Data Structures & Algorithms

Programs

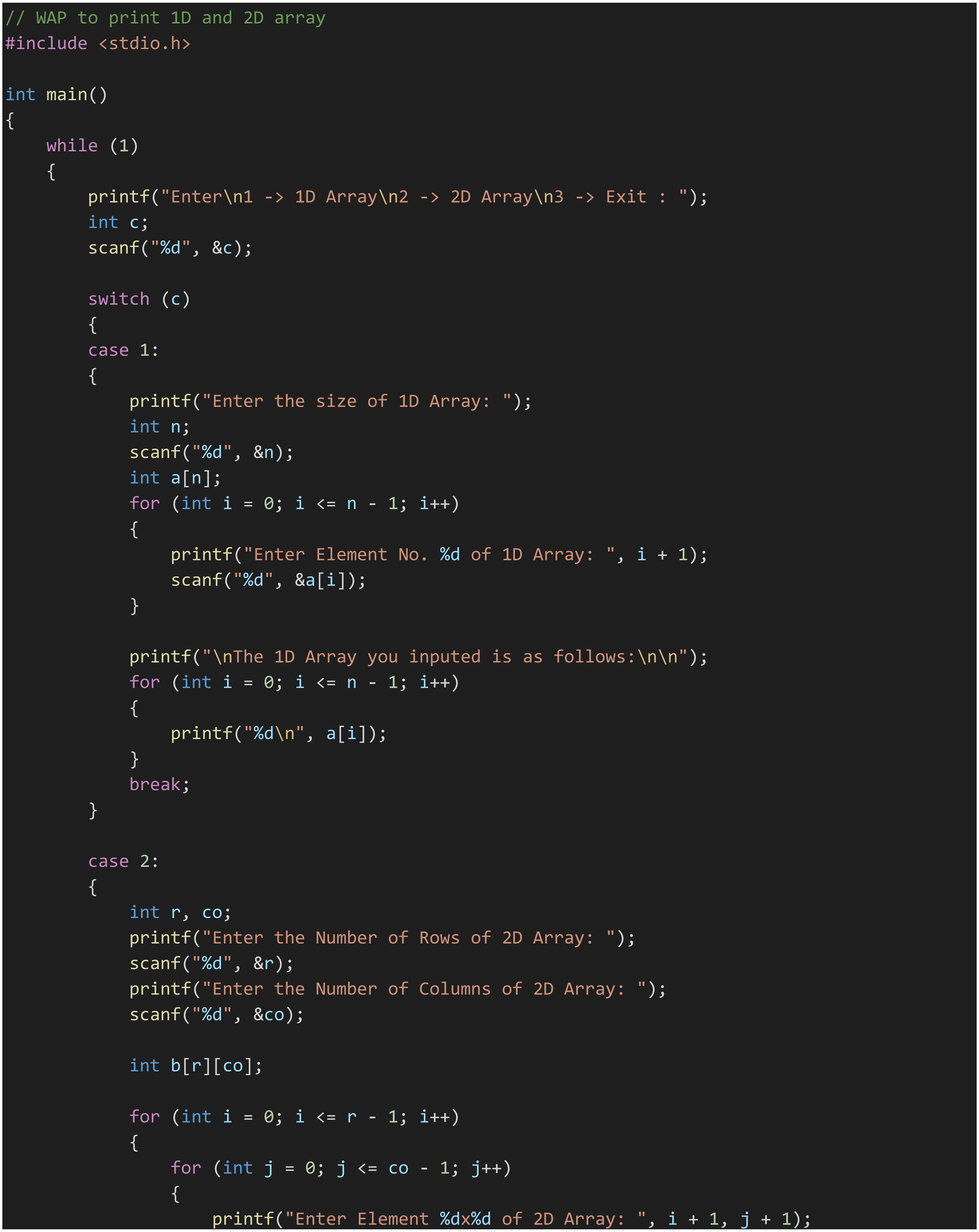
(IIIrd Semester)

2022-2023 Session

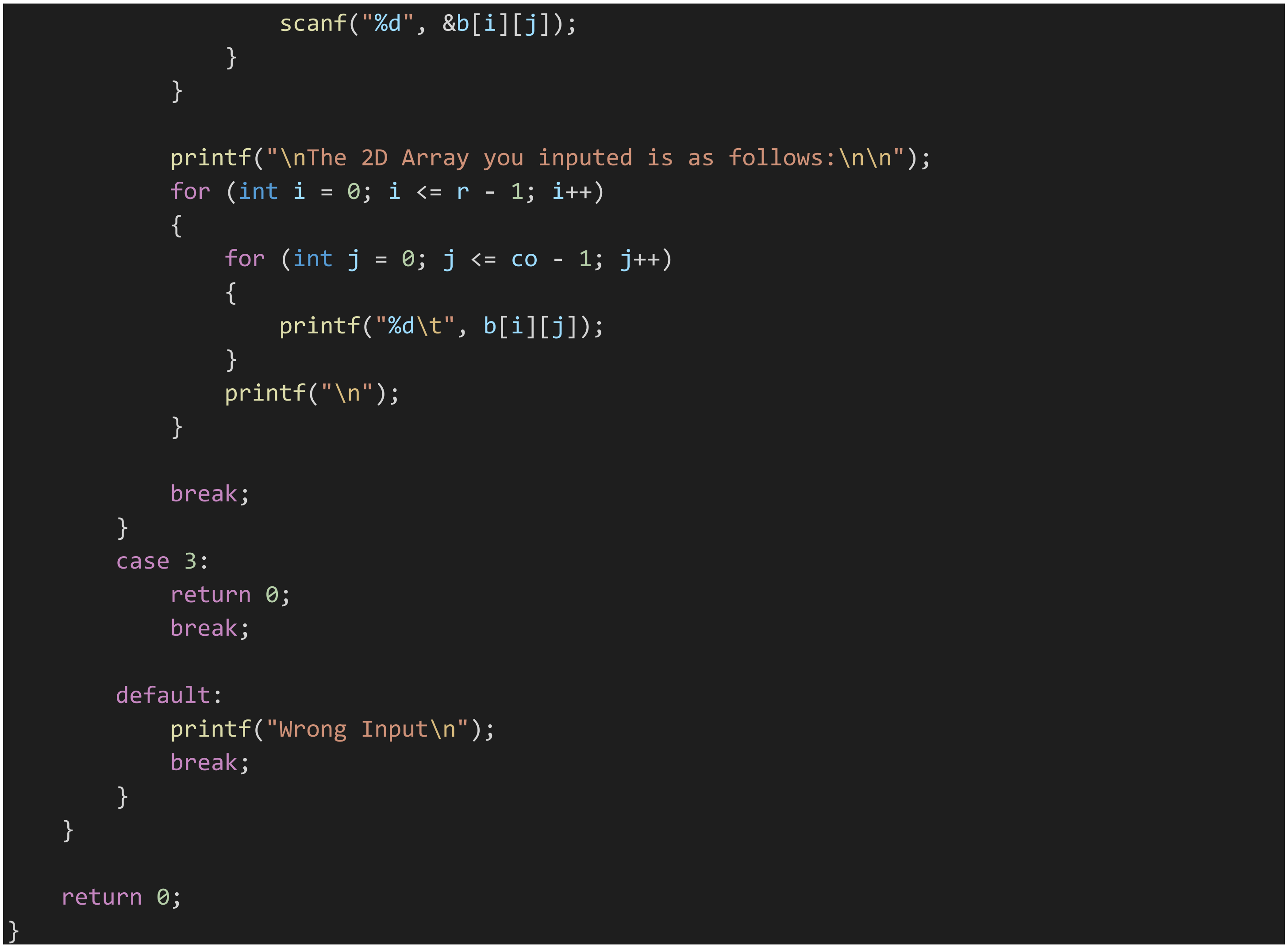
- **MD. IBRAHIM AKHTAR**
- **Computer Science Dept.**
- **Serial No.:- 07**
- **Roll No.:- 21BCS007**
- **Semester :- 3**

Sr. No.:	Date:	Program:
1	18/7/2022	1D & 2D array using Pointer
2	25/7/2022	Insert and Delete at an Index in the array
3	1/8/2022	Structure to store name, roll, marks & sem of student
4	22/8/2022	Implement Stack ADT
5	29/8/2022	Menu Driven Program to call Recursive Functions
6	12/9/2022	Implement Singly Linked List
7	26/9/2022	Implement Circular Queue
8	17/10/2022	Implement Complete Binary Tree
9	14/11/2022	Implement Complete Binary Tree using Linked List
Assignment Programs:		
1		Conversion of Number Systems
2		Implement Doubly Linked List
3		Implement Memory Efficient (XOR) Linked List

PROGRAM: 1

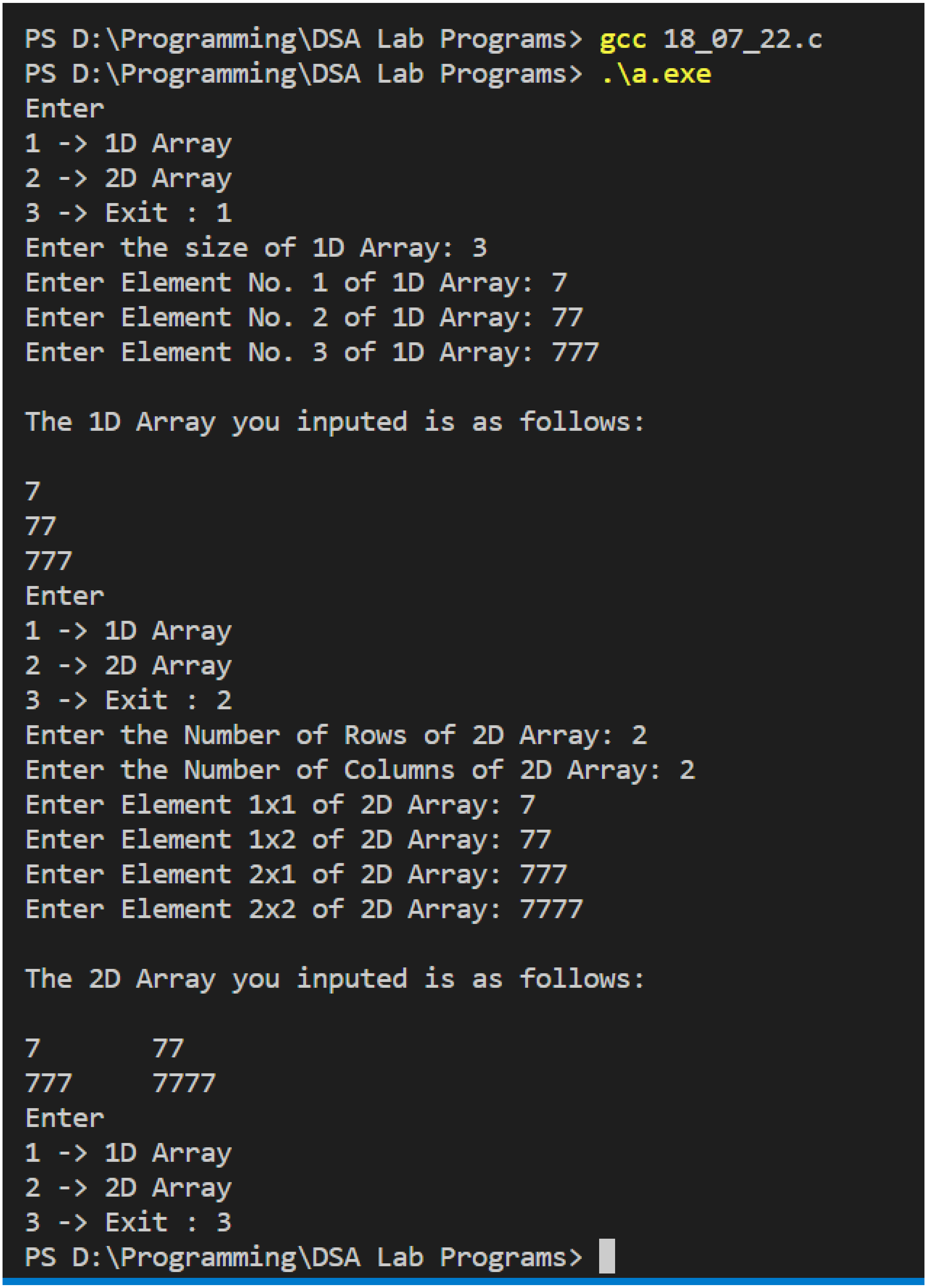


```c
// WAP to print 1D and 2D array
#include <stdio.h>

int main()
{
    while (1)
    {
        printf("Enter\n1 -> 1D Array\n2 -> 2D Array\n3 -> Exit : ");
        int c;
        scanf("%d", &c);

        switch (c)
        {
        case 1:
        {
            printf("Enter the size of 1D Array: ");
            int n;
            scanf("%d", &n);
            int a[n];
            for (int i = 0; i <= n - 1; i++)
            {
                printf("Enter Element No. %d of 1D Array: ", i + 1);
                scanf("%d", &a[i]);
            }

            printf("\nThe 1D Array you inputed is as follows:\n\n");
            for (int i = 0; i <= n - 1; i++)
            {
                printf("%d\n", a[i]);
            }
            break;
        }

        case 2:
        {
            int r, co;
            printf("Enter the Number of Rows of 2D Array: ");
            scanf("%d", &r);
            printf("Enter the Number of Columns of 2D Array: ");
            scanf("%d", &co);

            int b[r][co];

            for (int i = 0; i <= r - 1; i++)
            {
                for (int j = 0; j <= co - 1; j++)
                {
                    printf("Enter Element %dx%d of 2D Array: ", i + 1, j + 1);
```



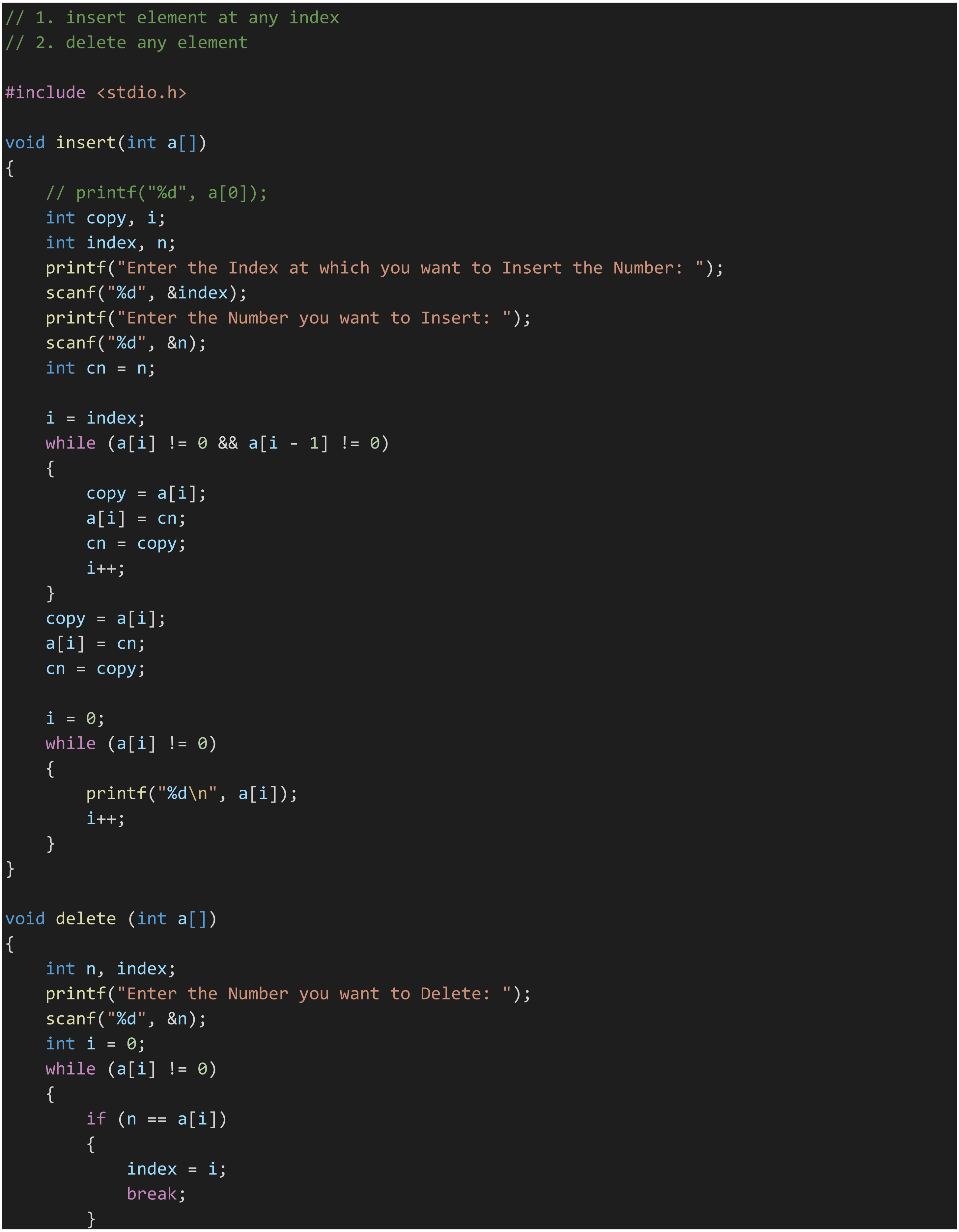
```
                    scanf("%d", &b[i][j]);
                }
            }

            printf("\nThe 2D Array you inputed is as follows:\n\n");
            for (int i = 0; i <= r - 1; i++)
            {
                for (int j = 0; j <= co - 1; j++)
                {
                    printf("%d\t", b[i][j]);
                }
                printf("\n");
            }

            break;
        }
        case 3:
            return 0;
            break;

        default:
            printf("Wrong Input\n");
            break;
        }
    }

    return 0;
}
```

Output: 1

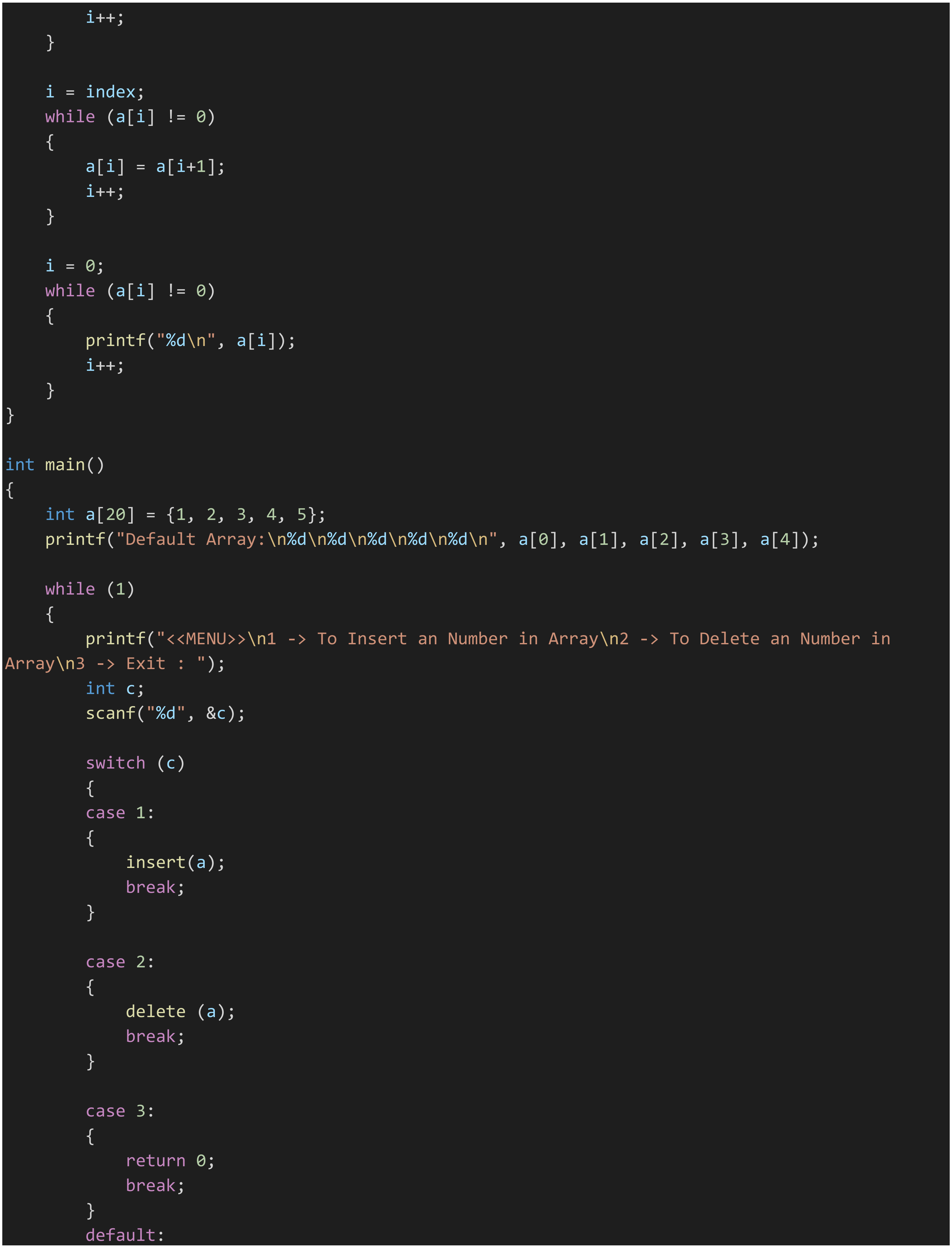


```
PS D:\Programming\DSA Lab Programs> gcc 18_07_22.c
PS D:\Programming\DSA Lab Programs> .\a.exe
Enter
1 -> 1D Array
2 -> 2D Array
3 -> Exit : 1
Enter the size of 1D Array: 3
Enter Element No. 1 of 1D Array: 7
Enter Element No. 2 of 1D Array: 77
Enter Element No. 3 of 1D Array: 777

The 1D Array you inputed is as follows:

7
77
777
Enter
1 -> 1D Array
2 -> 2D Array
3 -> Exit : 2
Enter the Number of Rows of 2D Array: 2
Enter the Number of Columns of 2D Array: 2
Enter Element 1x1 of 2D Array: 7
Enter Element 1x2 of 2D Array: 77
Enter Element 2x1 of 2D Array: 777
Enter Element 2x2 of 2D Array: 7777

The 2D Array you inputed is as follows:

7       77
777     7777
Enter
1 -> 1D Array
2 -> 2D Array
3 -> Exit : 3
PS D:\Programming\DSA Lab Programs>
```

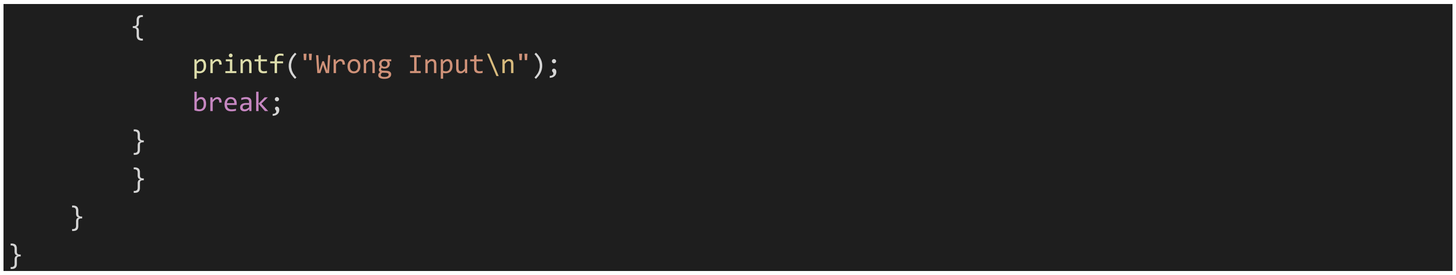
PROGRAM: 2



```c
// 1. insert element at any index
// 2. delete any element

#include <stdio.h>

void insert(int a[])
{
    // printf("%d", a[0]);
    int copy, i;
    int index, n;
    printf("Enter the Index at which you want to Insert the Number: ");
    scanf("%d", &index);
    printf("Enter the Number you want to Insert: ");
    scanf("%d", &n);
    int cn = n;

    i = index;
    while (a[i] != 0 && a[i - 1] != 0)
    {
        copy = a[i];
        a[i] = cn;
        cn = copy;
        i++;
    }
    copy = a[i];
    a[i] = cn;
    cn = copy;

    i = 0;
    while (a[i] != 0)
    {
        printf("%d\n", a[i]);
        i++;
    }
}

void delete (int a[])
{
    int n, index;
    printf("Enter the Number you want to Delete: ");
    scanf("%d", &n);
    int i = 0;
    while (a[i] != 0)
    {
        if (n == a[i])
        {
            index = i;
            break;
        }
```

