("%d", &v);
            printf("DFS Traversal: ");
            vis2[v] = 1;
            printf("%d ", v);
            DFS(vis2, v);
            break;
        case 3:
            printf("\n\tEXIT POINT!");
        }
    } while (choice != 3);
    return 0;
}
```

```
//WAP to implement Hashing Table using array
#include <stdio.h>
#include <stdlib.h>
```



```c
#define max 10
int hashing(int val)
{
    return val % max;
}
void linearprob(int a[], int val)
{
    for (int i = 0; i < max; i++)
    {
        int code = hashing(hashing(val) + i);
        if (a[code] == -1)
        {
            a[code] = val;
            break;
        }
    }
}
void quadprob(int a[], int val)
{
    for (int i = 0; i < max; i++)
    {
        int code = hashing(hashing(val) + i * i);
        if (a[code] == -1)
        {
            a[code] = val;
            break;
        }
    }
}
void display(int a[])
{
    printf("------------------------------------------------------\n");
    for (int i = 0; i < max; i++)
    {
        printf("| %d ", a[i]);
    }
    printf("|\n------------------------------------------------------\n");
}
void create(int a[])
{
    for (int i = 0; i < max; i++)
    {
        a[i] = -1;
    }
}
int main()
{
    int val, choice, n, a[max];
```



```
    printf("This program is an implementation of hashing table using
array\n\n");
    printf("Enter the number of elements: ");
    scanf("%d", &n);
    do
    {
        create(a);
        printf("Choose collision resolution method:\n");
        printf("1. LINEAR PROBING\n2. QUADRATIC\n3. EXIT\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        for (int i = 0; i < n; i++)
        {
            printf("Enter Inserting Element: ");
            scanf("%d", &val);
            switch (choice)
            {
            case 1:
                linearprob(a, val);
                display(a);
                break;
            case 2:
                quadprob(a, val);
                display(a);
                break;
            case 3:
                printf("\n\tEXIT POINT!");
                break;
            }
        }
    } while (choice != 3);
    return 0;
}
```