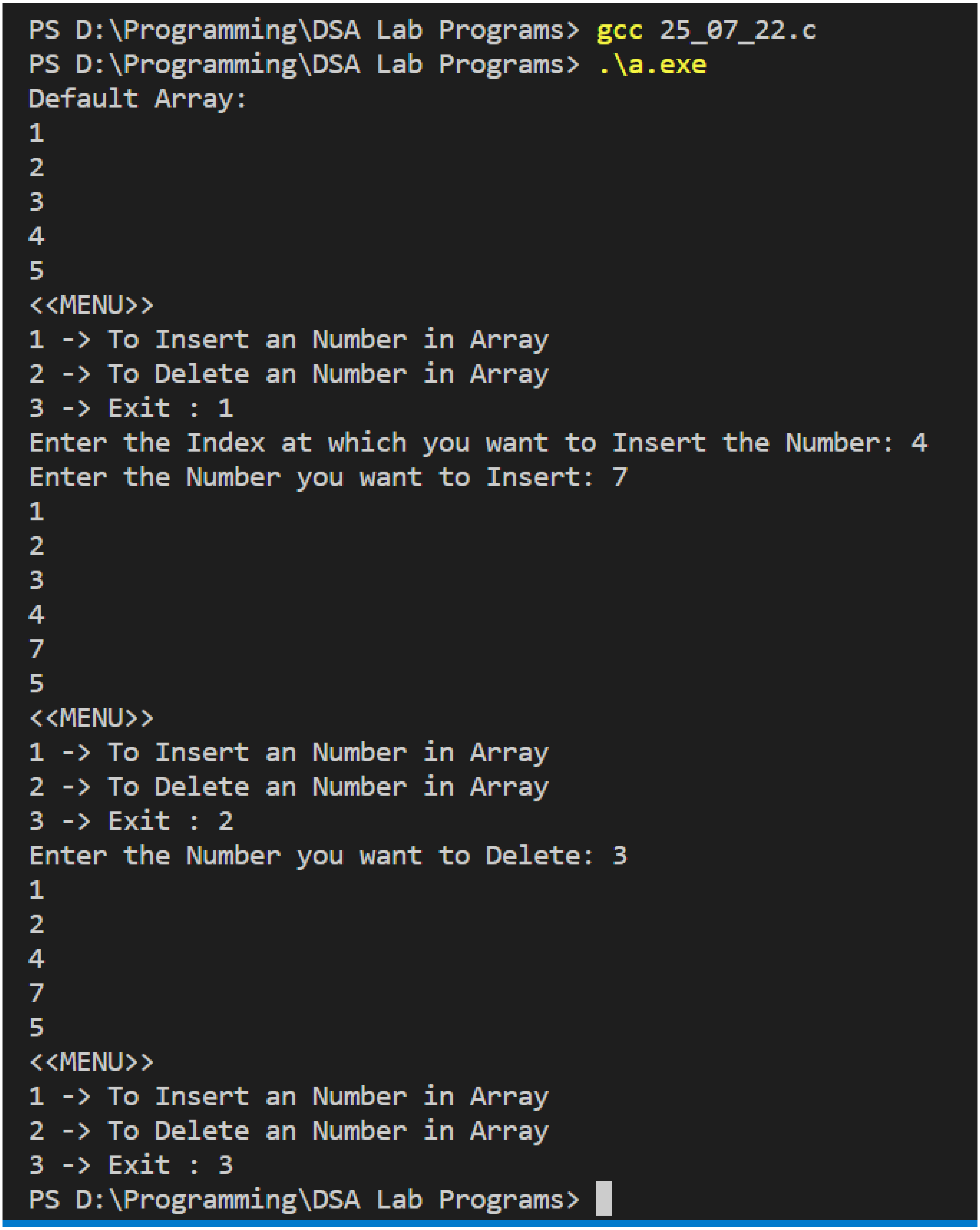


```
        i++;
    }

    i = index;
    while (a[i] != 0)
    {
        a[i] = a[i+1];
        i++;
    }

    i = 0;
    while (a[i] != 0)
    {
        printf("%d\n", a[i]);
        i++;
    }
}

int main()
{
    int a[20] = {1, 2, 3, 4, 5};
    printf("Default Array:\n%d\n%d\n%d\n%d\n%d\n", a[0], a[1], a[2], a[3], a[4]);

    while (1)
    {
        printf("<<MENU>>\n1 -> To Insert an Number in Array\n2 -> To Delete an Number in 
Array\n3 -> Exit : ");
        int c;
        scanf("%d", &c);

        switch (c)
        {
        case 1:
        {
            insert(a);
            break;
        }

        case 2:
        {
            delete (a);
            break;
        }

        case 3:
        {
            return 0;
            break;
        }
        default:
```



```
        {
            printf("Wrong Input\n");
            break;
        }
        }
    }
}
```

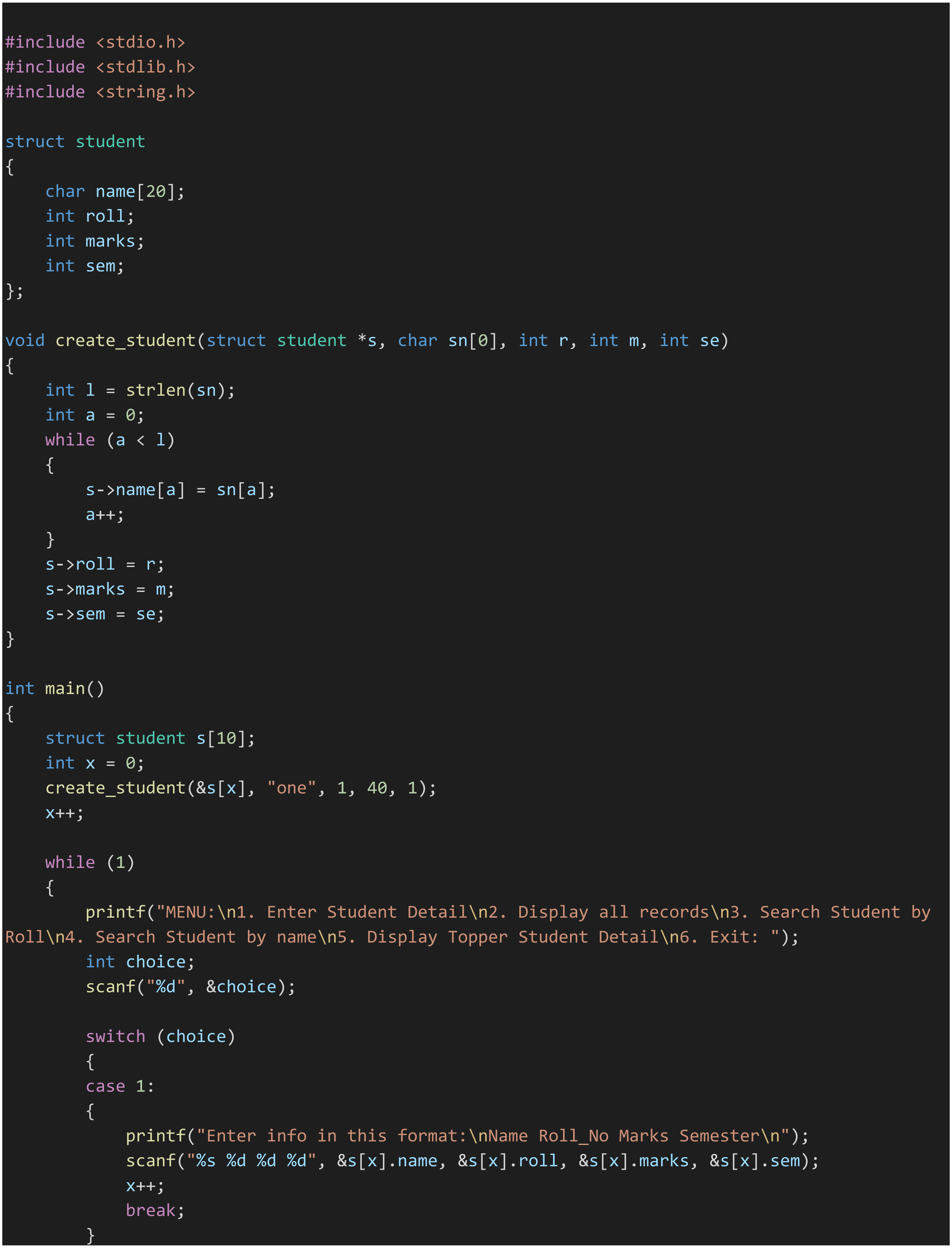
Output: 2



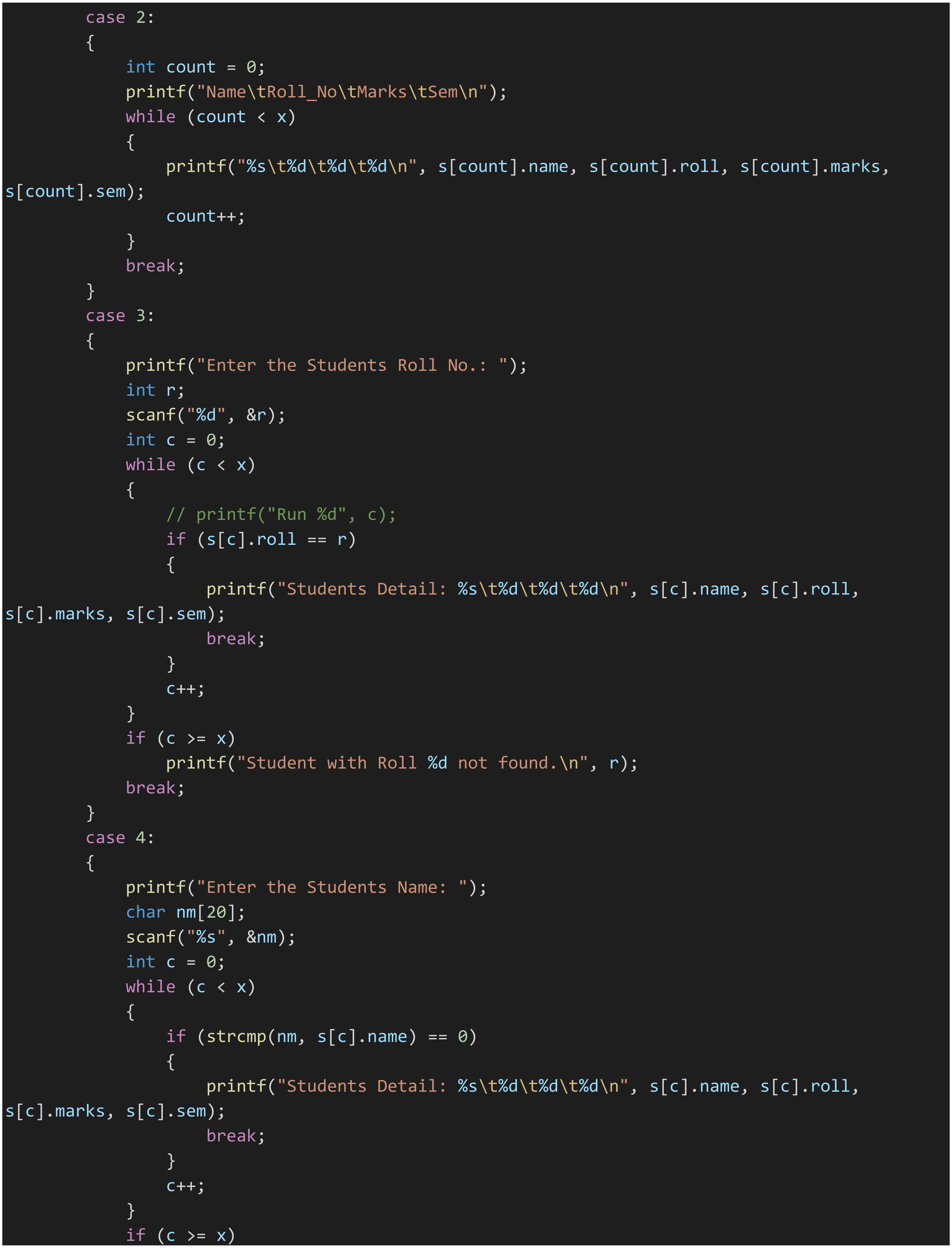
```
PS D:\Programming\DSA Lab Programs> gcc 25_07_22.c
PS D:\Programming\DSA Lab Programs> .\a.exe
Default Array:
1
2
3
4
5
<<MENU>>
1 -> To Insert an Number in Array
2 -> To Delete an Number in Array
3 -> Exit : 1
Enter the Index at which you want to Insert the Number: 4
Enter the Number you want to Insert: 7
1
2
3
4
7
5
<<MENU>>
1 -> To Insert an Number in Array
2 -> To Delete an Number in Array
3 -> Exit : 2
Enter the Number you want to Delete: 3
1
2
4
7
5
<<MENU>>
1 -> To Insert an Number in Array
2 -> To Delete an Number in Array
3 -> Exit : 3
PS D:\Programming\DSA Lab Programs>
```

PROGRAM: 3

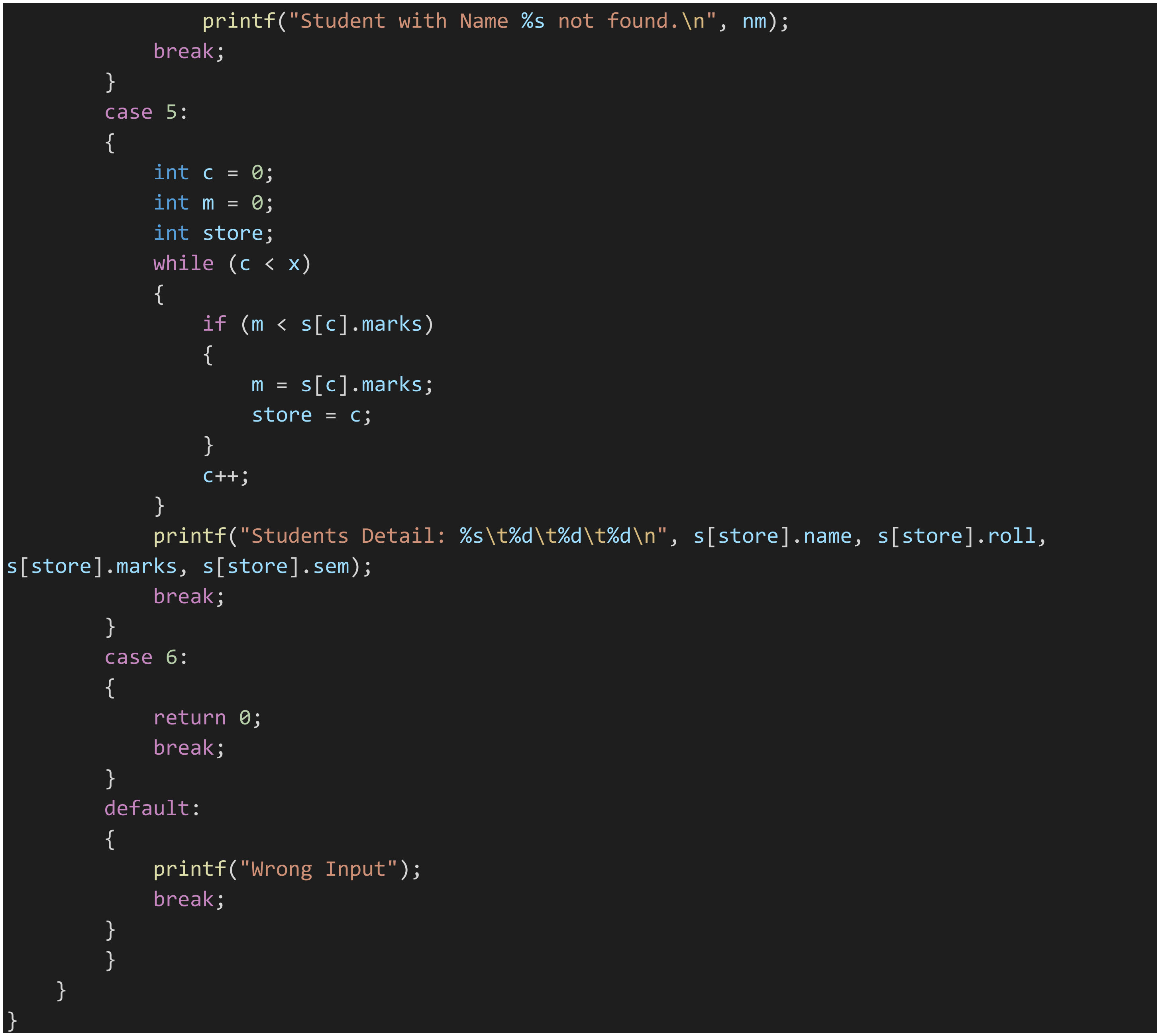
```
// Structures
```



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

struct student
{
    char name[20];
    int roll;
    int marks;
    int sem;
};

void create_student(struct student *s, char sn[0], int r, int m, int se)
{
    int l = strlen(sn);
    int a = 0;
    while (a < l)
    {
        s->name[a] = sn[a];
        a++;
    }
    s->roll = r;
    s->marks = m;
    s->sem = se;
}

int main()
{
    struct student s[10];
    int x = 0;
    create_student(&s[x], "one", 1, 40, 1);
    x++;

    while (1)
    {
        printf("MENU:\n1. Enter Student Detail\n2. Display all records\n3. Search Student by Roll\n4. Search Student by name\n5. Display Topper Student Detail\n6. Exit: ");
        int choice;
        scanf("%d", &choice);

        switch (choice)
        {
        case 1:
        {
            printf("Enter info in this format:\nName Roll_No Marks Semester\n");
            scanf("%s %d %d %d", &s[x].name, &s[x].roll, &s[x].marks, &s[x].sem);
            x++;
            break;
        }
```

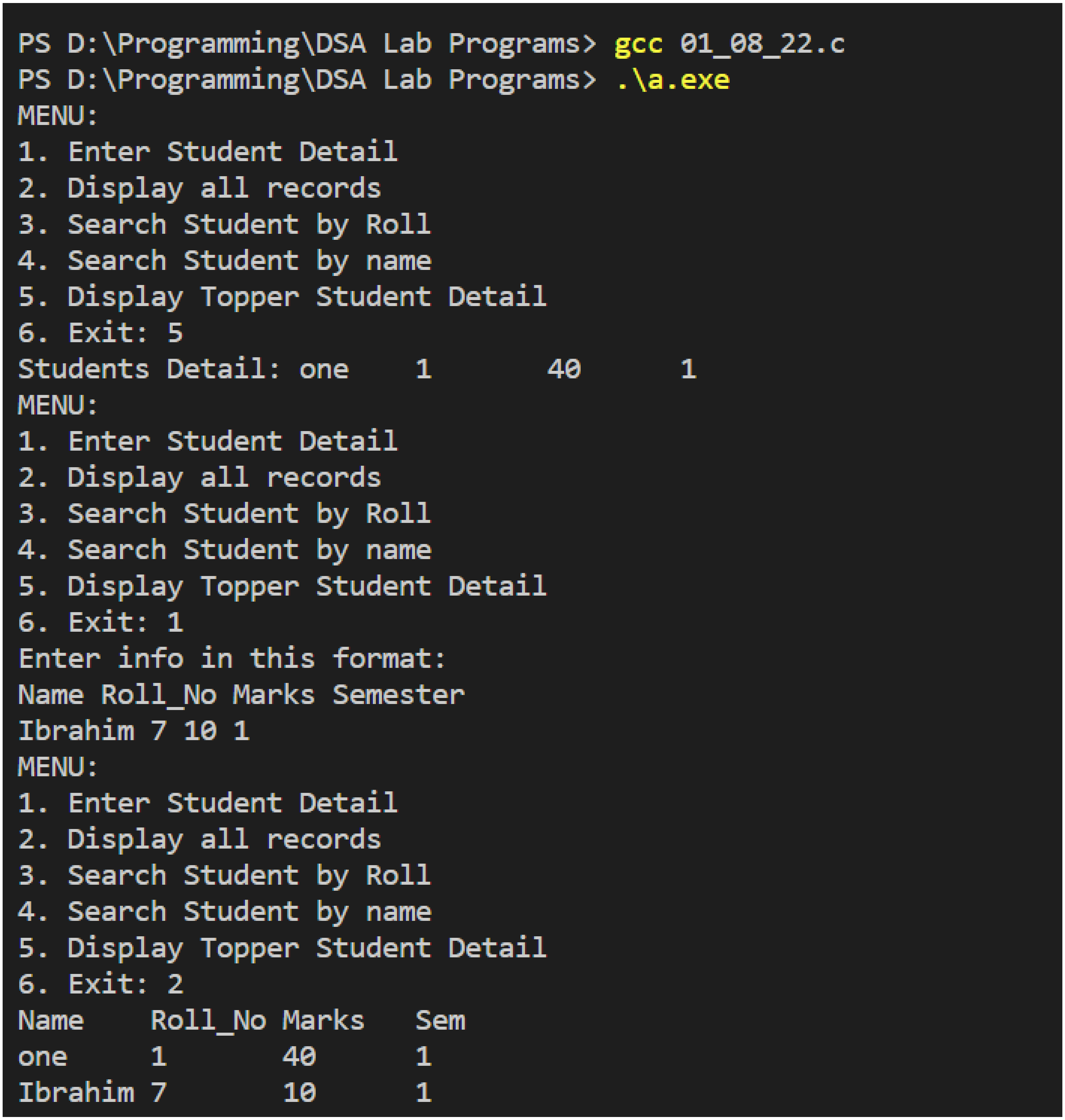


```
        case 2:
        {
            int count = 0;
            printf("Name\tRoll_No\tMarks\tSem\n");
            while (count < x)
            {
                printf("%s\t%d\t%d\t%d\n", s[count].name, s[count].roll, s[count].marks,
s[count].sem);
                count++;
            }
            break;
        }
        case 3:
        {
            printf("Enter the Students Roll No.: ");
            int r;
            scanf("%d", &r);
            int c = 0;
            while (c < x)
            {
                // printf("Run %d", c);
                if (s[c].roll == r)
                {
                    printf("Students Detail: %s\t%d\t%d\t%d\n", s[c].name, s[c].roll,
s[c].marks, s[c].sem);
                    break;
                }
                c++;
            }
            if (c >= x)
                printf("Student with Roll %d not found.\n", r);
            break;
        }
        case 4:
        {
            printf("Enter the Students Name: ");
            char nm[20];
            scanf("%s", &nm);
            int c = 0;
            while (c < x)
            {
                if (strcmp(nm, s[c].name) == 0)
                {
                    printf("Students Detail: %s\t%d\t%d\t%d\n", s[c].name, s[c].roll,
s[c].marks, s[c].sem);
                    break;
                }
                c++;
            }
            if (c >= x)
```

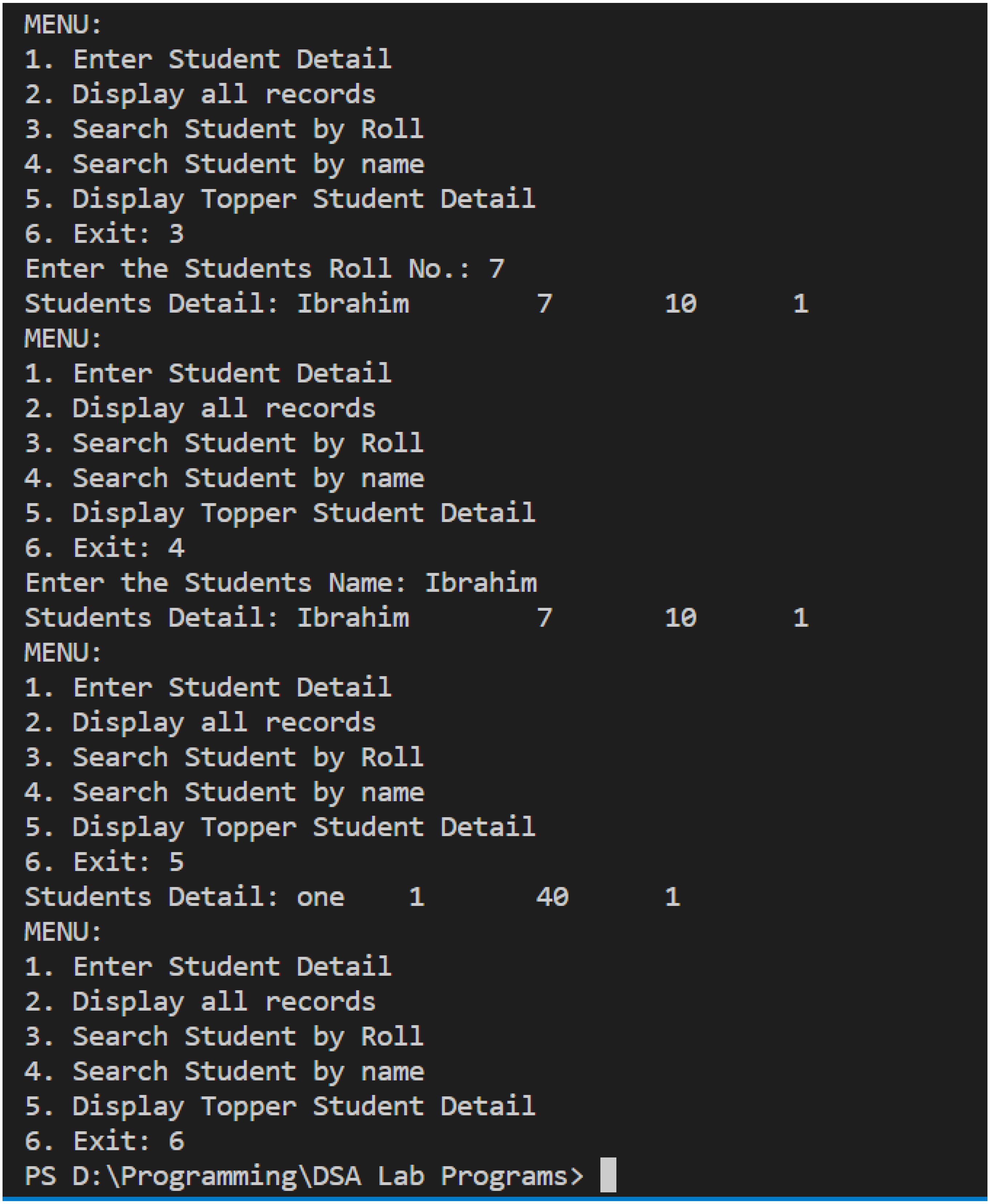


```
                printf("Student with Name %s not found.\n", nm);
            break;
        }
        case 5:
        {
            int c = 0;
            int m = 0;
            int store;
            while (c < x)
            {
                if (m < s[c].marks)
                {
                    m = s[c].marks;
                    store = c;
                }
                c++;
            }
            printf("Students Detail: %s\t%d\t%d\t%d\n", s[store].name, s[store].roll,
s[store].marks, s[store].sem);
            break;
        }
        case 6:
        {
            return 0;
            break;
        }
        default:
        {
            printf("Wrong Input");
            break;
        }
        }
    }
}
```

Output: 3



```
PS D:\Programming\DSA Lab Programs> gcc 01_08_22.c
PS D:\Programming\DSA Lab Programs> .\a.exe
MENU:
1. Enter Student Detail
2. Display all records
3. Search Student by Roll
4. Search Student by name
5. Display Topper Student Detail
6. Exit: 5
Students Detail: one    1       40      1
MENU:
1. Enter Student Detail
2. Display all records
3. Search Student by Roll
4. Search Student by name
5. Display Topper Student Detail
6. Exit: 1
Enter info in this format:
Name Roll_No Marks Semester
Ibrahim 7 10 1
MENU:
1. Enter Student Detail
2. Display all records
3. Search Student by Roll
4. Search Student by name
5. Display Topper Student Detail
6. Exit: 2
Name    Roll_No Marks   Sem
one     1       40      1
Ibrahim 7       10      1
```

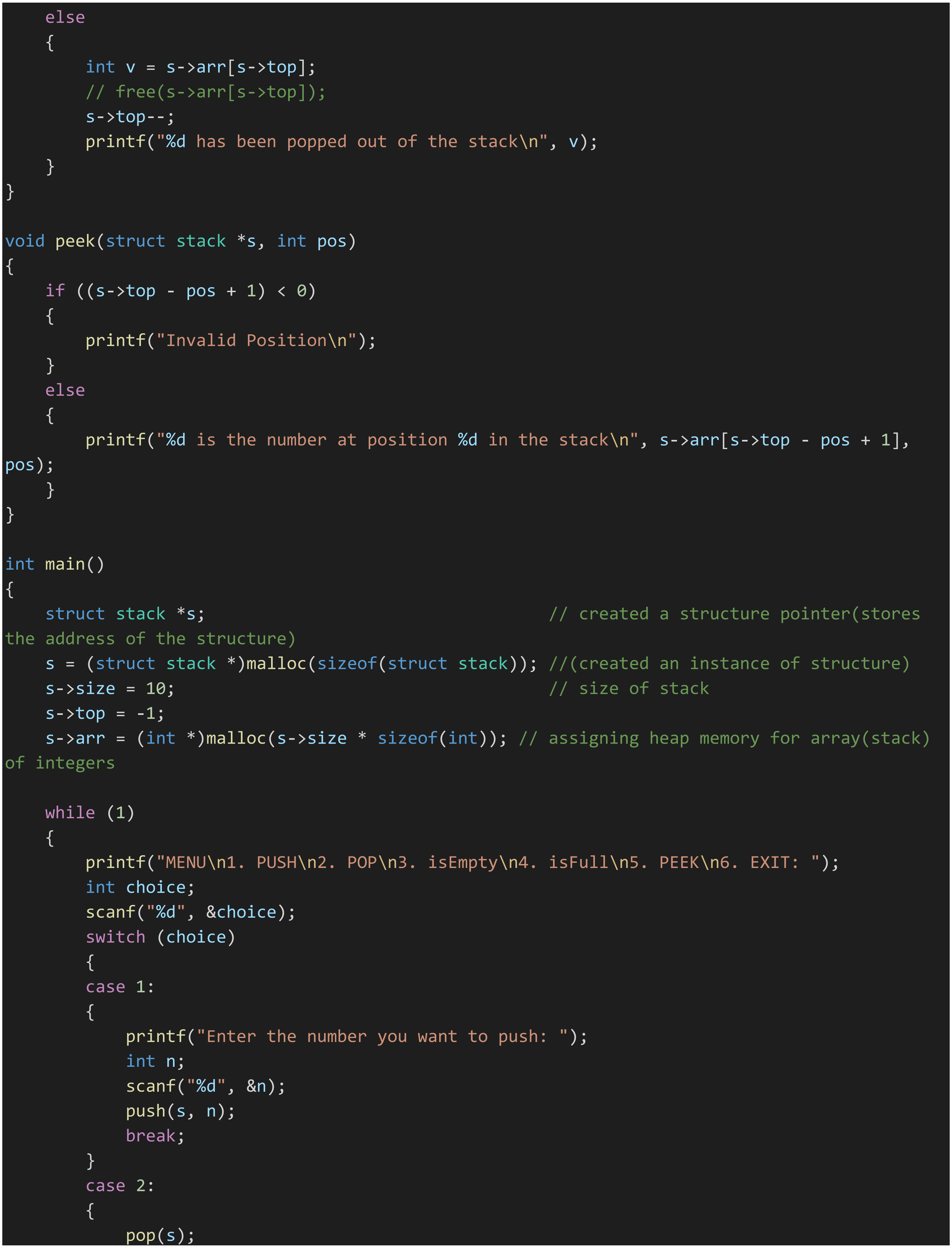


```
MENU:
1. Enter Student Detail
2. Display all records
3. Search Student by Roll
4. Search Student by name
5. Display Topper Student Detail
6. Exit: 3
Enter the Students Roll No.: 7
Students Detail: Ibrahim        7       10      1
MENU:
1. Enter Student Detail
2. Display all records
3. Search Student by Roll
4. Search Student by name
5. Display Topper Student Detail
6. Exit: 4
Enter the Students Name: Ibrahim
Students Detail: Ibrahim        7       10      1
MENU:
1. Enter Student Detail
2. Display all records
3. Search Student by Roll
4. Search Student by name
5. Display Topper Student Detail
6. Exit: 5
Students Detail: one    1       40      1
MENU:
1. Enter Student Detail
2. Display all records
3. Search Student by Roll
4. Search Student by name
5. Display Topper Student Detail
6. Exit: 6
PS D:\Programming\DSA Lab Programs>
```

PROGRAM: 4

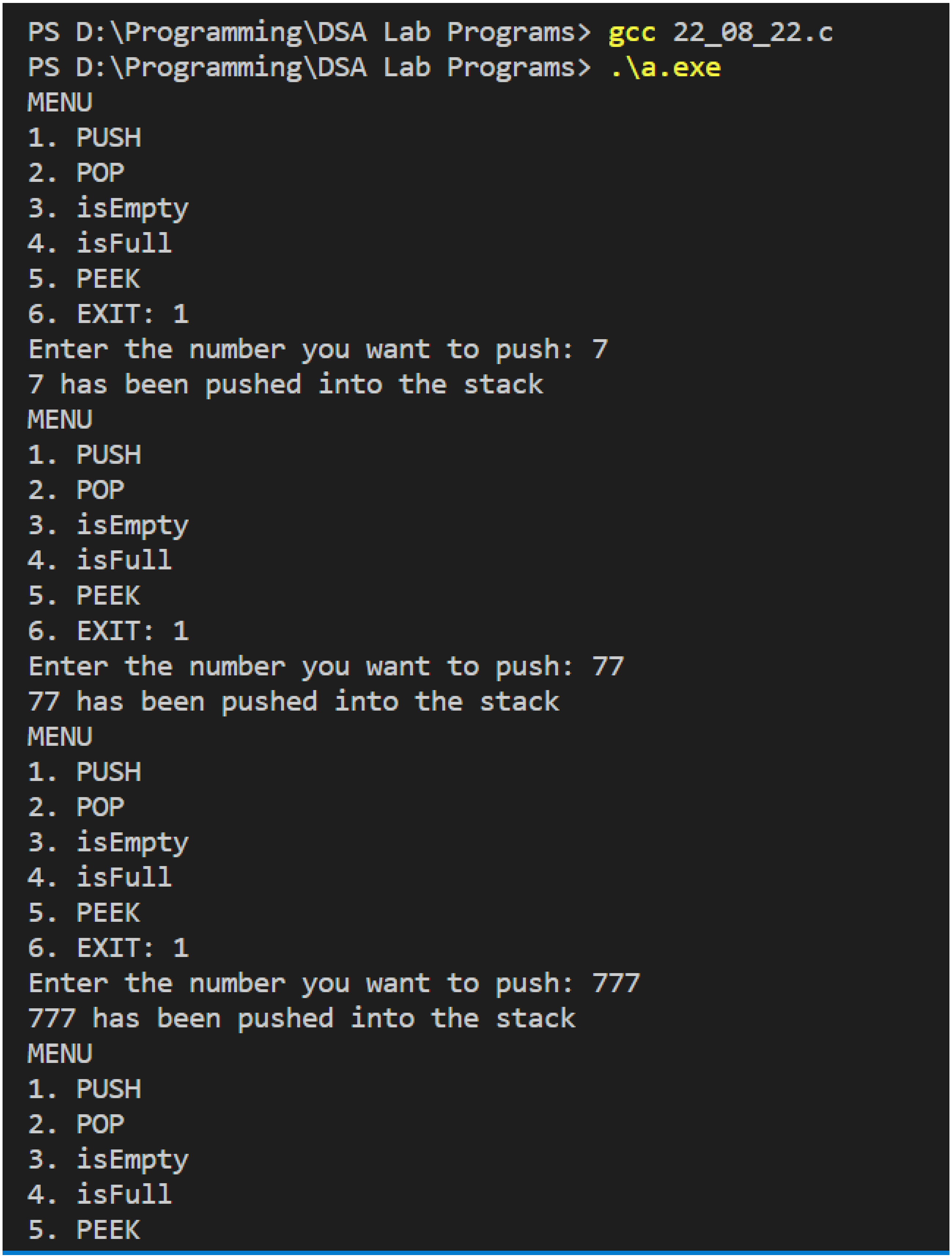
```
// Stack ADT
```

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

struct stack
{
    int size;
    int top;
    int *arr;
};

int isEmpty(struct stack *s)
{
    if (s->top == -1)
    {
        return 1;
    }
    else
    {
        return 0;
    }
}

int isFull(struct stack *s)
{
    if (s->top == s->size - 1)
    {
        return 1;
    }
    else
    {
        return 0;
    }
}

void push(struct stack *s, int val)
{
    if (isFull(s))
        printf("Stack OverFlow\n");
    else
    {
        s->top++;
        s->arr[s->top] = val;
        printf("%d has been pushed into the stack\n", val);
    }
}

void pop(struct stack *s)
{
    if (isEmpty(s))
        printf("Stack UnderFlow\n");
```

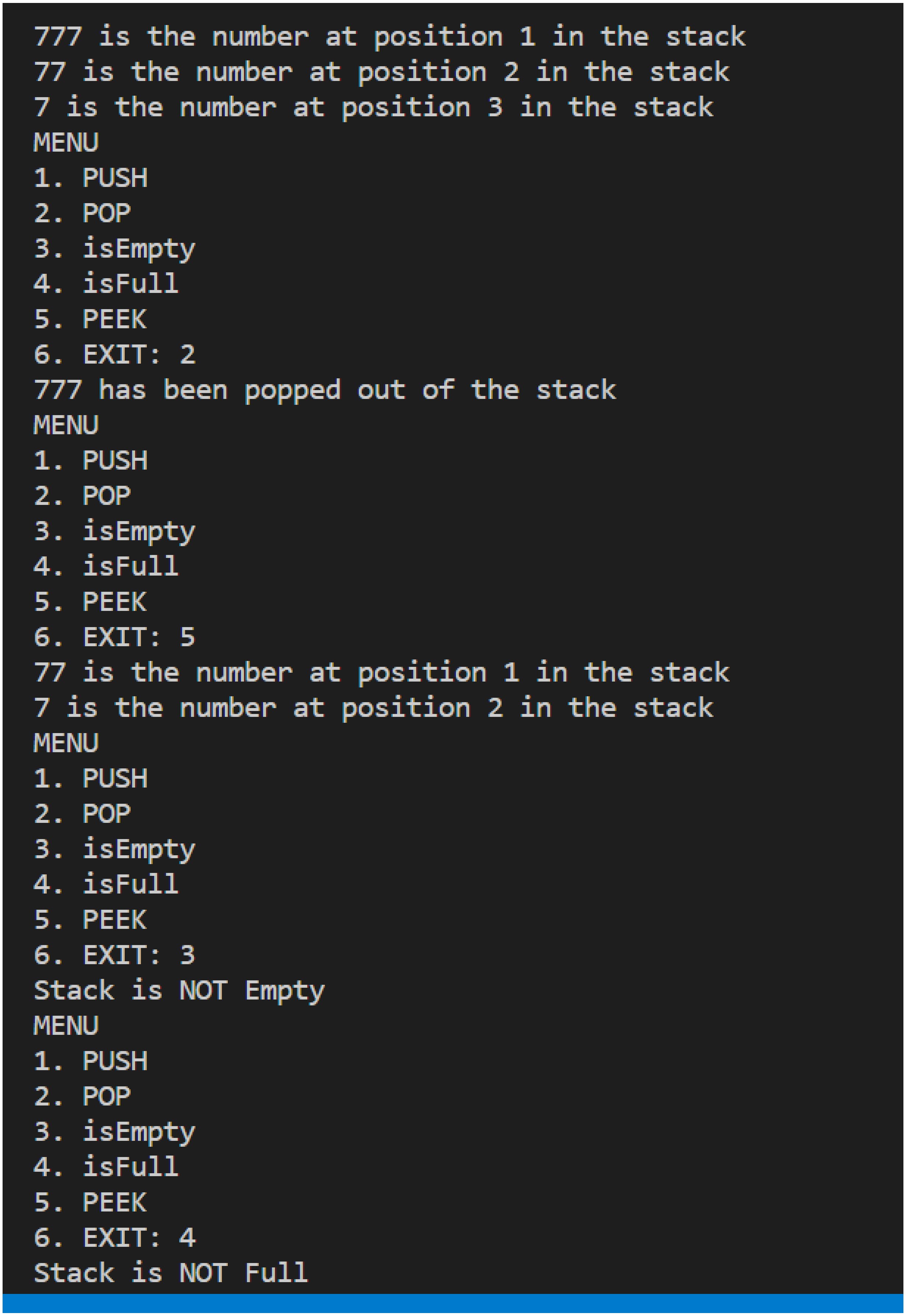


```
    else
    {
        int v = s->arr[s->top];
        // free(s->arr[s->top]);
        s->top--;
        printf("%d has been popped out of the stack\n", v);
    }
}

void peek(struct stack *s, int pos)
{
    if ((s->top - pos + 1) < 0)
    {
        printf("Invalid Position\n");
    }
    else
    {
        printf("%d is the number at position %d in the stack\n", s->arr[s->top - pos + 1],
pos);
    }
}

int main()
{
    struct stack *s;                                  // created a structure pointer(stores
the address of the structure)
    s = (struct stack *)malloc(sizeof(struct stack)); //(created an instance of structure)
    s->size = 10;                                     // size of stack
    s->top = -1;
    s->arr = (int *)malloc(s->size * sizeof(int)); // assigning heap memory for array(stack)
of integers

    while (1)
    {
        printf("MENU\n1. PUSH\n2. POP\n3. isEmpty\n4. isFull\n5. PEEK\n6. EXIT: ");
        int choice;
        scanf("%d", &choice);
        switch (choice)
        {
        case 1:
        {
            printf("Enter the number you want to push: ");
            int n;
            scanf("%d", &n);
            push(s, n);
            break;
        }
        case 2:
        {
            pop(s);
            break;
        }

        case 3:
        {
            if (isEmpty(s))
                printf("Stack UnderFlow\n");
            else
                printf("Stack is NOT Empty\n");
            break;
        }
        case 4:
        {
            if (isFull(s))
                printf("Stack OverFlow\n");
            else
                printf("Stack is NOT Full\n");
            break;
        }
        case 5:
        {
            if (isEmpty(s))
                printf("Stack UnderFlow\n");
            else
            {
                for (int i = 1; i <= s->top + 1; i++)
                    peek(s, i);
            }
            break;
        }
        case 6:
        {
            return 0;
            break;
        }
        default:
        {
            printf("Wrong Input!!!\n");
            break;
        }
        }
    }
    return 0;
}
```

Output: 4

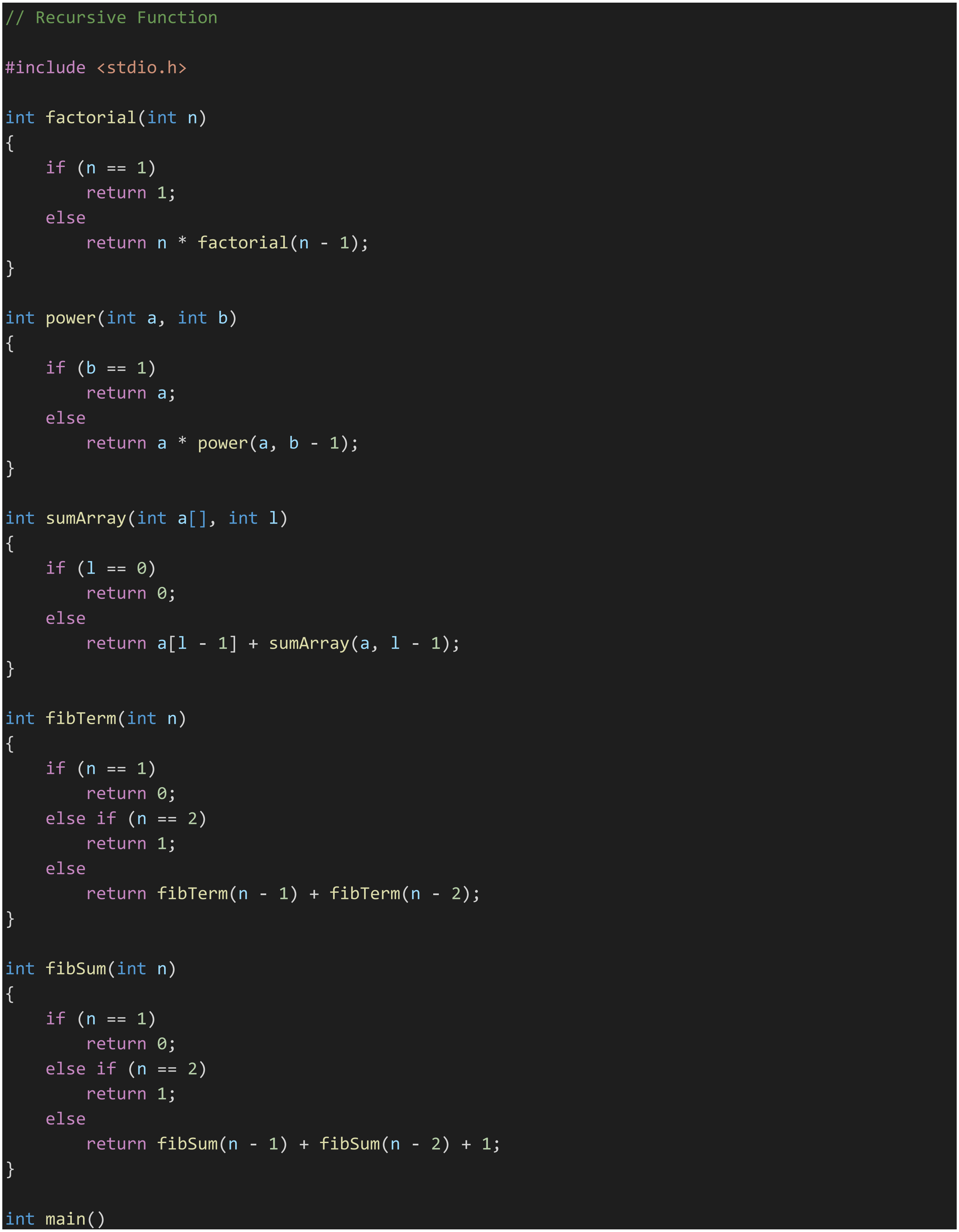


```
PS D:\Programming\DSA Lab Programs> gcc 22_08_22.c
PS D:\Programming\DSA Lab Programs> .\a.exe
MENU
1. PUSH
2. POP
3. isEmpty
4. isFull
5. PEEK
6. EXIT: 1
Enter the number you want to push: 7
7 has been pushed into the stack
MENU
1. PUSH
2. POP
3. isEmpty
4. isFull
5. PEEK
6. EXIT: 1
Enter the number you want to push: 77
77 has been pushed into the stack
MENU
1. PUSH
2. POP
3. isEmpty
4. isFull
5. PEEK
6. EXIT: 1
Enter the number you want to push: 777
777 has been pushed into the stack
MENU
1. PUSH
2. POP
3. isEmpty
4. isFull
5. PEEK
```

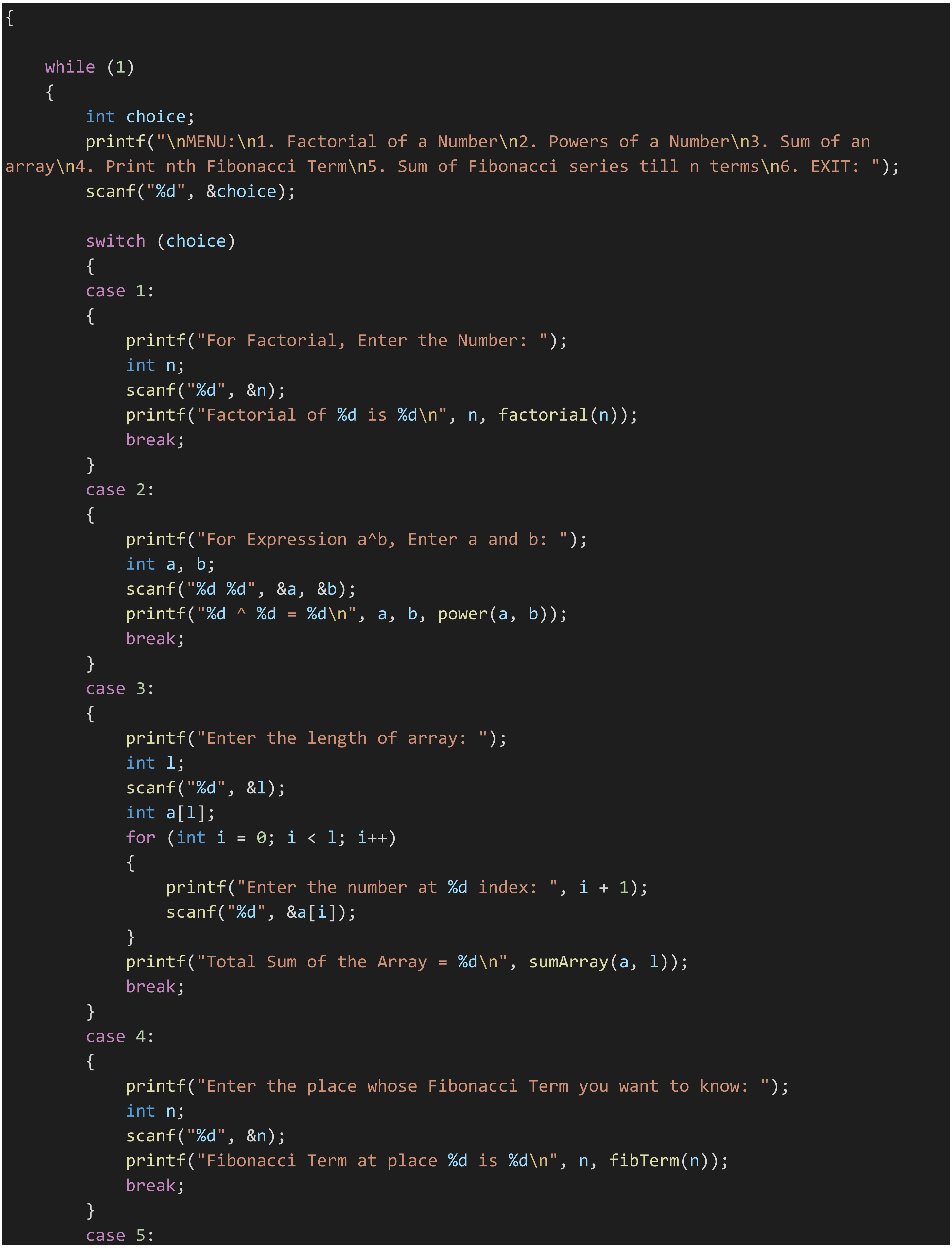


```
777 is the number at position 1 in the stack
77 is the number at position 2 in the stack
7 is the number at position 3 in the stack
MENU
1. PUSH
2. POP
3. isEmpty
4. isFull
5. PEEK
6. EXIT: 2
777 has been popped out of the stack
MENU
1. PUSH
2. POP
3. isEmpty
4. isFull
5. PEEK
6. EXIT: 5
77 is the number at position 1 in the stack
7 is the number at position 2 in the stack
MENU
1. PUSH
2. POP
3. isEmpty
4. isFull
5. PEEK
6. EXIT: 3
Stack is NOT Empty
MENU
1. PUSH
2. POP
3. isEmpty
4. isFull
5. PEEK
6. EXIT: 4
Stack is NOT Full
```

PROGRAM: 5



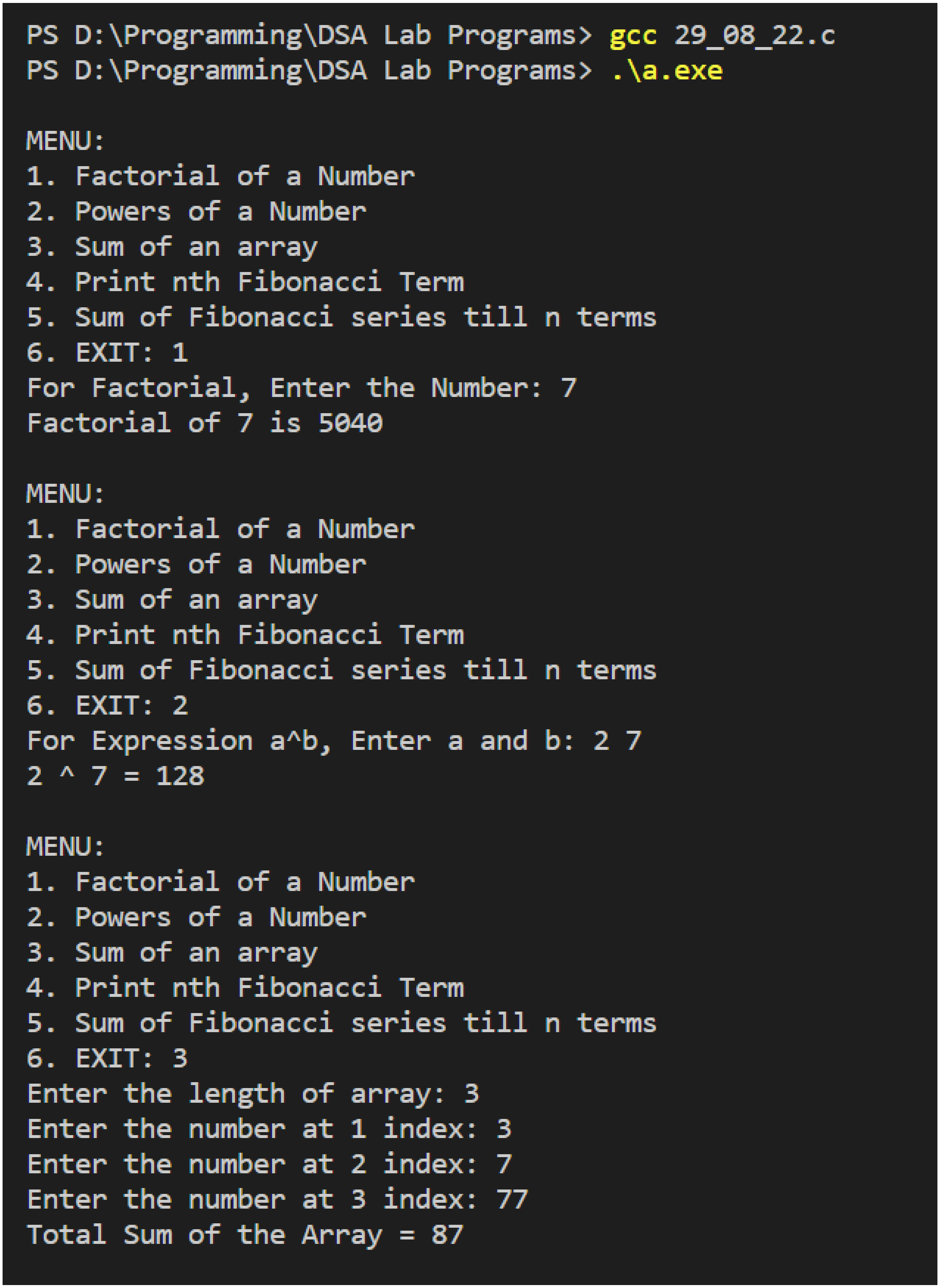
```c
// Recursive Function

#include <stdio.h>

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

int power(int a, int b)
{
    if (b == 1)
        return a;
    else
        return a * power(a, b - 1);
}

int sumArray(int a[], int l)
{
    if (l == 0)
        return 0;
    else
        return a[l - 1] + sumArray(a, l - 1);
}

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

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

int main()
```



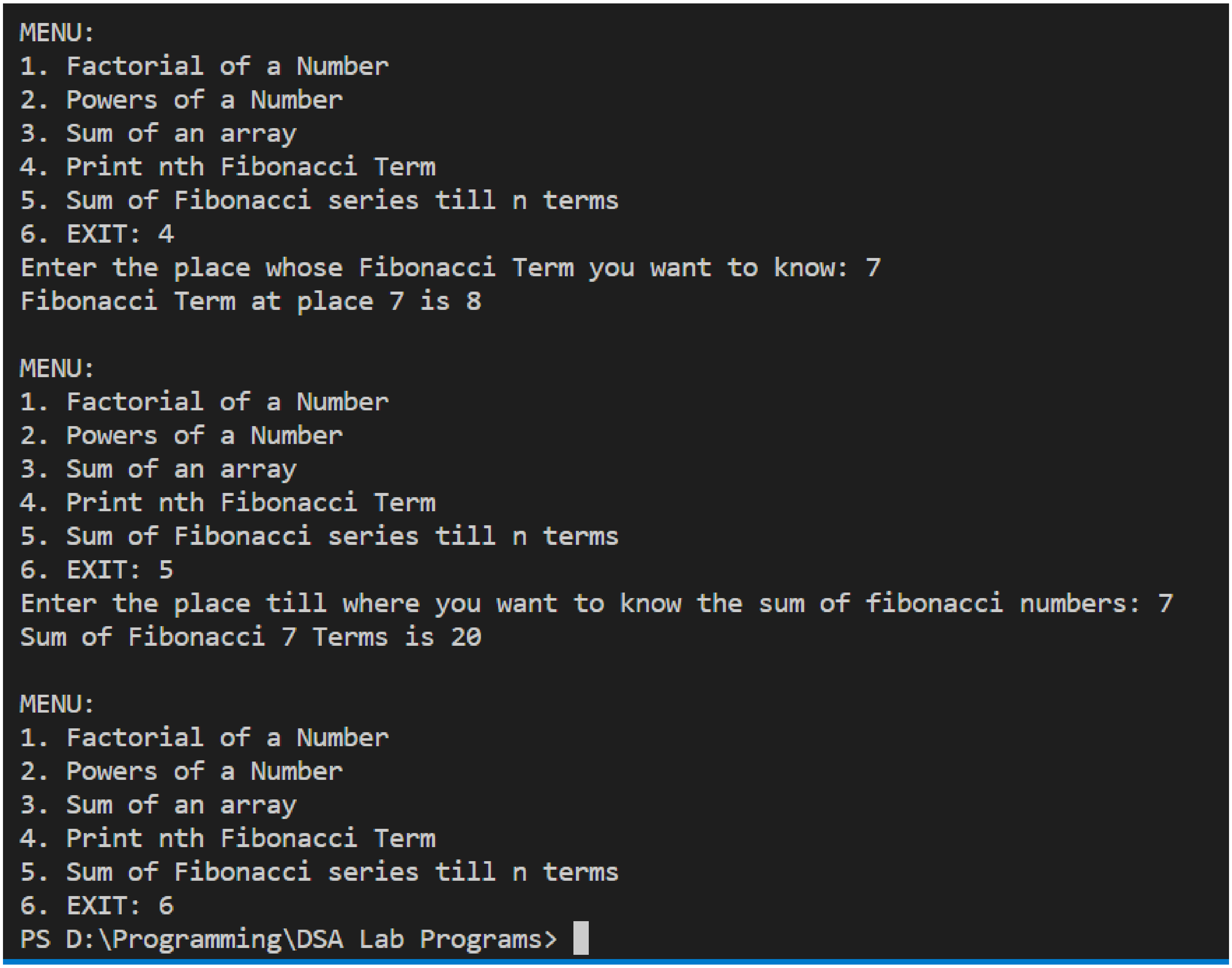
```
{

    while (1)
    {
        int choice;
        printf("\nMENU:\n1. Factorial of a Number\n2. Powers of a Number\n3. Sum of an array\n4. Print nth Fibonacci Term\n5. Sum of Fibonacci series till n terms\n6. EXIT: ");
        scanf("%d", &choice);

        switch (choice)
        {
        case 1:
        {
            printf("For Factorial, Enter the Number: ");
            int n;
            scanf("%d", &n);
            printf("Factorial of %d is %d\n", n, factorial(n));
            break;
        }
        case 2:
        {
            printf("For Expression a^b, Enter a and b: ");
            int a, b;
            scanf("%d %d", &a, &b);
            printf("%d ^ %d = %d\n", a, b, power(a, b));
            break;
        }
        case 3:
        {
            printf("Enter the length of array: ");
            int l;
            scanf("%d", &l);
            int a[l];
            for (int i = 0; i < l; i++)
            {
                printf("Enter the number at %d index: ", i + 1);
                scanf("%d", &a[i]);
            }
            printf("Total Sum of the Array = %d\n", sumArray(a, l));
            break;
        }
        case 4:
        {
            printf("Enter the place whose Fibonacci Term you want to know: ");
            int n;
            scanf("%d", &n);
            printf("Fibonacci Term at place %d is %d\n", n, fibTerm(n));
            break;
        }
        case 5:
```

```
        {
            printf("Enter the place till where you want to know the sum of fibonacci numbers: 
");
            int n;
            scanf("%d", &n);
            printf("Sum of Fibonacci %d Terms is %d\n", n, fibSum(n));
            break;
        }
        case 6:
        {
            return 0;
            break;
        }
        default:
        {
            printf("Invalid Input!!!");
            break;
        }
        }
    }
    return 0;
}
```

Output: 5

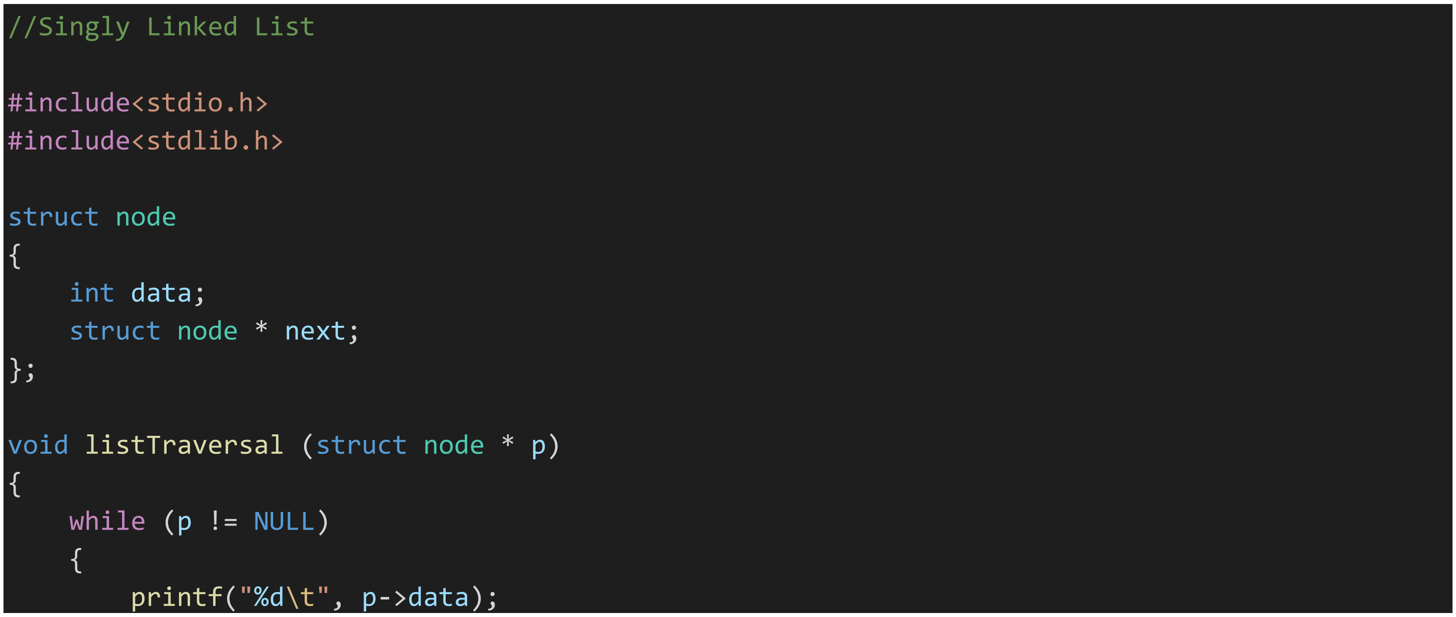


```
PS D:\Programming\DSA Lab Programs> gcc 29_08_22.c
PS D:\Programming\DSA Lab Programs> .\a.exe

MENU:
1. Factorial of a Number
2. Powers of a Number
3. Sum of an array
4. Print nth Fibonacci Term
5. Sum of Fibonacci series till n terms
6. EXIT: 1
For Factorial, Enter the Number: 7
Factorial of 7 is 5040

MENU:
1. Factorial of a Number
2. Powers of a Number
3. Sum of an array
4. Print nth Fibonacci Term
5. Sum of Fibonacci series till n terms
6. EXIT: 2
For Expression a^b, Enter a and b: 2 7
2 ^ 7 = 128

MENU:
1. Factorial of a Number
2. Powers of a Number
3. Sum of an array
4. Print nth Fibonacci Term
5. Sum of Fibonacci series till n terms
6. EXIT: 3
Enter the length of array: 3
Enter the number at 1 index: 3
Enter the number at 2 index: 7
Enter the number at 3 index: 77
Total Sum of the Array = 87
```

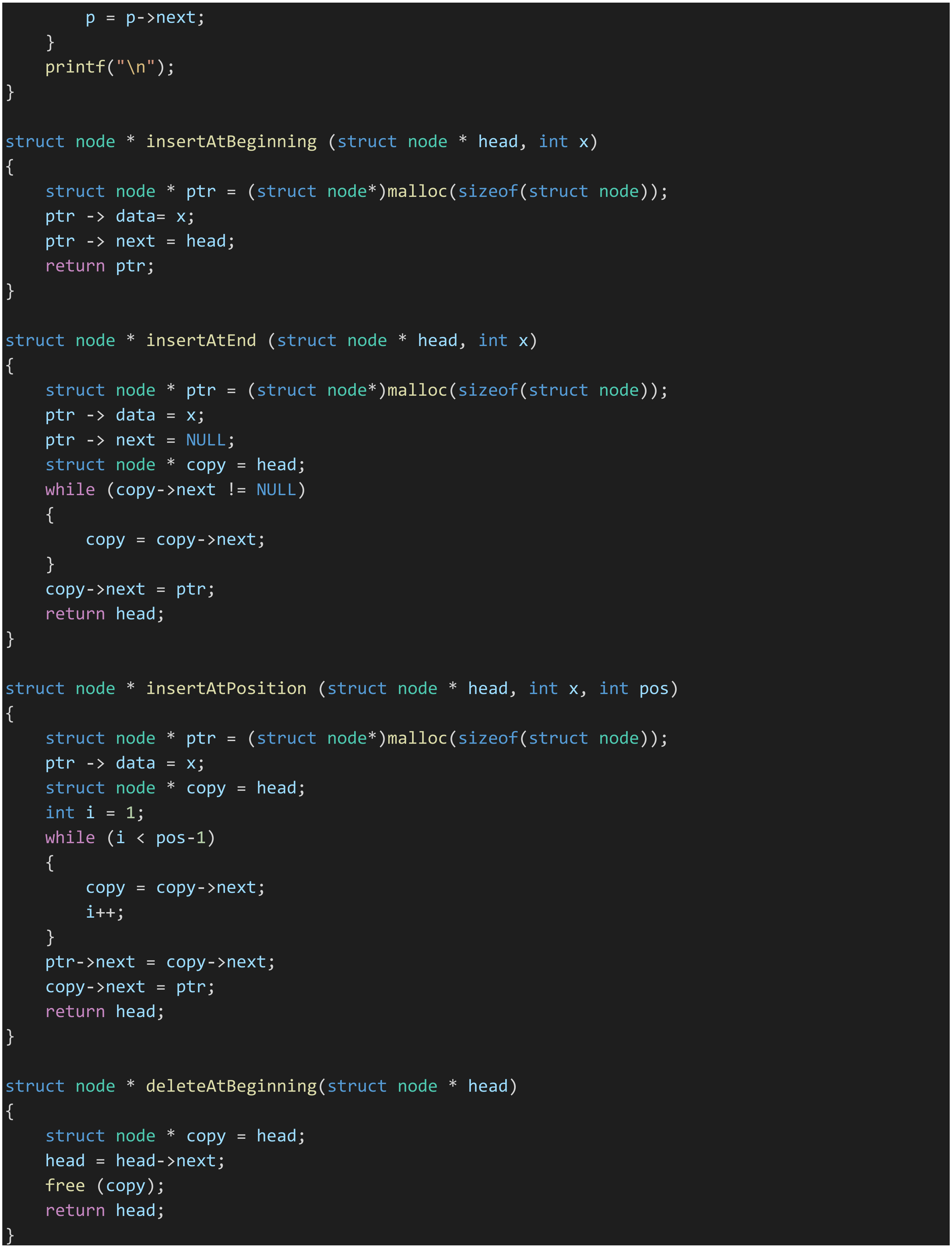


```
MENU:
1. Factorial of a Number
2. Powers of a Number
3. Sum of an array
4. Print nth Fibonacci Term
5. Sum of Fibonacci series till n terms
6. EXIT: 4
Enter the place whose Fibonacci Term you want to know: 7
Fibonacci Term at place 7 is 8

MENU:
1. Factorial of a Number
2. Powers of a Number
3. Sum of an array
4. Print nth Fibonacci Term
5. Sum of Fibonacci series till n terms
6. EXIT: 5
Enter the place till where you want to know the sum of fibonacci numbers: 7
Sum of Fibonacci 7 Terms is 20

MENU:
1. Factorial of a Number
2. Powers of a Number
3. Sum of an array
4. Print nth Fibonacci Term
5. Sum of Fibonacci series till n terms
6. EXIT: 6
PS D:\Programming\DSA Lab Programs>
```

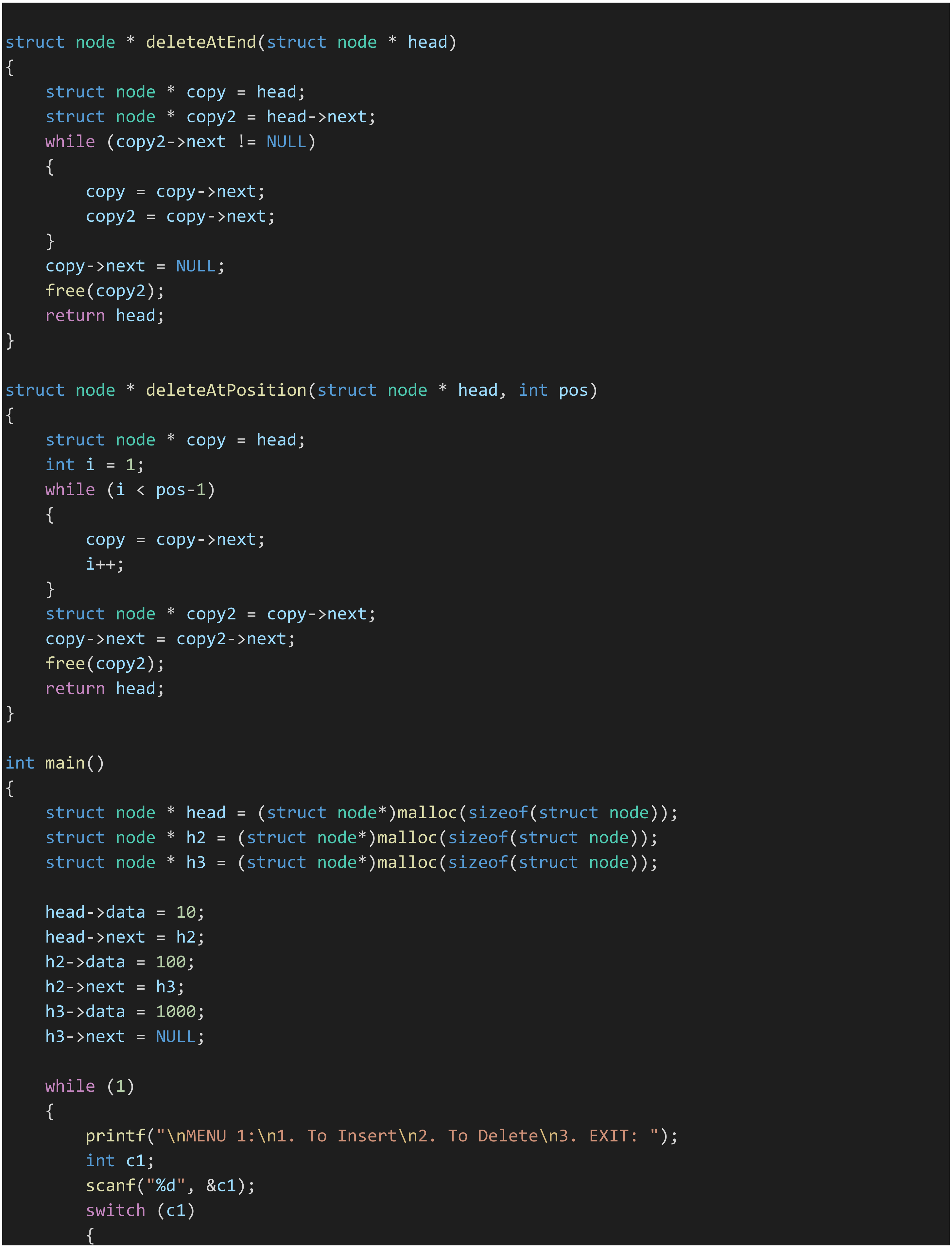
PROGRAM: 6



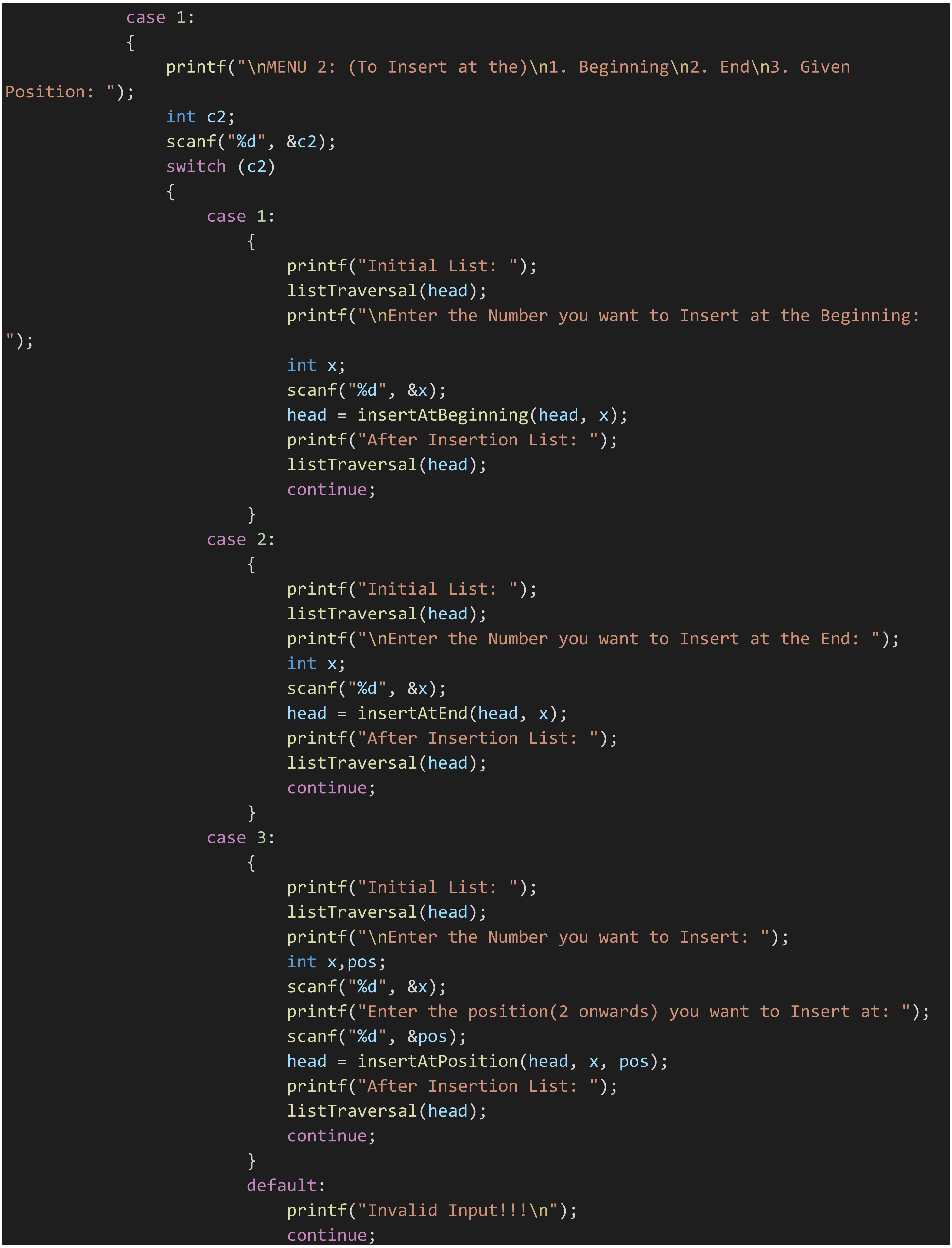
```c
//Singly Linked List

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

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

void listTraversal (struct node * p)
{
    while (p != NULL)
    {
        printf("%d\t", p->data);
```



```
        p = p->next;
    }
    printf("\n");
}

struct node * insertAtBeginning (struct node * head, int x)
{
    struct node * ptr = (struct node*)malloc(sizeof(struct node));
    ptr -> data= x;
    ptr -> next = head;
    return ptr;
}

struct node * insertAtEnd (struct node * head, int x)
{
    struct node * ptr = (struct node*)malloc(sizeof(struct node));
    ptr -> data = x;
    ptr -> next = NULL;
    struct node * copy = head;
    while (copy->next != NULL)
    {
        copy = copy->next;
    }
    copy->next = ptr;
    return head;
}

struct node * insertAtPosition (struct node * head, int x, int pos)
{
    struct node * ptr = (struct node*)malloc(sizeof(struct node));
    ptr -> data = x;
    struct node * copy = head;
    int i = 1;
    while (i < pos-1)
    {
        copy = copy->next;
        i++;
    }
    ptr->next = copy->next;
    copy->next = ptr;
    return head;
}

struct node * deleteAtBeginning(struct node * head)
{
    struct node * copy = head;
    head = head->next;
    free (copy);
    return head;
}
```

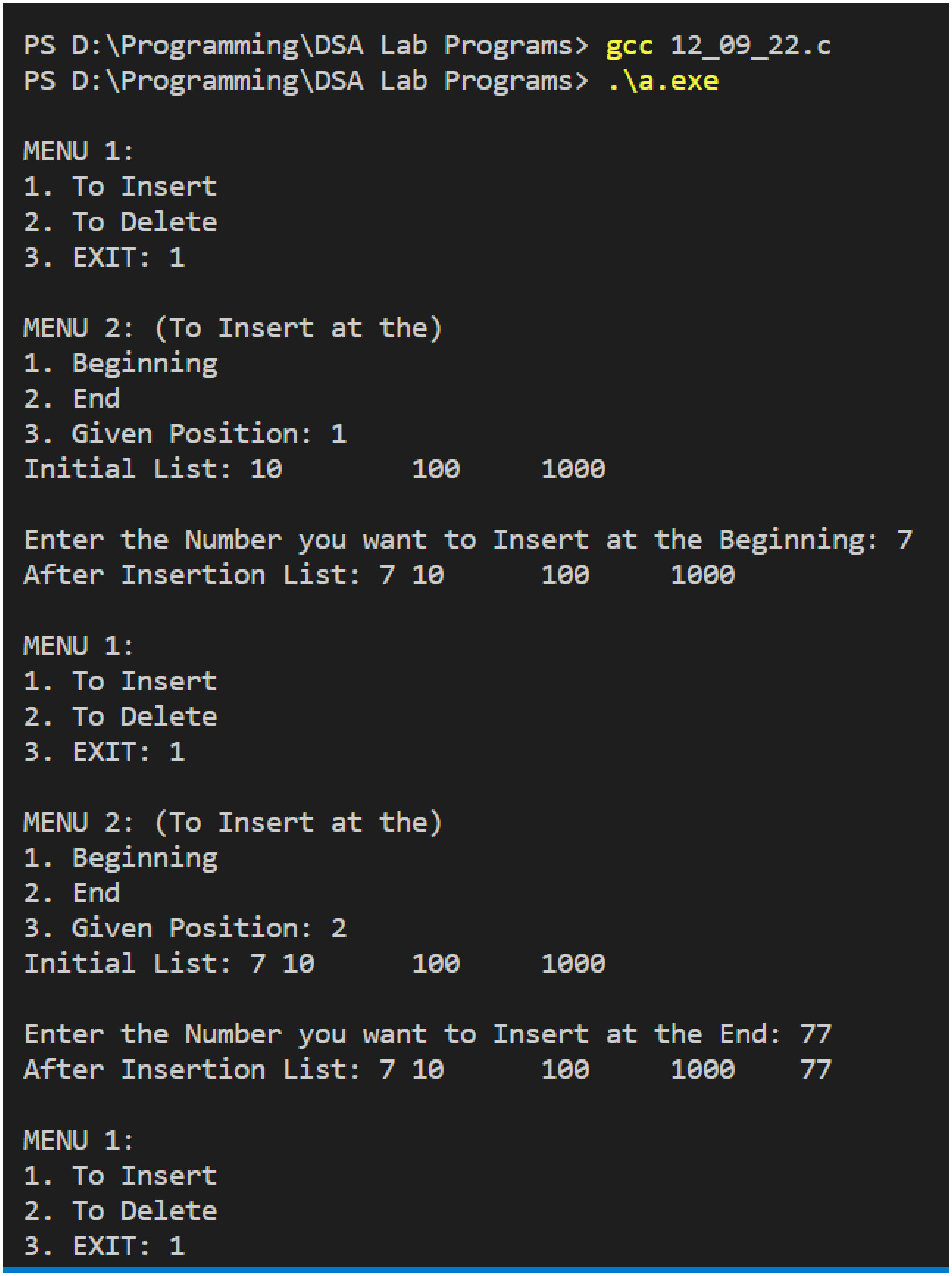


```
struct node * deleteAtEnd(struct node * head)
{
    struct node * copy = head;
    struct node * copy2 = head->next;
    while (copy2->next != NULL)
    {
        copy = copy->next;
        copy2 = copy->next;
    }
    copy->next = NULL;
    free(copy2);
    return head;
}

struct node * deleteAtPosition(struct node * head, int pos)
{
    struct node * copy = head;
    int i = 1;
    while (i < pos-1)
    {
        copy = copy->next;
        i++;
    }
    struct node * copy2 = copy->next;
    copy->next = copy2->next;
    free(copy2);
    return head;
}

int main()
{
    struct node * head = (struct node*)malloc(sizeof(struct node));
    struct node * h2 = (struct node*)malloc(sizeof(struct node));
    struct node * h3 = (struct node*)malloc(sizeof(struct node));

    head->data = 10;
    head->next = h2;
    h2->data = 100;
    h2->next = h3;
    h3->data = 1000;
    h3->next = NULL;

    while (1)
    {
        printf("\nMENU 1:\n1. To Insert\n2. To Delete\n3. EXIT: ");
        int c1;
        scanf("%d", &c1);
        switch (c1)
        {
```

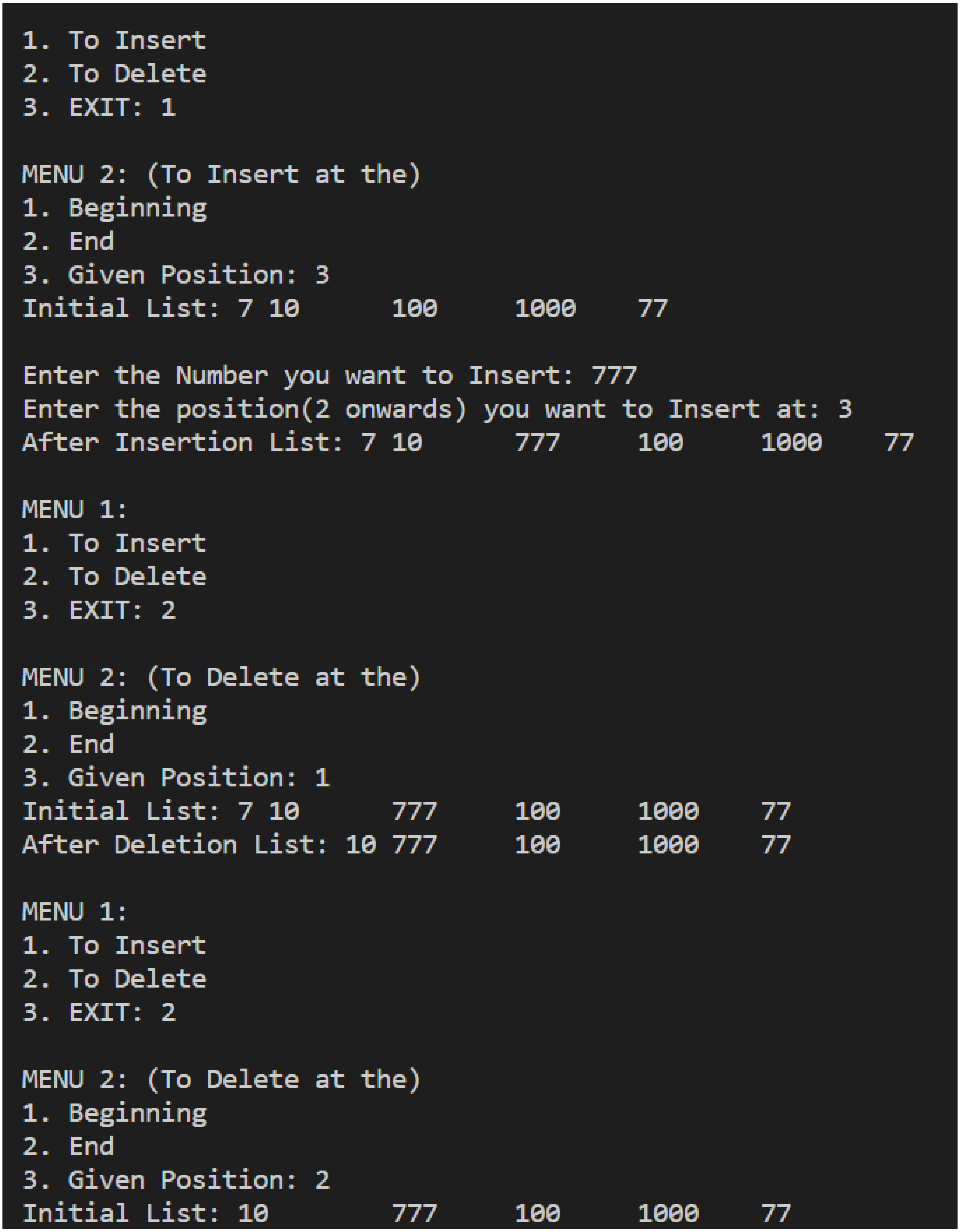


```
            case 1:
            {
                printf("\nMENU 2: (To Insert at the)\n1. Beginning\n2. End\n3. Given
Position: ");
                int c2;
                scanf("%d", &c2);
                switch (c2)
                {
                    case 1:
                        {
                            printf("Initial List: ");
                            listTraversal(head);
                            printf("\nEnter the Number you want to Insert at the Beginning:
");
                            int x;
                            scanf("%d", &x);
                            head = insertAtBeginning(head, x);
                            printf("After Insertion List: ");
                            listTraversal(head);
                            continue;
                        }
                    case 2:
                        {
                            printf("Initial List: ");
                            listTraversal(head);
                            printf("\nEnter the Number you want to Insert at the End: ");
                            int x;
                            scanf("%d", &x);
                            head = insertAtEnd(head, x);
                            printf("After Insertion List: ");
                            listTraversal(head);
                            continue;
                        }
                    case 3:
                        {
                            printf("Initial List: ");
                            listTraversal(head);
                            printf("\nEnter the Number you want to Insert: ");
                            int x,pos;
                            scanf("%d", &x);
                            printf("Enter the position(2 onwards) you want to Insert at: ");
                            scanf("%d", &pos);
                            head = insertAtPosition(head, x, pos);
                            printf("After Insertion List: ");
                            listTraversal(head);
                            continue;
                        }
                        default:
                            printf("Invalid Input!!!\n");
                            continue;
                }
            }
            case 2:
            {
                printf("\nMENU 2: (To Delete at the)\n1. Beginning\n2. End\n3. Given
Position: ");
                int c2;
                scanf("%d", &c2);
                switch (c2)
                {
                    case 1:
                        {
                            printf("Initial List: ");
                            listTraversal(head);
                            head = deleteAtBeginning(head);
                            printf("After Deletion List: ");
                            listTraversal(head);
                            continue;
                        }
                    case 2:
                        {
                            printf("Initial List: ");
                            listTraversal(head);
                            head = deleteAtEnd(head);
                            printf("After Deletion List: ");
                            listTraversal(head);
                            continue;
                        }
                    case 3:
                        {
                            printf("Initial List: ");
                            listTraversal(head);
                            printf("Enter the position(2 onwards) you want to Delete at: ");
                            int pos;
                            scanf("%d", &pos);
                            head = deleteAtPosition(head, pos);
                            printf("After Deletion List: ");
                            listTraversal(head);
                            continue;
                        }
                        default:
                            printf("Invalid Input!!!\n");
                            continue;
                }
            }
            case 3:
                return 0;
                break;
            default:
                printf("Invalid Input!!!");
                break;
        }
    }
}
```

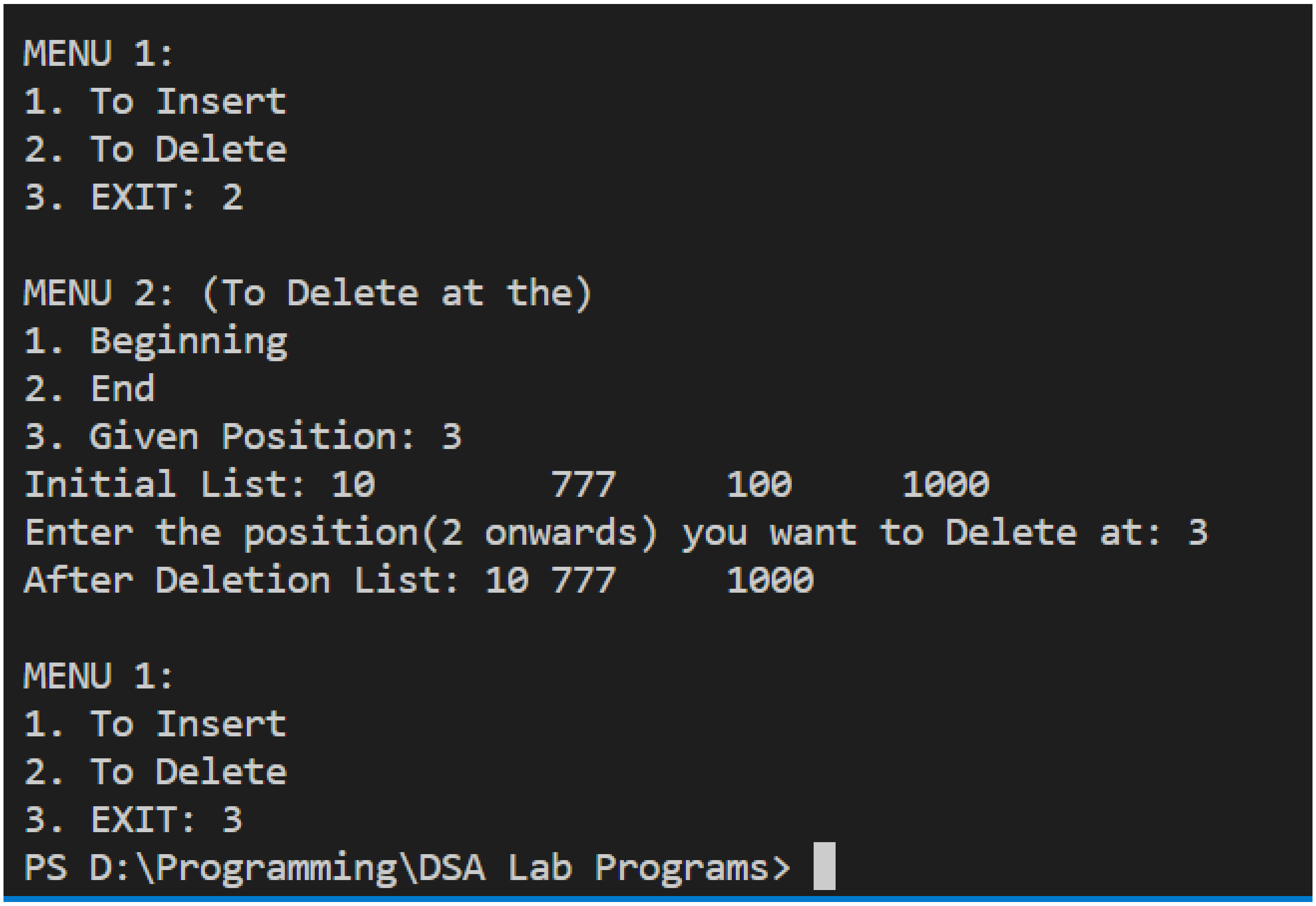
Output: 6



```
PS D:\Programming\DSA Lab Programs> gcc 12_09_22.c
PS D:\Programming\DSA Lab Programs> .\a.exe

MENU 1:
1. To Insert
2. To Delete
3. EXIT: 1

MENU 2: (To Insert at the)
1. Beginning
2. End
3. Given Position: 1
Initial List: 10        100     1000

Enter the Number you want to Insert at the Beginning: 7
After Insertion List: 7 10      100     1000

MENU 1:
1. To Insert
2. To Delete
3. EXIT: 1

MENU 2: (To Insert at the)
1. Beginning
2. End
3. Given Position: 2
Initial List: 7 10      100     1000

Enter the Number you want to Insert at the End: 77
After Insertion List: 7 10      100     1000    77

MENU 1:
1. To Insert
2. To Delete
3. EXIT: 1
```

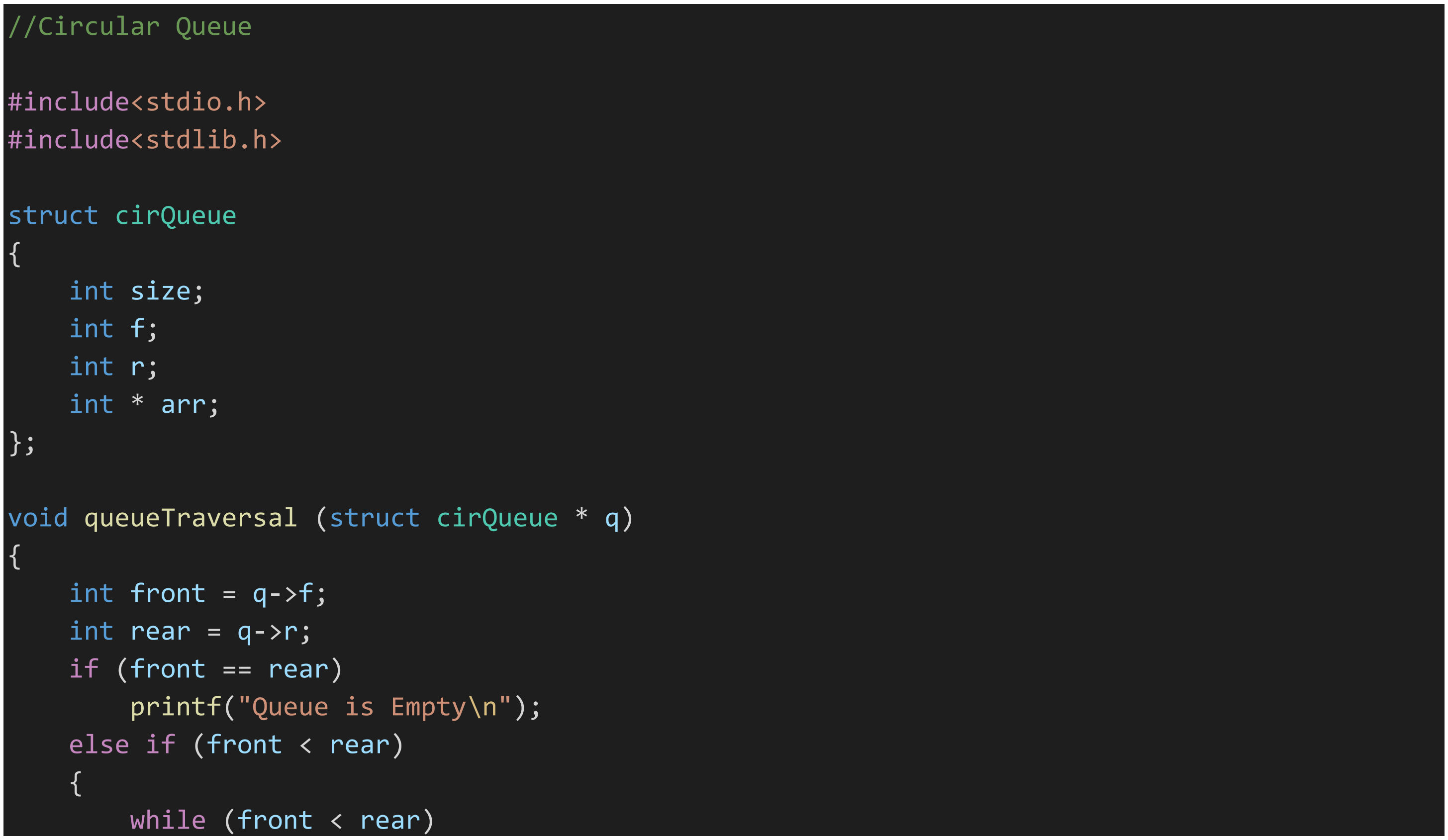


```
1. To Insert
2. To Delete
3. EXIT: 1

MENU 2: (To Insert at the)
1. Beginning
2. End
3. Given Position: 3
Initial List: 7 10      100     1000    77

Enter the Number you want to Insert: 777
Enter the position(2 onwards) you want to Insert at: 3
After Insertion List: 7 10      777     100     1000    77

MENU 1:
1. To Insert
2. To Delete
3. EXIT: 2

MENU 2: (To Delete at the)
1. Beginning
2. End
3. Given Position: 1
Initial List: 7 10      777     100     1000    77
After Deletion List: 10 777     100     1000    77

MENU 1:
1. To Insert
2. To Delete
3. EXIT: 2

MENU 2: (To Delete at the)
1. Beginning
2. End
3. Given Position: 2
Initial List: 10        777     100     1000    77
```

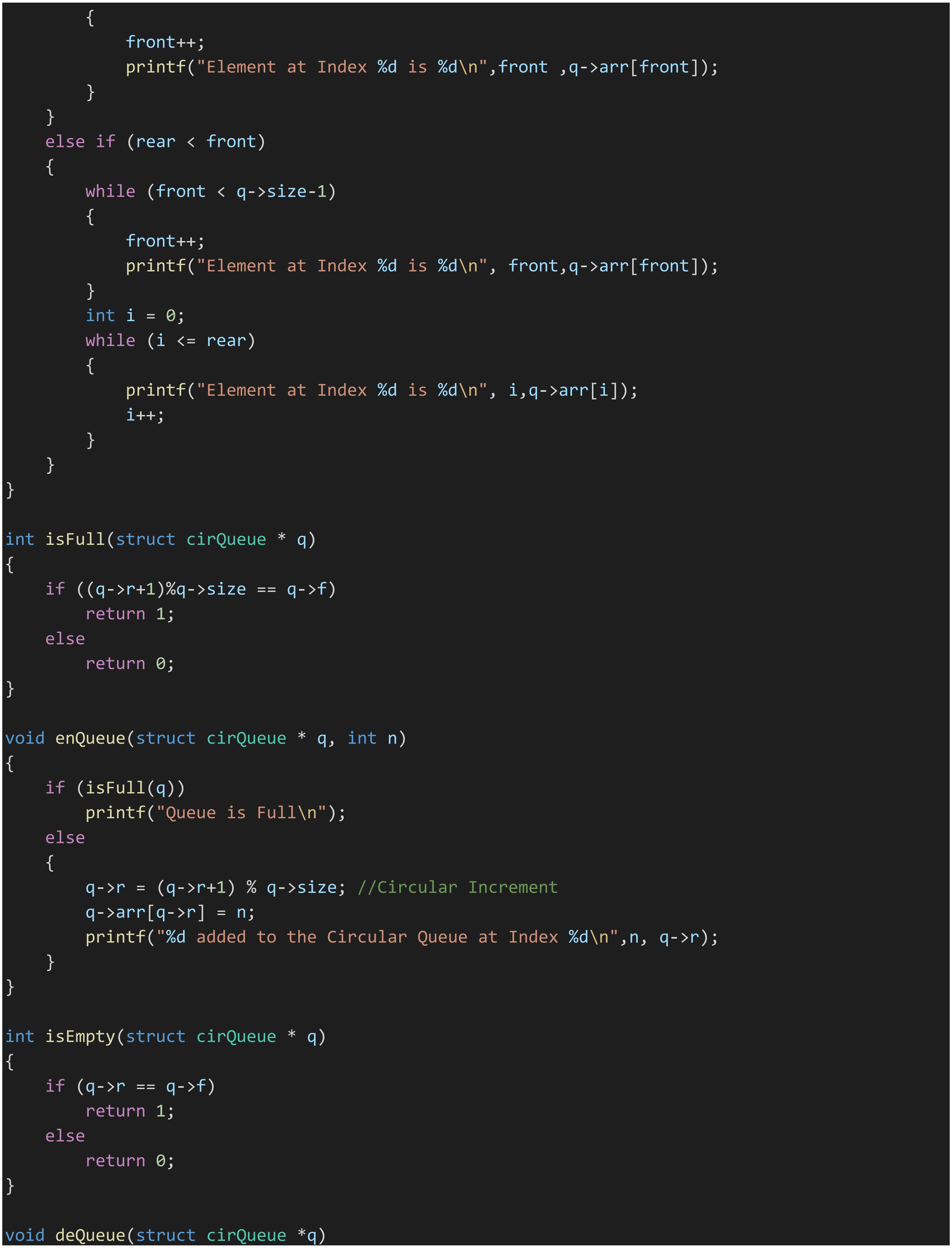


```
MENU 1:
1. To Insert
2. To Delete
3. EXIT: 2

MENU 2: (To Delete at the)
1. Beginning
2. End
3. Given Position: 3
Initial List: 10        777     100     1000
Enter the position(2 onwards) you want to Delete at: 3
After Deletion List: 10 777     1000

MENU 1:
1. To Insert
2. To Delete
3. EXIT: 3
PS D:\Programming\DSA Lab Programs>
```

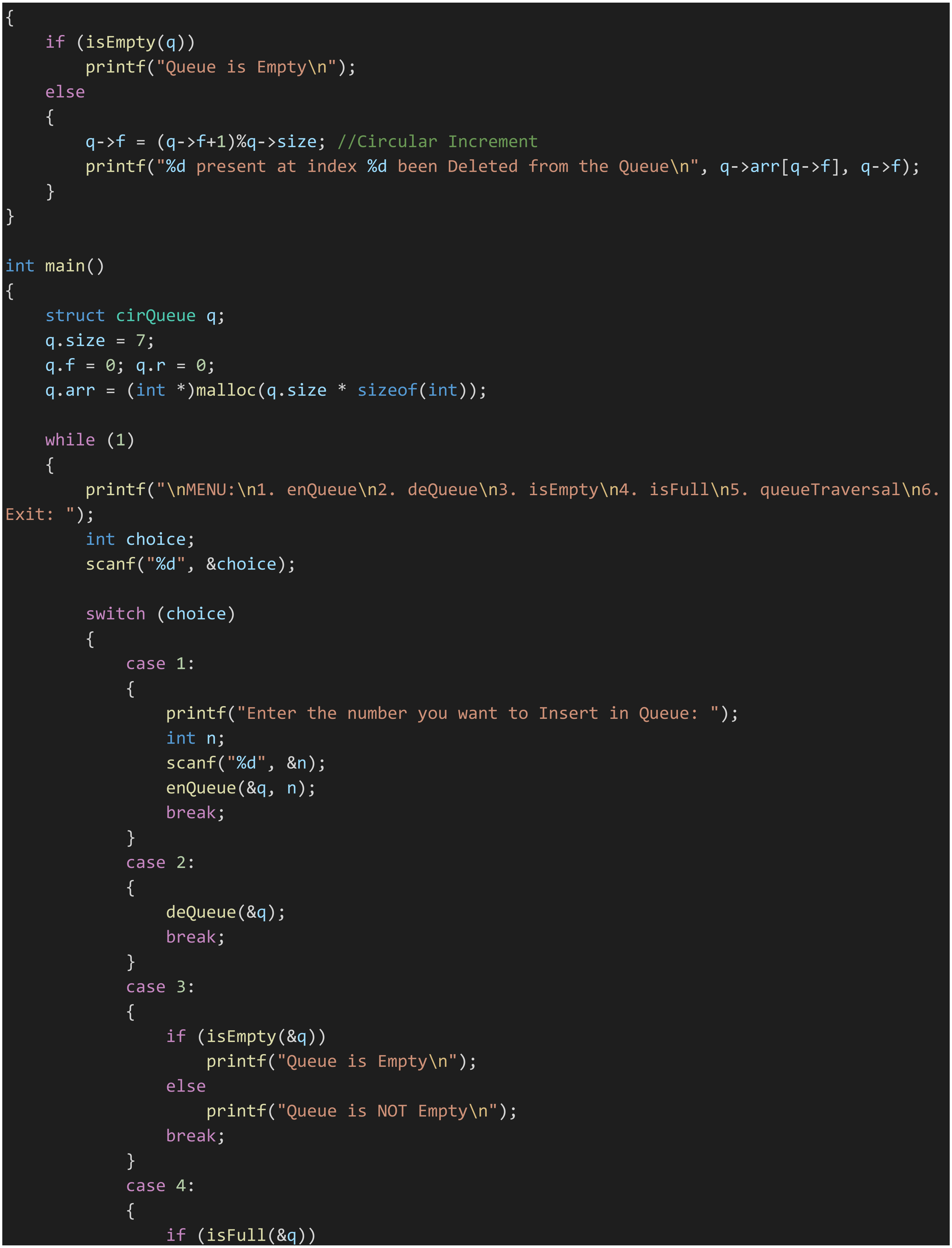
PROGRAM: 7



```c
//Circular Queue

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

struct cirQueue
{
    int size;
    int f;
    int r;
    int * arr;
};

void queueTraversal (struct cirQueue * q)
{
    int front = q->f;
    int rear = q->r;
    if (front == rear)
        printf("Queue is Empty\n");
    else if (front < rear)
    {
        while (front < rear)
```



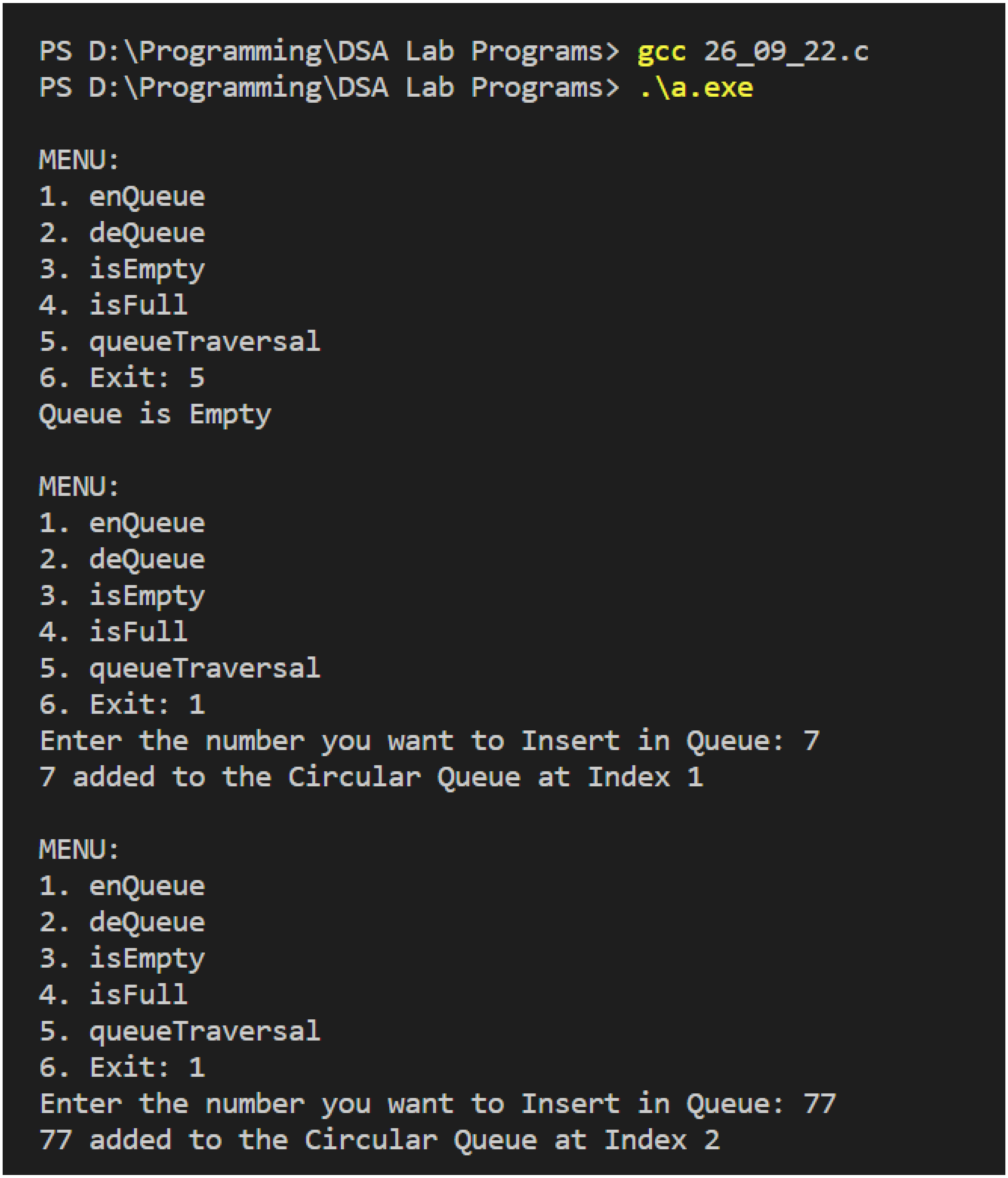
```
        {
            front++;
            printf("Element at Index %d is %d\n",front ,q->arr[front]);
        }
    }
    else if (rear < front)
    {
        while (front < q->size-1)
        {
            front++;
            printf("Element at Index %d is %d\n", front,q->arr[front]);
        }
        int i = 0;
        while (i <= rear)
        {
            printf("Element at Index %d is %d\n", i,q->arr[i]);
            i++;
        }
    }
}

int isFull(struct cirQueue * q)
{
    if ((q->r+1)%q->size == q->f)
        return 1;
    else
        return 0;
}

void enQueue(struct cirQueue * q, int n)
{
    if (isFull(q))
        printf("Queue is Full\n");
    else
    {
        q->r = (q->r+1) % q->size; //Circular Increment
        q->arr[q->r] = n;
        printf("%d added to the Circular Queue at Index %d\n",n, q->r);
    }
}

int isEmpty(struct cirQueue * q)
{
    if (q->r == q->f)
        return 1;
    else
        return 0;
}

void deQueue(struct cirQueue *q)
```



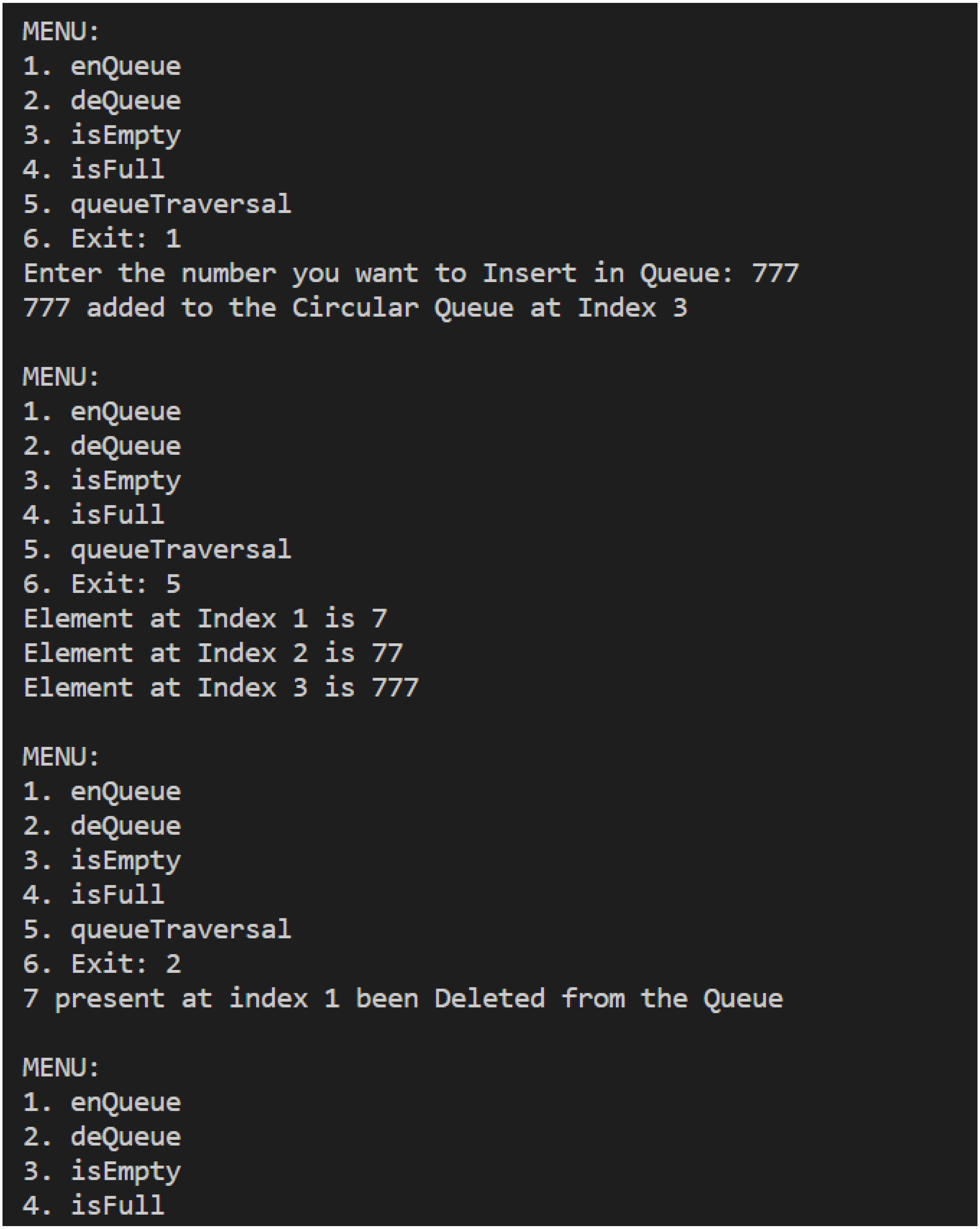
```
{
    if (isEmpty(q))
        printf("Queue is Empty\n");
    else
    {
        q->f = (q->f+1)%q->size; //Circular Increment
        printf("%d present at index %d been Deleted from the Queue\n", q->arr[q->f], q->f);
    }
}

int main()
{
    struct cirQueue q;
    q.size = 7;
    q.f = 0; q.r = 0;
    q.arr = (int *)malloc(q.size * sizeof(int));

    while (1)
    {
        printf("\nMENU:\n1. enQueue\n2. deQueue\n3. isEmpty\n4. isFull\n5. queueTraversal\n6.
Exit: ");
        int choice;
        scanf("%d", &choice);

        switch (choice)
        {
            case 1:
            {
                printf("Enter the number you want to Insert in Queue: ");
                int n;
                scanf("%d", &n);
                enQueue(&q, n);
                break;
            }
            case 2:
            {
                deQueue(&q);
                break;
            }
            case 3:
            {
                if (isEmpty(&q))
                    printf("Queue is Empty\n");
                else
                    printf("Queue is NOT Empty\n");
                break;
            }
            case 4:
            {
                if (isFull(&q))
```

```c
                    printf("Queue is Full\n");
                else
                    printf("Queue is NOT Full\n");
                break;
            }
            case 5:
            {
                queueTraversal(&q);
                break;
            }
            case 6:
                return 0;
                break;
            default:
                printf("Invalid Input!!!");
                break;
        }
    }

    return 0;
}
```

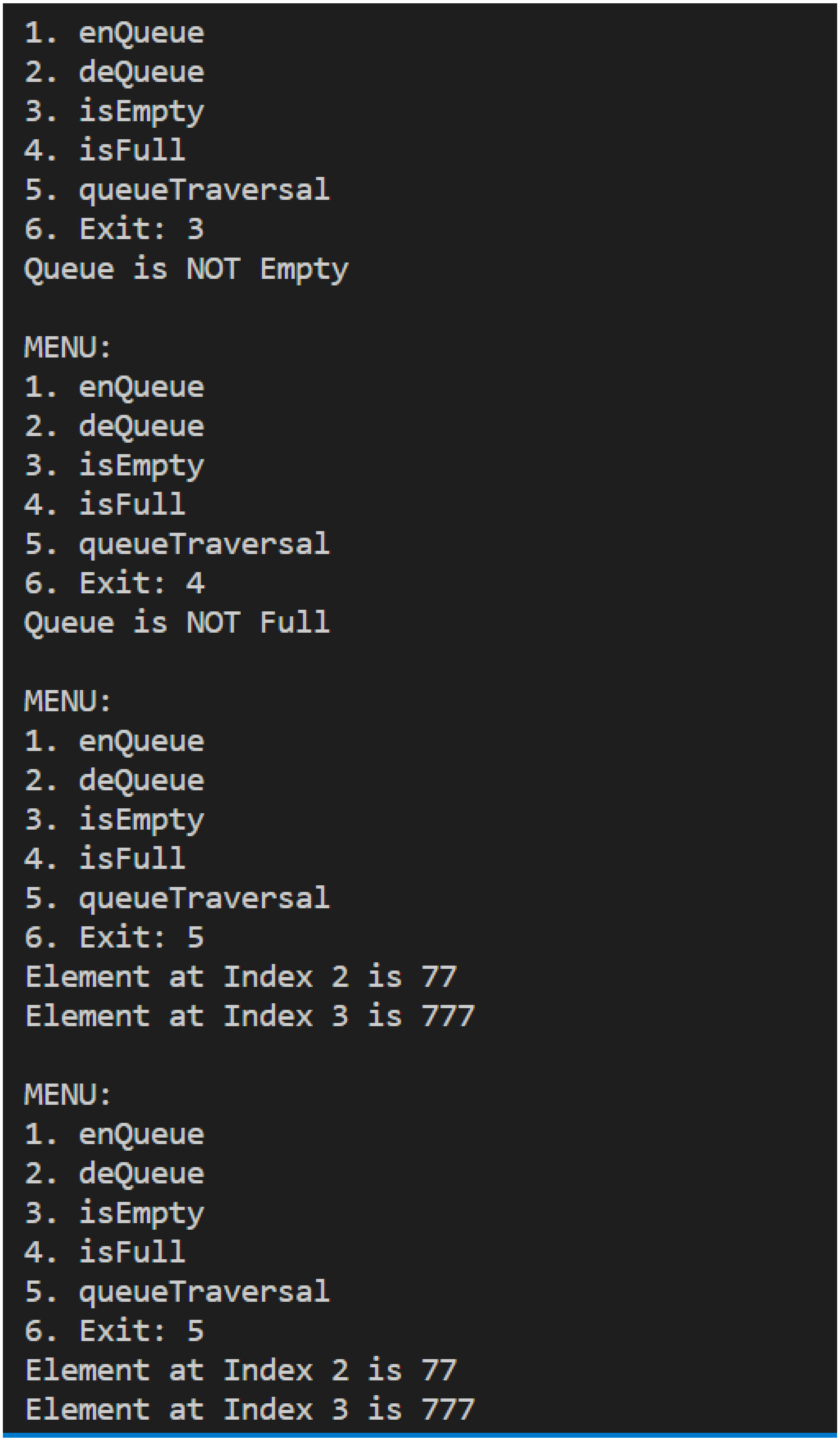
Output: 7



```
PS D:\Programming\DSA Lab Programs> gcc 26_09_22.c
PS D:\Programming\DSA Lab Programs> .\a.exe

MENU:
1. enQueue
2. deQueue
3. isEmpty
4. isFull
5. queueTraversal
6. Exit: 5
Queue is Empty

MENU:
1. enQueue
2. deQueue
3. isEmpty
4. isFull
5. queueTraversal
6. Exit: 1
Enter the number you want to Insert in Queue: 7
7 added to the Circular Queue at Index 1

MENU:
1. enQueue
2. deQueue
3. isEmpty
4. isFull
5. queueTraversal
6. Exit: 1
Enter the number you want to Insert in Queue: 77
77 added to the Circular Queue at Index 2
```

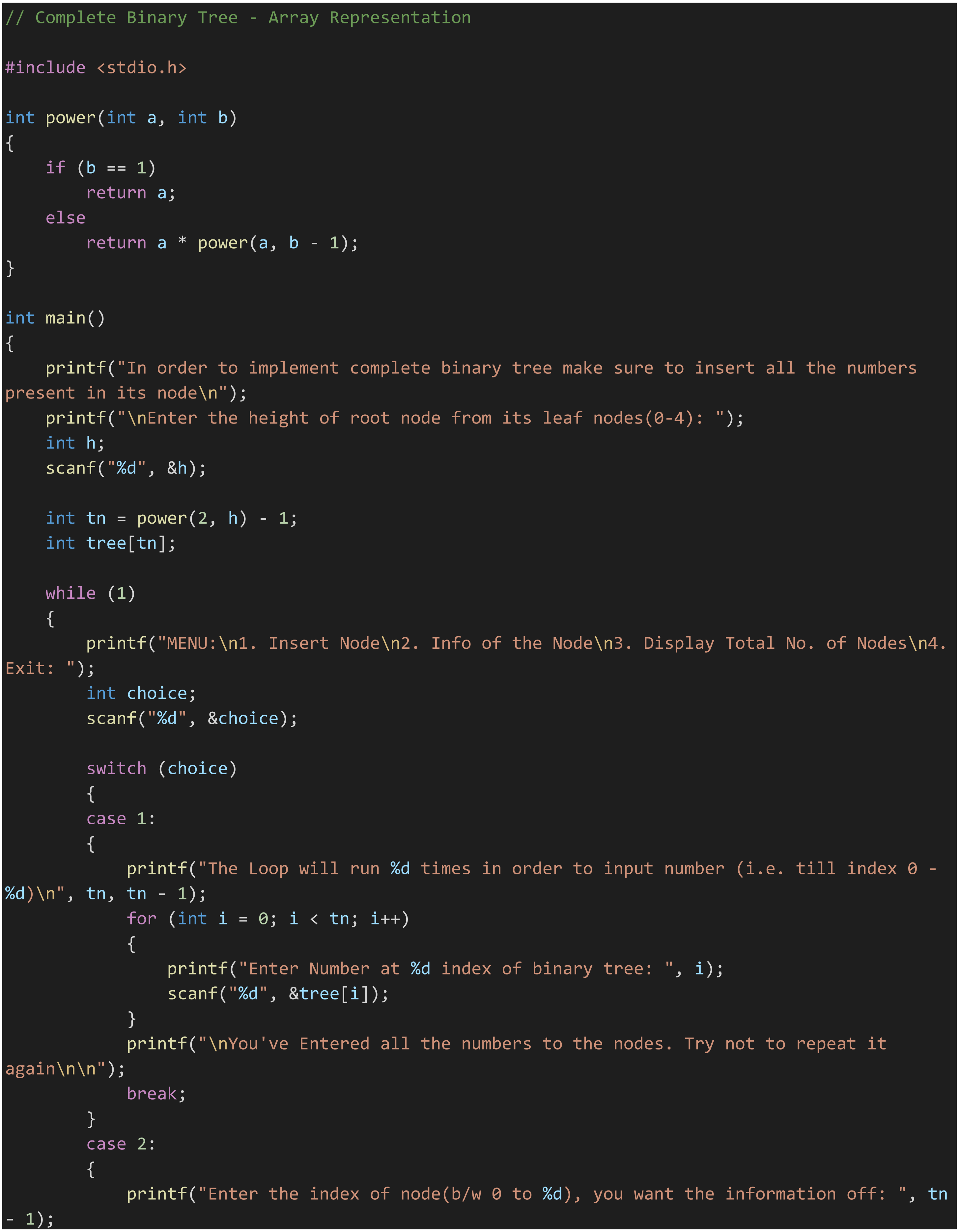


```
MENU:
1. enQueue
2. deQueue
3. isEmpty
4. isFull
5. queueTraversal
6. Exit: 1
Enter the number you want to Insert in Queue: 777
777 added to the Circular Queue at Index 3

MENU:
1. enQueue
2. deQueue
3. isEmpty
4. isFull
5. queueTraversal
6. Exit: 5
Element at Index 1 is 7
Element at Index 2 is 77
Element at Index 3 is 777

MENU:
1. enQueue
2. deQueue
3. isEmpty
4. isFull
5. queueTraversal
6. Exit: 2
7 present at index 1 been Deleted from the Queue

MENU:
1. enQueue
2. deQueue
3. isEmpty
4. isFull
```



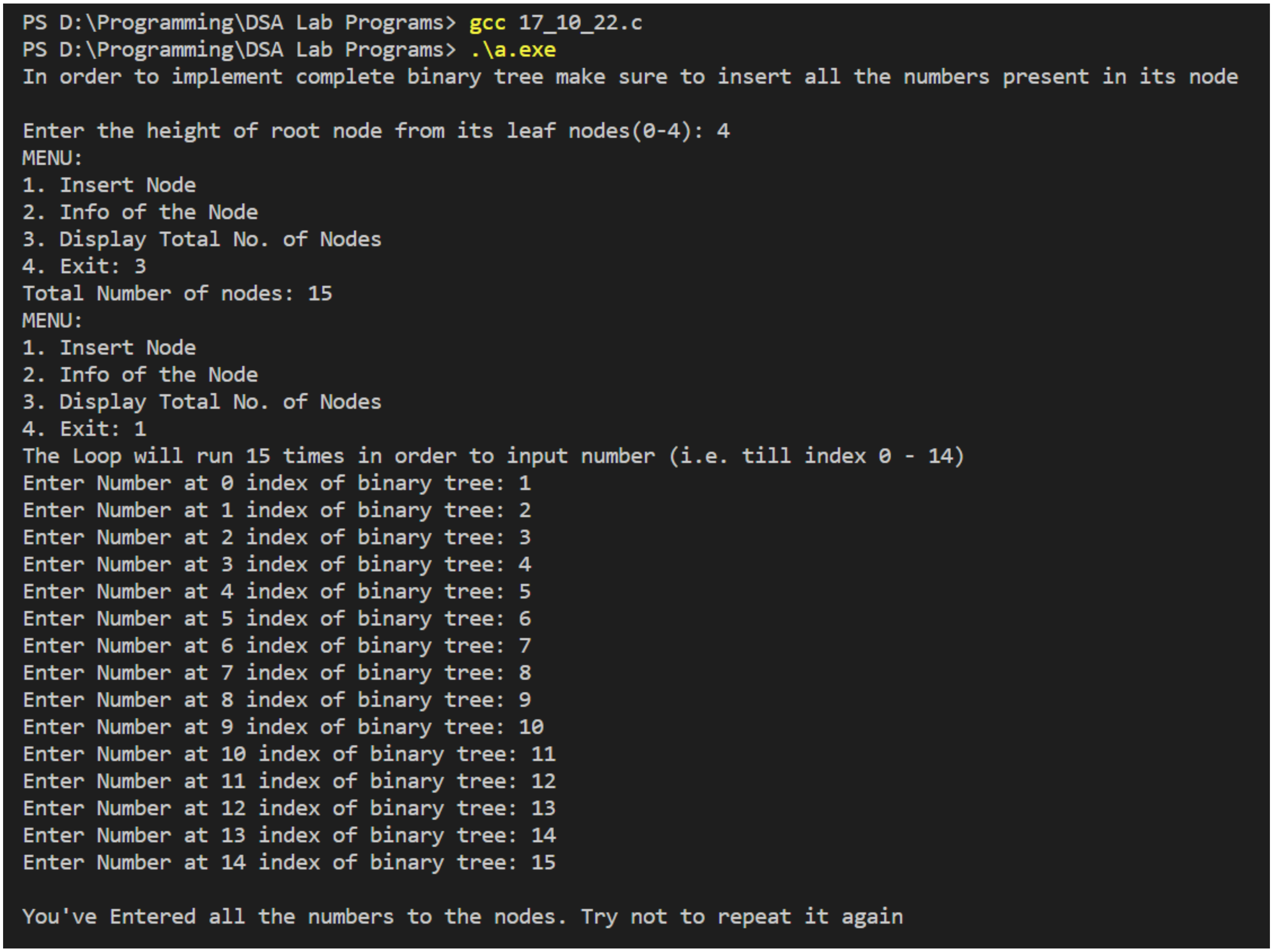
```
1. enQueue
2. deQueue
3. isEmpty
4. isFull
5. queueTraversal
6. Exit: 3
Queue is NOT Empty

MENU:
1. enQueue
2. deQueue
3. isEmpty
4. isFull
5. queueTraversal
6. Exit: 4
Queue is NOT Full

MENU:
1. enQueue
2. deQueue
3. isEmpty
4. isFull
5. queueTraversal
6. Exit: 5
Element at Index 2 is 77
Element at Index 3 is 777

MENU:
1. enQueue
2. deQueue
3. isEmpty
4. isFull
5. queueTraversal
6. Exit: 5
Element at Index 2 is 77
Element at Index 3 is 777
```

PROGRAM: 8

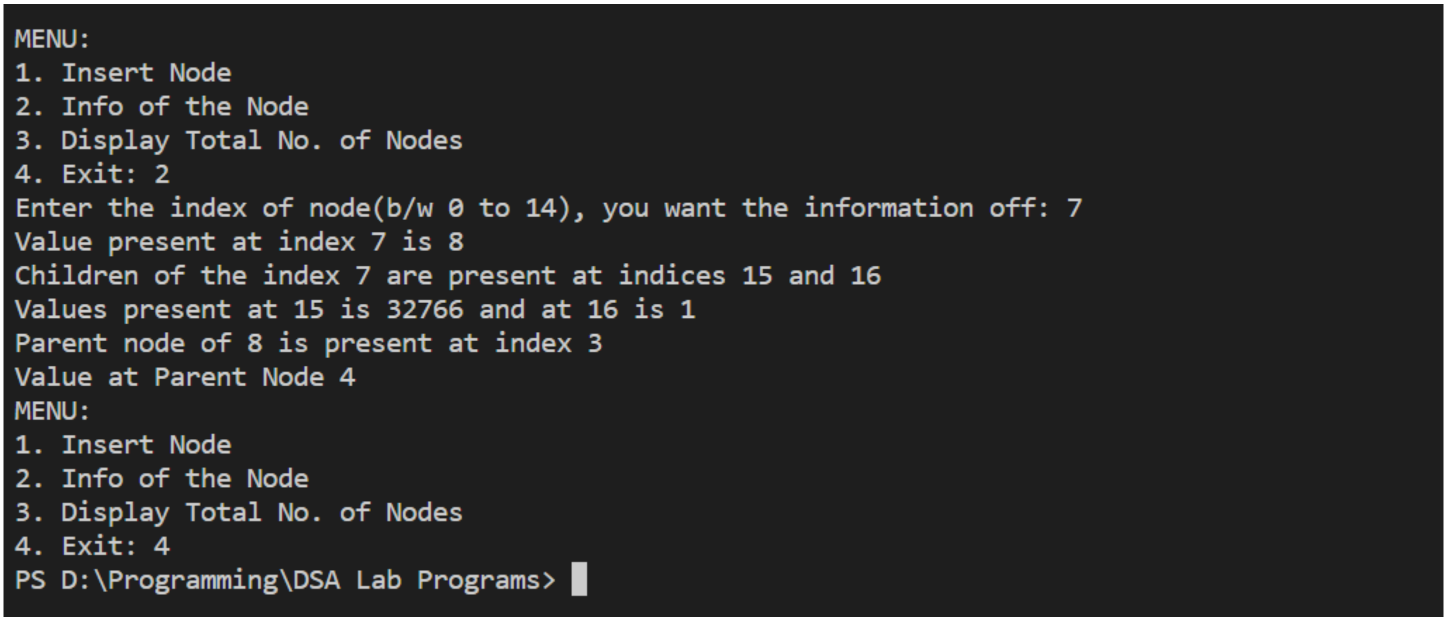


```c
// Complete Binary Tree - Array Representation

#include <stdio.h>

int power(int a, int b)
{
    if (b == 1)
        return a;
    else
        return a * power(a, b - 1);
}

int main()
{
    printf("In order to implement complete binary tree make sure to insert all the numbers present in its node\n");
    printf("\nEnter the height of root node from its leaf nodes(0-4): ");
    int h;
    scanf("%d", &h);

    int tn = power(2, h) - 1;
    int tree[tn];

    while (1)
    {
        printf("MENU:\n1. Insert Node\n2. Info of the Node\n3. Display Total No. of Nodes\n4. Exit: ");
        int choice;
        scanf("%d", &choice);

        switch (choice)
        {
        case 1:
        {
            printf("The Loop will run %d times in order to input number (i.e. till index 0 - %d)\n", tn, tn - 1);
            for (int i = 0; i < tn; i++)
            {
                printf("Enter Number at %d index of binary tree: ", i);
                scanf("%d", &tree[i]);
            }
            printf("\nYou've Entered all the numbers to the nodes. Try not to repeat it again\n\n");
            break;
        }
        case 2:
        {
            printf("Enter the index of node(b/w 0 to %d), you want the information off: ", tn - 1);
            int in;
            scanf("%d", &in);

            printf("Value present at index %d is %d\n", in, tree[in]);
            if (in <= power(2, h - 1))
            {
                printf("Children of the index %d are present at indices %d and %d\n", in, (2
* in) + 1, (2 * in) + 2);
                printf("Values present at %d is %d and at %d is %d\n", (2 * in) + 1, tree[(2
* in) + 1], (2 * in) + 2, tree[(2 * in) + 2]);
            }
            else
                printf("%d index contains a Leaf Node, hence no roots\n", in);

            if (in == 0)
                printf("Since its the Root Node it, it doesn't have an Parent\n");
            else if (in % 2 == 1)
            {
                printf("Parent node of %d is present at index %d\n", tree[in], in / 2);
                printf("Value at Parent Node %d\n", tree[in / 2]);
            }
            else if (in % 2 == 0)
            {
                printf("Parent node of %d is present at index %d\n", tree[in], (in - 1) / 2);
                printf("Value at Parent Node %d\n", tree[(in - 1) / 2]);
            }
            break;
        }
        case 3:
            printf("Total Number of nodes: %d\n", tn);
            break;
        case 4:
            return 0;
            break;
        default:
            printf("Invalid Input!!!\n");
            break;
        }
    }
    return 0;
}
```

Output: 8

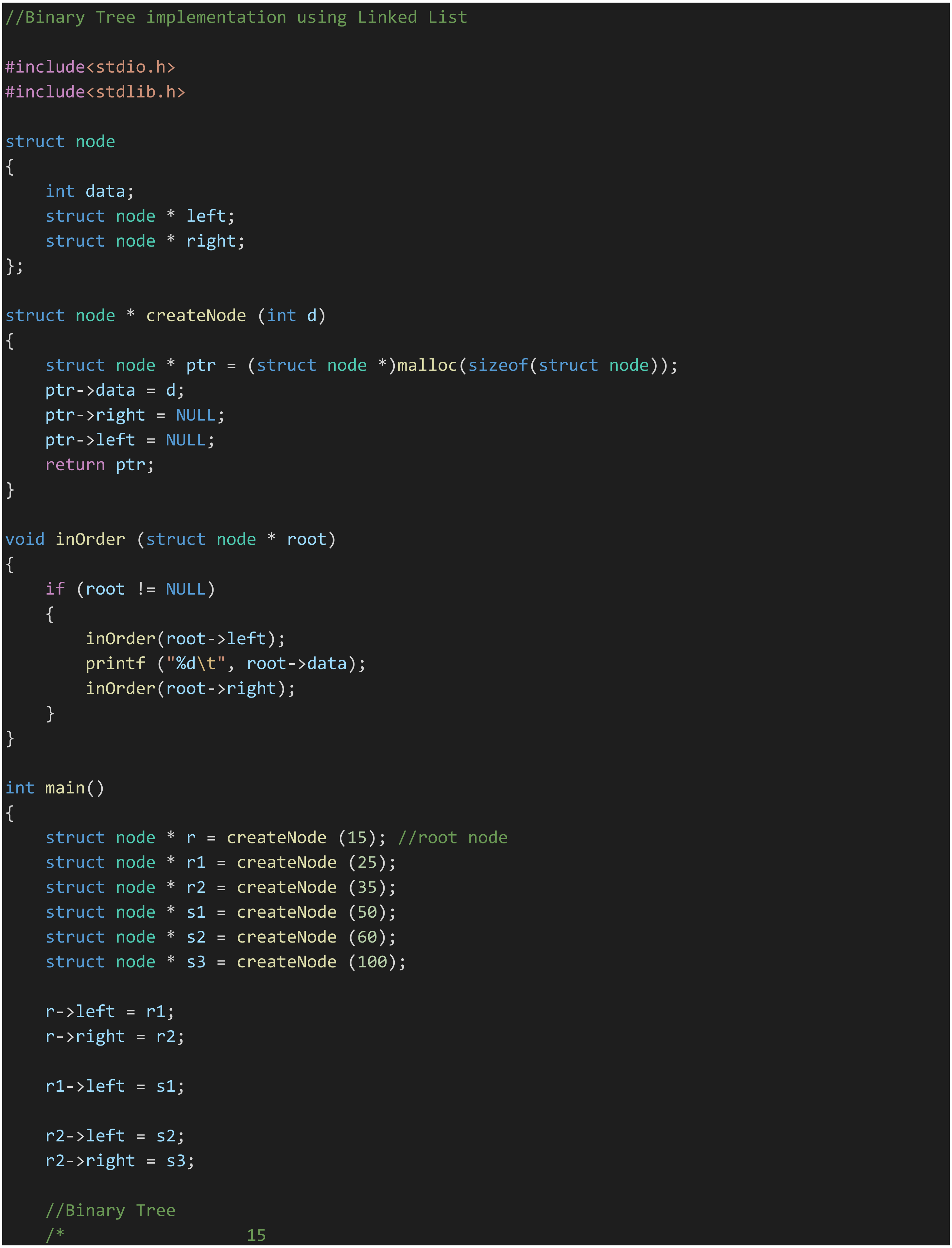


```
PS D:\Programming\DSA Lab Programs> gcc 17_10_22.c
PS D:\Programming\DSA Lab Programs> .\a.exe
In order to implement complete binary tree make sure to insert all the numbers present in its node

Enter the height of root node from its leaf nodes(0-4): 4
MENU:
1. Insert Node
2. Info of the Node
3. Display Total No. of Nodes
4. Exit: 3
Total Number of nodes: 15
MENU:
1. Insert Node
2. Info of the Node
3. Display Total No. of Nodes
4. Exit: 1
The Loop will run 15 times in order to input number (i.e. till index 0 - 14)
Enter Number at 0 index of binary tree: 1
Enter Number at 1 index of binary tree: 2
Enter Number at 2 index of binary tree: 3
Enter Number at 3 index of binary tree: 4
Enter Number at 4 index of binary tree: 5
Enter Number at 5 index of binary tree: 6
Enter Number at 6 index of binary tree: 7
Enter Number at 7 index of binary tree: 8
Enter Number at 8 index of binary tree: 9
Enter Number at 9 index of binary tree: 10
Enter Number at 10 index of binary tree: 11
Enter Number at 11 index of binary tree: 12
Enter Number at 12 index of binary tree: 13
Enter Number at 13 index of binary tree: 14
Enter Number at 14 index of binary tree: 15

You've Entered all the numbers to the nodes. Try not to repeat it again
```



```
MENU:
1. Insert Node
2. Info of the Node
3. Display Total No. of Nodes
4. Exit: 2
Enter the index of node(b/w 0 to 14), you want the information off: 7
Value present at index 7 is 8
Children of the index 7 are present at indices 15 and 16
Values present at 15 is 32766 and at 16 is 1
Parent node of 8 is present at index 3
Value at Parent Node 4
MENU:
1. Insert Node
2. Info of the Node
3. Display Total No. of Nodes
4. Exit: 4
PS D:\Programming\DSA Lab Programs> █
```

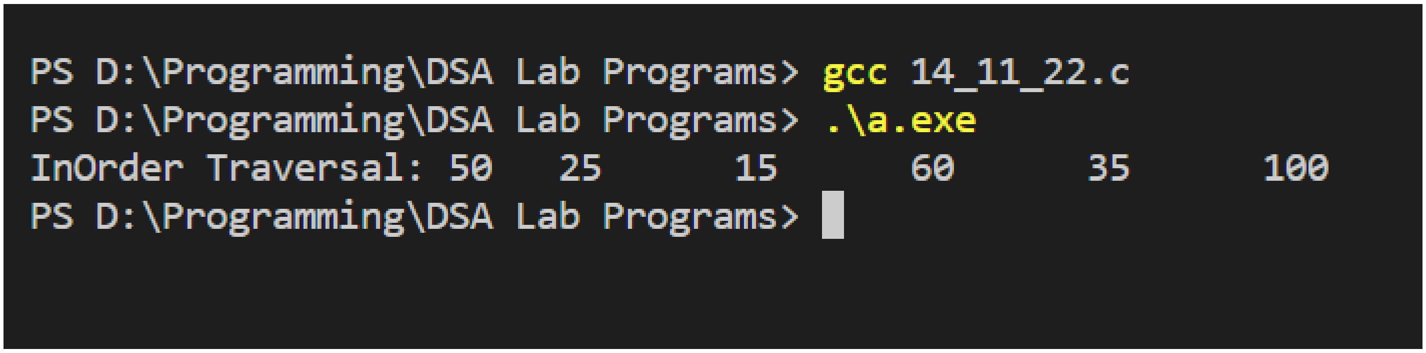
PROGRAM: 9



```c
//Binary Tree implementation using Linked List

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

struct node
{
    int data;
    struct node * left;
    struct node * right;
};

struct node * createNode (int d)
{
    struct node * ptr = (struct node *)malloc(sizeof(struct node));
    ptr->data = d;
    ptr->right = NULL;
    ptr->left = NULL;
    return ptr;
}

void inOrder (struct node * root)
{
    if (root != NULL)
    {
        inOrder(root->left);
        printf ("%d\t", root->data);
        inOrder(root->right);
    }
}

int main()
{
    struct node * r = createNode (15); //root node
    struct node * r1 = createNode (25);
    struct node * r2 = createNode (35);
    struct node * s1 = createNode (50);
    struct node * s2 = createNode (60);
    struct node * s3 = createNode (100);

    r->left = r1;
    r->right = r2;

    r1->left = s1;

    r2->left = s2;
    r2->right = s3;

    //Binary Tree
    /*                  15
```

```
                       /  \
                      25   35
                     /   /  \
                    50   60   100             */

    printf("InOrder Traversal: ");
    inOrder(r);

    return 0;
}
```

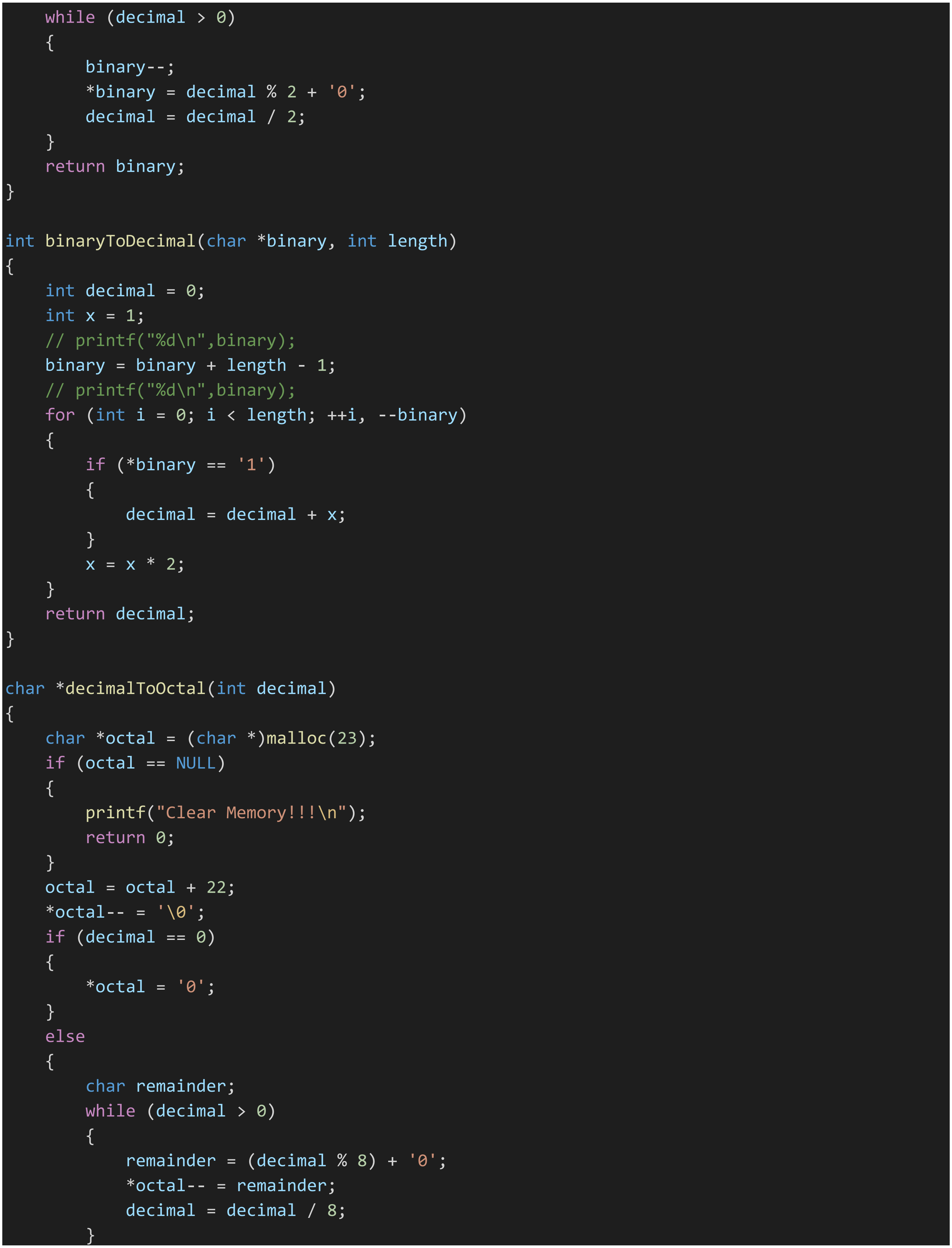
Output: 9



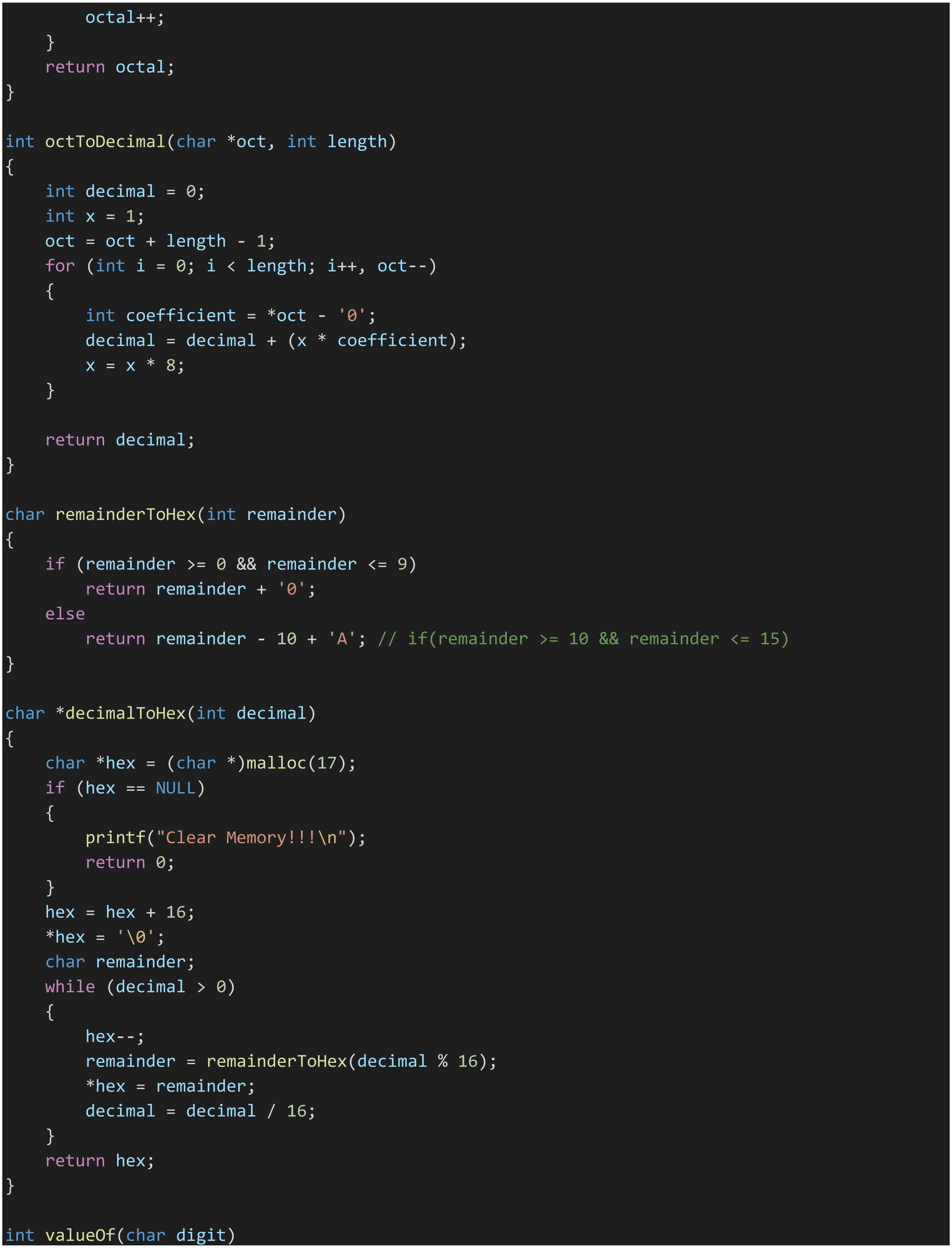
```
PS D:\Programming\DSA Lab Programs> gcc 14_11_22.c
PS D:\Programming\DSA Lab Programs> .\a.exe
InOrder Traversal: 50   25      15      60      35      100
PS D:\Programming\DSA Lab Programs>
```

ASSIGNMENT PROGRAM: 1

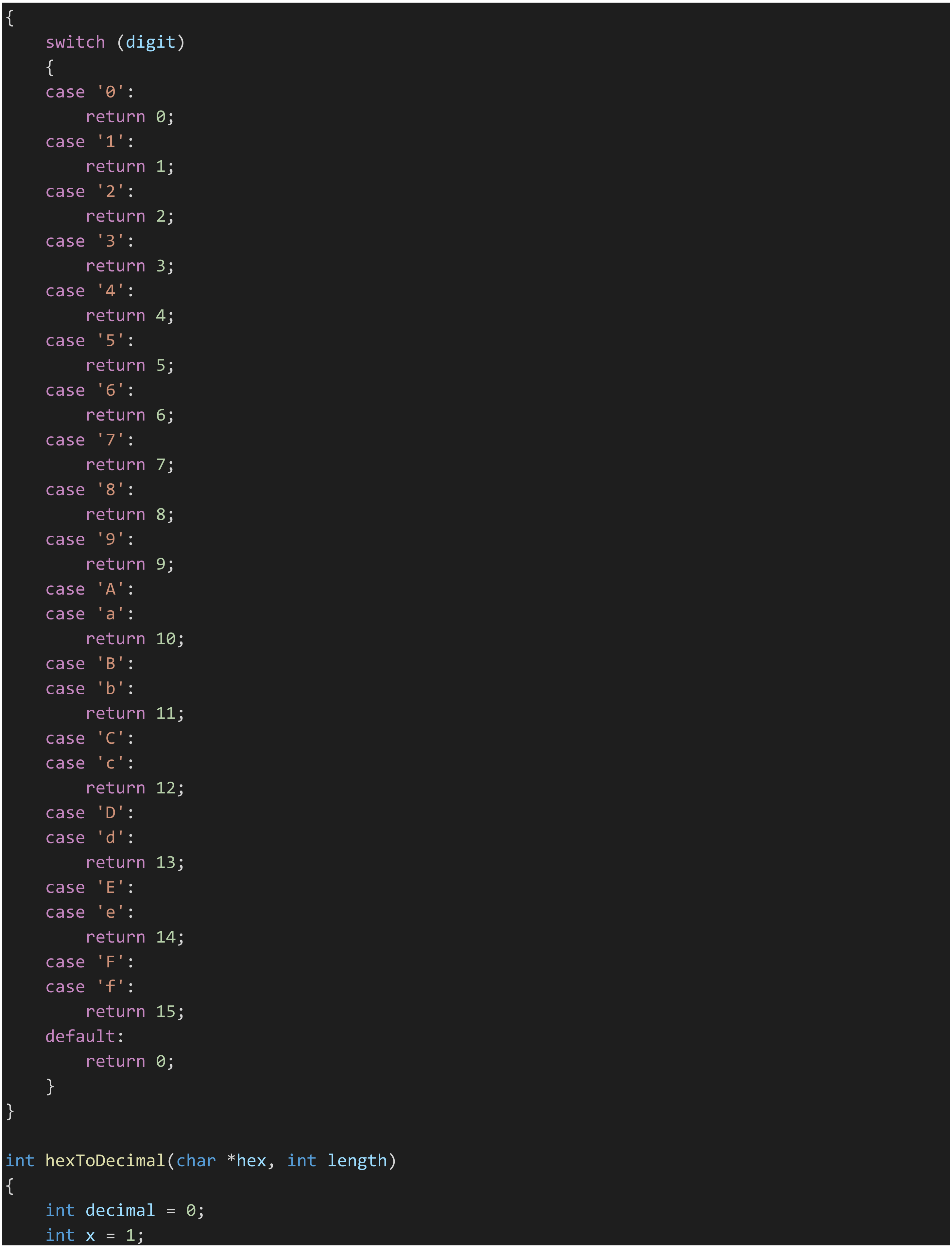
```
// assignment 1
// Number Systems

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

char *decimalToBinary(int decimal)
{
    char *binary = (char *)malloc(sizeof(int) + 1);
    if (binary == NULL)
    {
        printf("Clear Memory!!!\n");
        return 0;
    }
    *binary = '\0';
    // printf("%d/n", &binary);
    if (decimal == 0)
    {
        binary--;
        *binary = '0';
    }
```



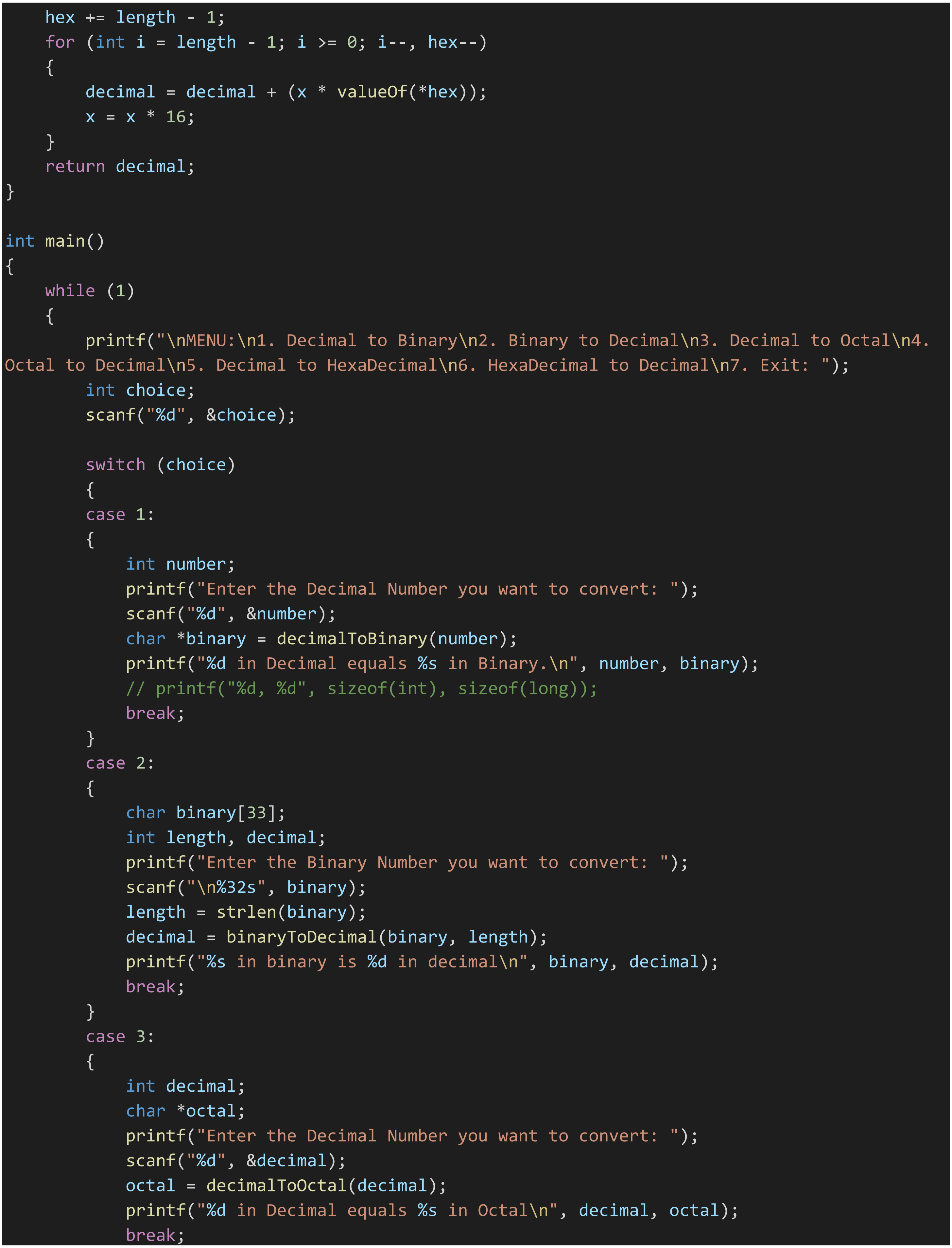
```c
    while (decimal > 0)
    {
        binary--;
        *binary = decimal % 2 + '0';
        decimal = decimal / 2;
    }
    return binary;
}

int binaryToDecimal(char *binary, int length)
{
    int decimal = 0;
    int x = 1;
    // printf("%d\n",binary);
    binary = binary + length - 1;
    // printf("%d\n",binary);
    for (int i = 0; i < length; ++i, --binary)
    {
        if (*binary == '1')
        {
            decimal = decimal + x;
        }
        x = x * 2;
    }
    return decimal;
}

char *decimalToOctal(int decimal)
{
    char *octal = (char *)malloc(23);
    if (octal == NULL)
    {
        printf("Clear Memory!!!\n");
        return 0;
    }
    octal = octal + 22;
    *octal-- = '\0';
    if (decimal == 0)
    {
        *octal = '0';
    }
    else
    {
        char remainder;
        while (decimal > 0)
        {
            remainder = (decimal % 8) + '0';
            *octal-- = remainder;
            decimal = decimal / 8;
        }
```



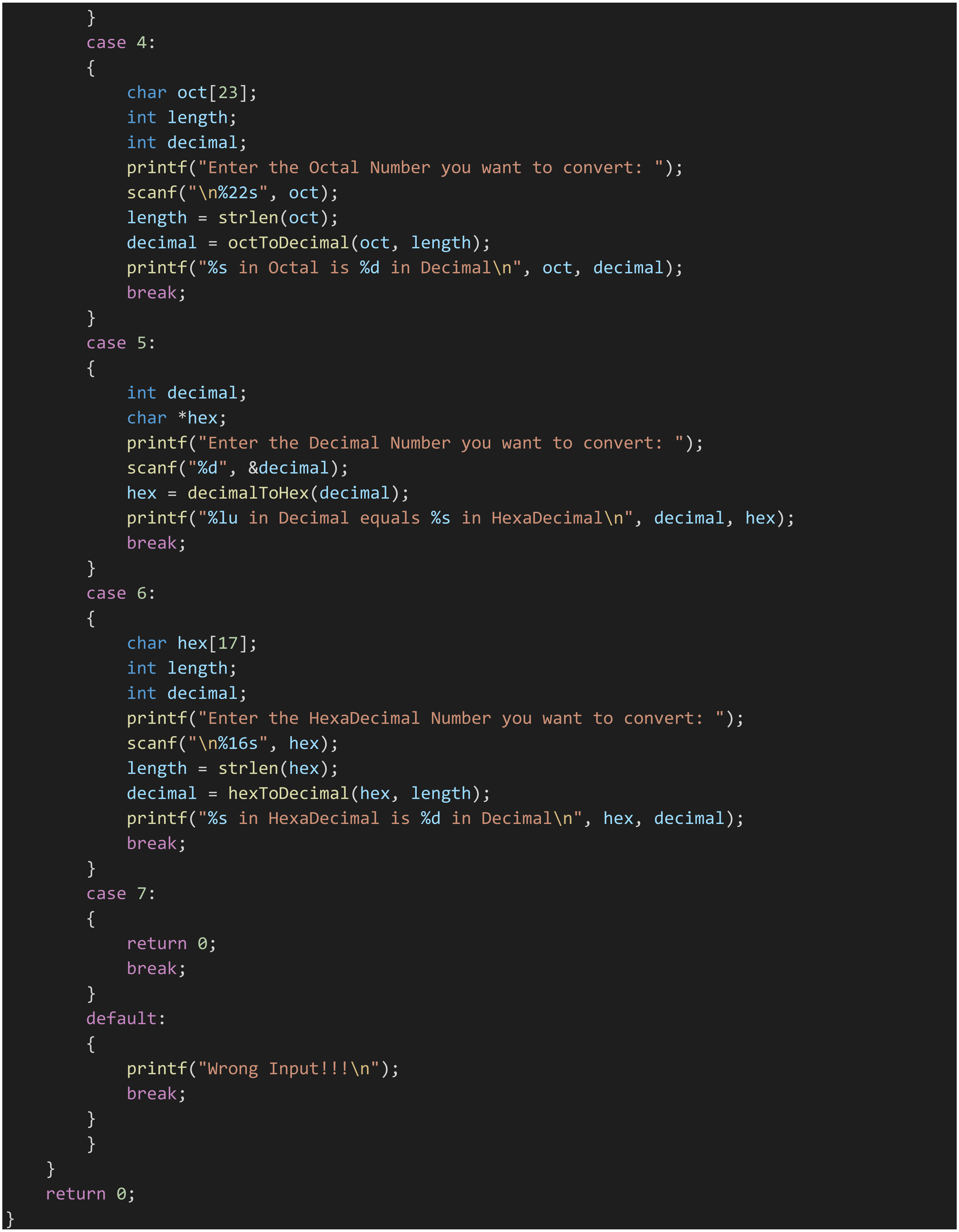
```c
        octal++;
    }
    return octal;
}

int octToDecimal(char *oct, int length)
{
    int decimal = 0;
    int x = 1;
    oct = oct + length - 1;
    for (int i = 0; i < length; i++, oct--)
    {
        int coefficient = *oct - '0';
        decimal = decimal + (x * coefficient);
        x = x * 8;
    }

    return decimal;
}

char remainderToHex(int remainder)
{
    if (remainder >= 0 && remainder <= 9)
        return remainder + '0';
    else
        return remainder - 10 + 'A'; // if(remainder >= 10 && remainder <= 15)
}

char *decimalToHex(int decimal)
{
    char *hex = (char *)malloc(17);
    if (hex == NULL)
    {
        printf("Clear Memory!!!\n");
        return 0;
    }
    hex = hex + 16;
    *hex = '\0';
    char remainder;
    while (decimal > 0)
    {
        hex--;
        remainder = remainderToHex(decimal % 16);
        *hex = remainder;
        decimal = decimal / 16;
    }
    return hex;
}

int valueOf(char digit)
```



```
{
    switch (digit)
    {
    case '0':
        return 0;
    case '1':
        return 1;
    case '2':
        return 2;
    case '3':
        return 3;
    case '4':
        return 4;
    case '5':
        return 5;
    case '6':
        return 6;
    case '7':
        return 7;
    case '8':
        return 8;
    case '9':
        return 9;
    case 'A':
    case 'a':
        return 10;
    case 'B':
    case 'b':
        return 11;
    case 'C':
    case 'c':
        return 12;
    case 'D':
    case 'd':
        return 13;
    case 'E':
    case 'e':
        return 14;
    case 'F':
    case 'f':
        return 15;
    default:
        return 0;
    }
}

int hexToDecimal(char *hex, int length)
{
    int decimal = 0;
    int x = 1;
```

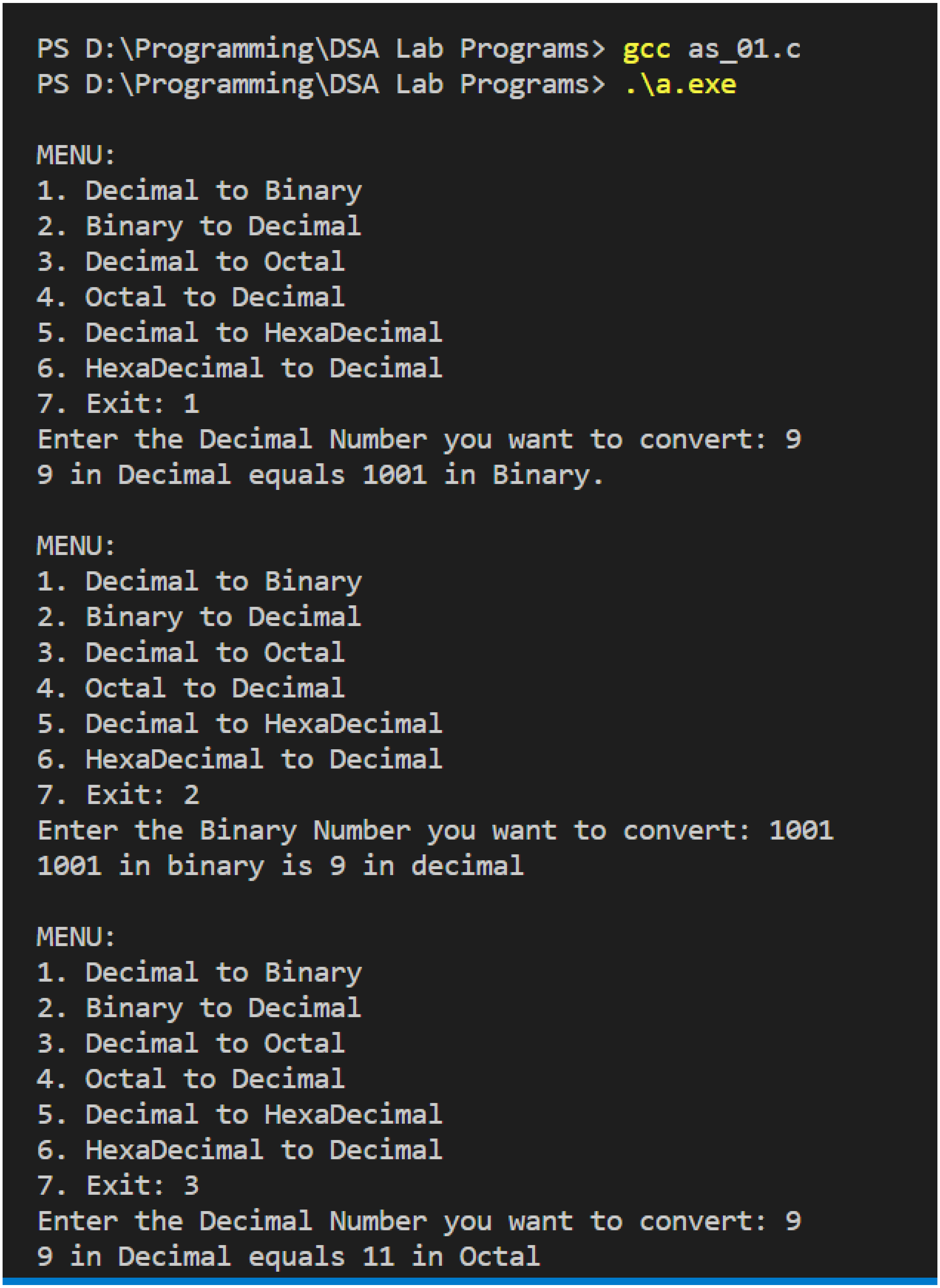


```c
    hex += length - 1;
    for (int i = length - 1; i >= 0; i--, hex--)
    {
        decimal = decimal + (x * valueOf(*hex));
        x = x * 16;
    }
    return decimal;
}

int main()
{
    while (1)
    {
        printf("\nMENU:\n1. Decimal to Binary\n2. Binary to Decimal\n3. Decimal to Octal\n4. Octal to Decimal\n5. Decimal to HexaDecimal\n6. HexaDecimal to Decimal\n7. Exit: ");
        int choice;
        scanf("%d", &choice);

        switch (choice)
        {
        case 1:
        {
            int number;
            printf("Enter the Decimal Number you want to convert: ");
            scanf("%d", &number);
            char *binary = decimalToBinary(number);
            printf("%d in Decimal equals %s in Binary.\n", number, binary);
            // printf("%d, %d", sizeof(int), sizeof(long));
            break;
        }
        case 2:
        {
            char binary[33];
            int length, decimal;
            printf("Enter the Binary Number you want to convert: ");
            scanf("\n%32s", binary);
            length = strlen(binary);
            decimal = binaryToDecimal(binary, length);
            printf("%s in binary is %d in decimal\n", binary, decimal);
            break;
        }
        case 3:
        {
            int decimal;
            char *octal;
            printf("Enter the Decimal Number you want to convert: ");
            scanf("%d", &decimal);
            octal = decimalToOctal(decimal);
            printf("%d in Decimal equals %s in Octal\n", decimal, octal);
            break;
```

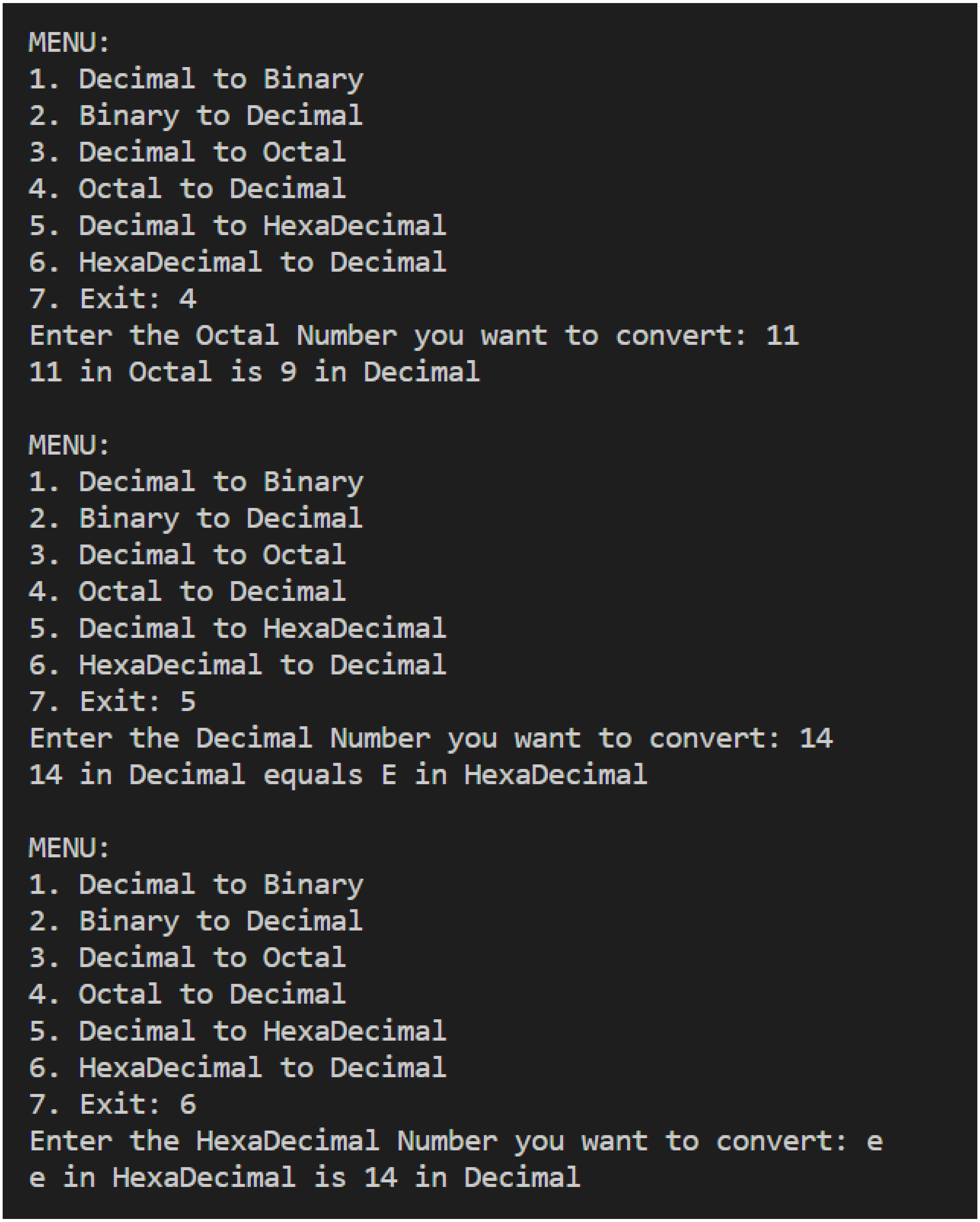


```c
        }
        case 4:
        {
            char oct[23];
            int length;
            int decimal;
            printf("Enter the Octal Number you want to convert: ");
            scanf("\n%22s", oct);
            length = strlen(oct);
            decimal = octToDecimal(oct, length);
            printf("%s in Octal is %d in Decimal\n", oct, decimal);
            break;
        }
        case 5:
        {
            int decimal;
            char *hex;
            printf("Enter the Decimal Number you want to convert: ");
            scanf("%d", &decimal);
            hex = decimalToHex(decimal);
            printf("%lu in Decimal equals %s in HexaDecimal\n", decimal, hex);
            break;
        }
        case 6:
        {
            char hex[17];
            int length;
            int decimal;
            printf("Enter the HexaDecimal Number you want to convert: ");
            scanf("\n%16s", hex);
            length = strlen(hex);
            decimal = hexToDecimal(hex, length);
            printf("%s in HexaDecimal is %d in Decimal\n", hex, decimal);
            break;
        }
        case 7:
        {
            return 0;
            break;
        }
        default:
        {
            printf("Wrong Input!!!\n");
            break;
        }
        }
    }
    return 0;
}
```

Assignment Program Output: 1



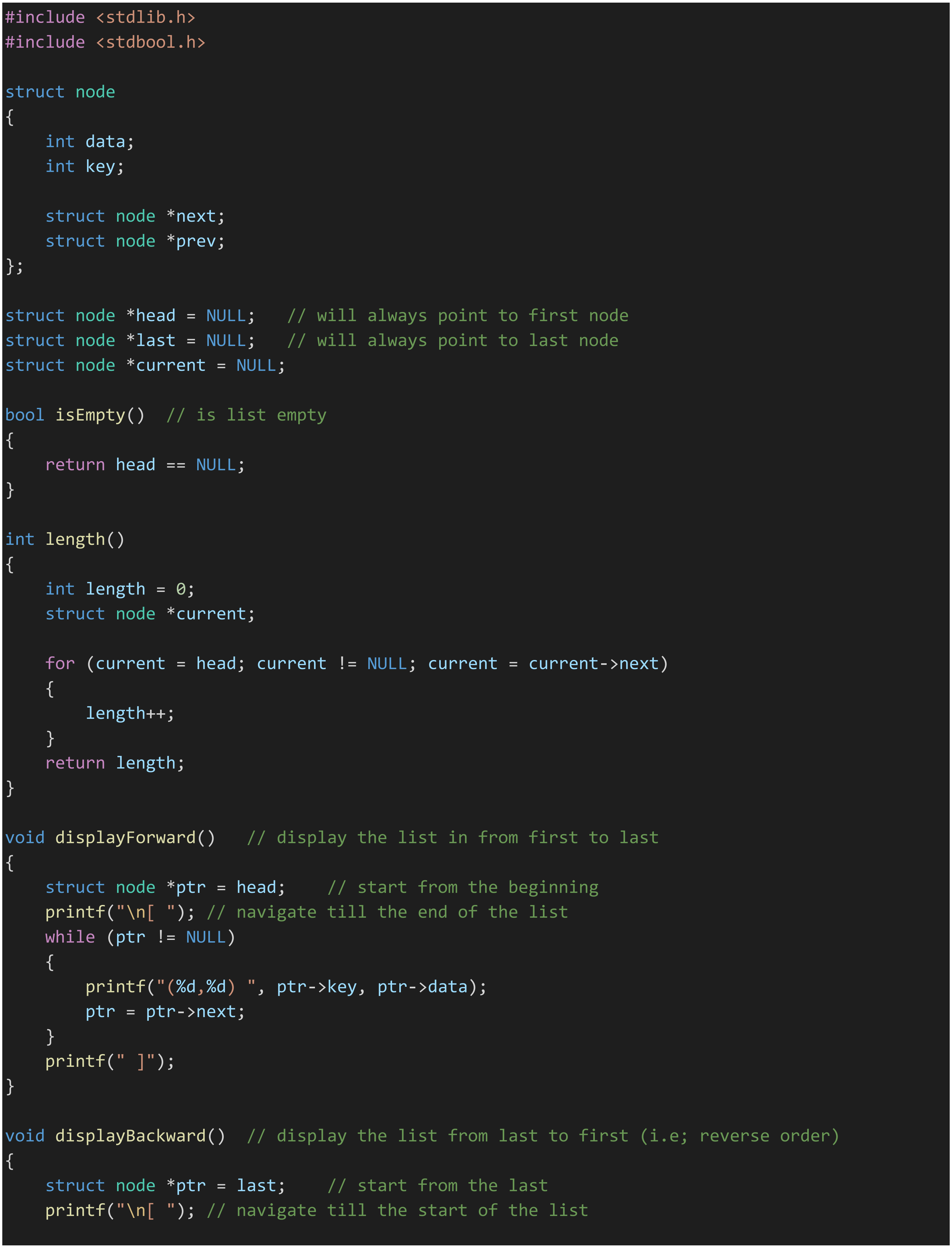
```
PS D:\Programming\DSA Lab Programs> gcc as_01.c
PS D:\Programming\DSA Lab Programs> .\a.exe

MENU:
1. Decimal to Binary
2. Binary to Decimal
3. Decimal to Octal
4. Octal to Decimal
5. Decimal to HexaDecimal
6. HexaDecimal to Decimal
7. Exit: 1
Enter the Decimal Number you want to convert: 9
9 in Decimal equals 1001 in Binary.

MENU:
1. Decimal to Binary
2. Binary to Decimal
3. Decimal to Octal
4. Octal to Decimal
5. Decimal to HexaDecimal
6. HexaDecimal to Decimal
7. Exit: 2
Enter the Binary Number you want to convert: 1001
1001 in binary is 9 in decimal

MENU:
1. Decimal to Binary
2. Binary to Decimal
3. Decimal to Octal
4. Octal to Decimal
5. Decimal to HexaDecimal
6. HexaDecimal to Decimal
7. Exit: 3
Enter the Decimal Number you want to convert: 9
9 in Decimal equals 11 in Octal
```



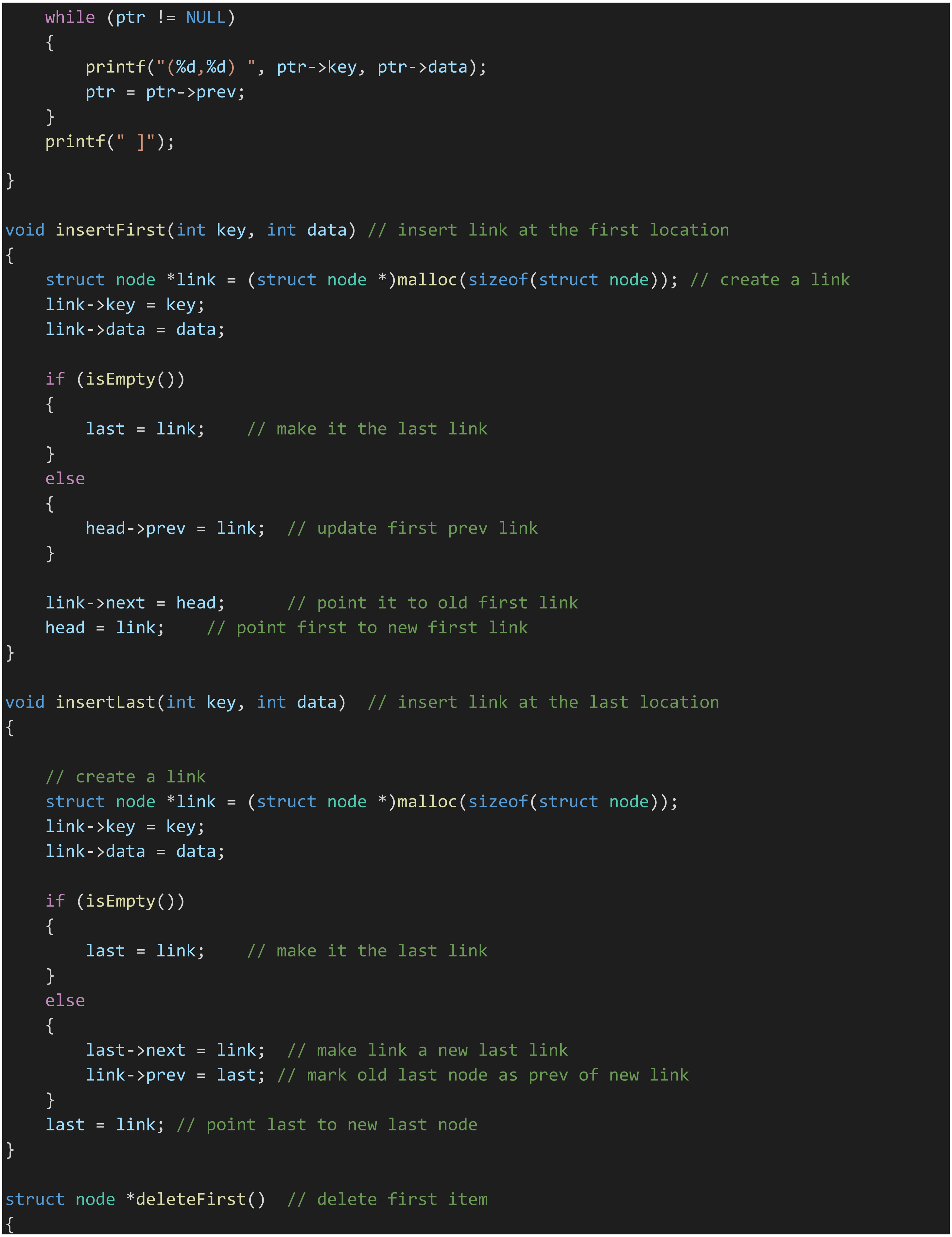
```
MENU:
1. Decimal to Binary
2. Binary to Decimal
3. Decimal to Octal
4. Octal to Decimal
5. Decimal to HexaDecimal
6. HexaDecimal to Decimal
7. Exit: 4
Enter the Octal Number you want to convert: 11
11 in Octal is 9 in Decimal

MENU:
1. Decimal to Binary
2. Binary to Decimal
3. Decimal to Octal
4. Octal to Decimal
5. Decimal to HexaDecimal
6. HexaDecimal to Decimal
7. Exit: 5
Enter the Decimal Number you want to convert: 14
14 in Decimal equals E in HexaDecimal

MENU:
1. Decimal to Binary
2. Binary to Decimal
3. Decimal to Octal
4. Octal to Decimal
5. Decimal to HexaDecimal
6. HexaDecimal to Decimal
7. Exit: 6
Enter the HexaDecimal Number you want to convert: e
e in HexaDecimal is 14 in Decimal
```

ASSIGNMENT PROGRAM: 2

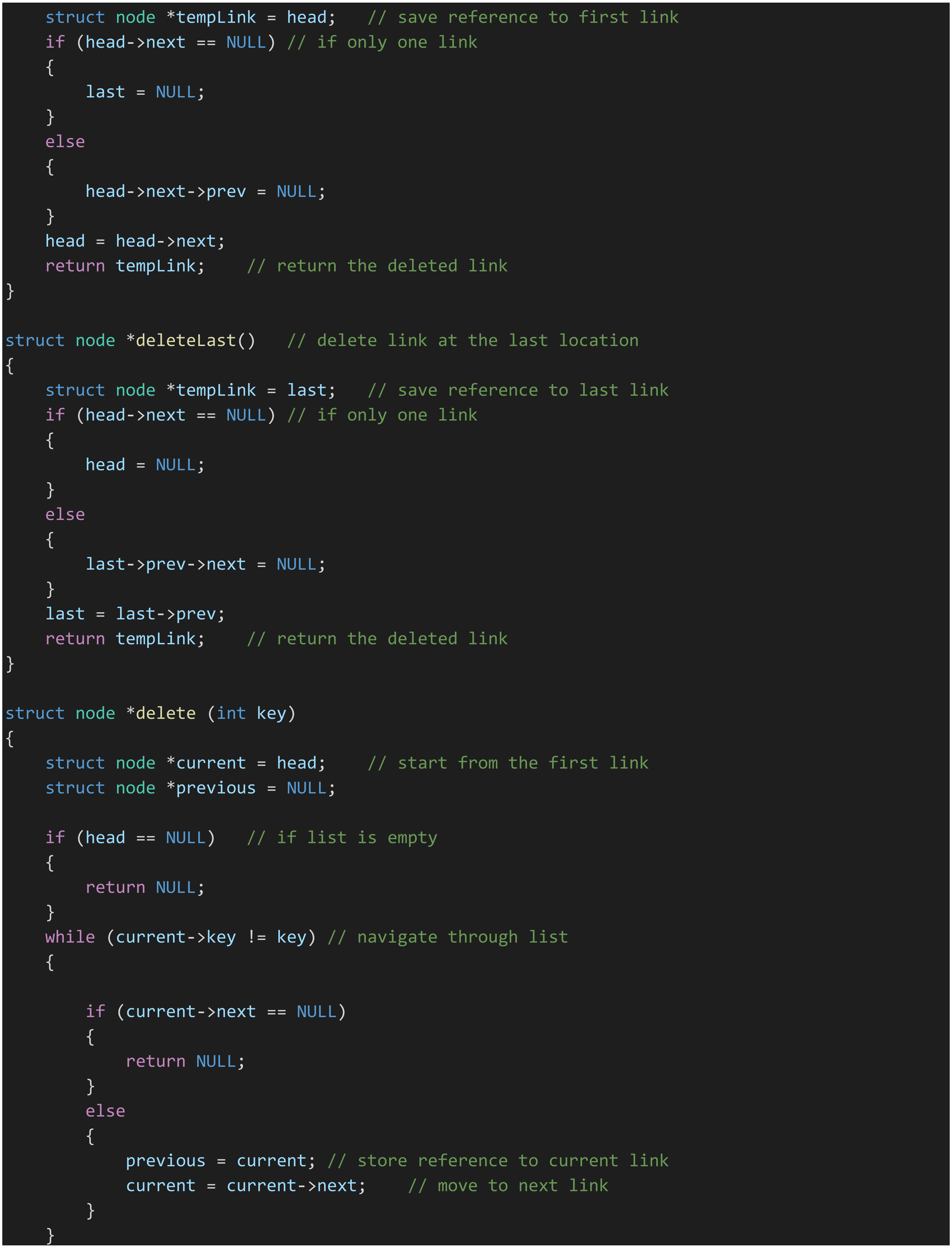
```
// Doubly Linked List

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



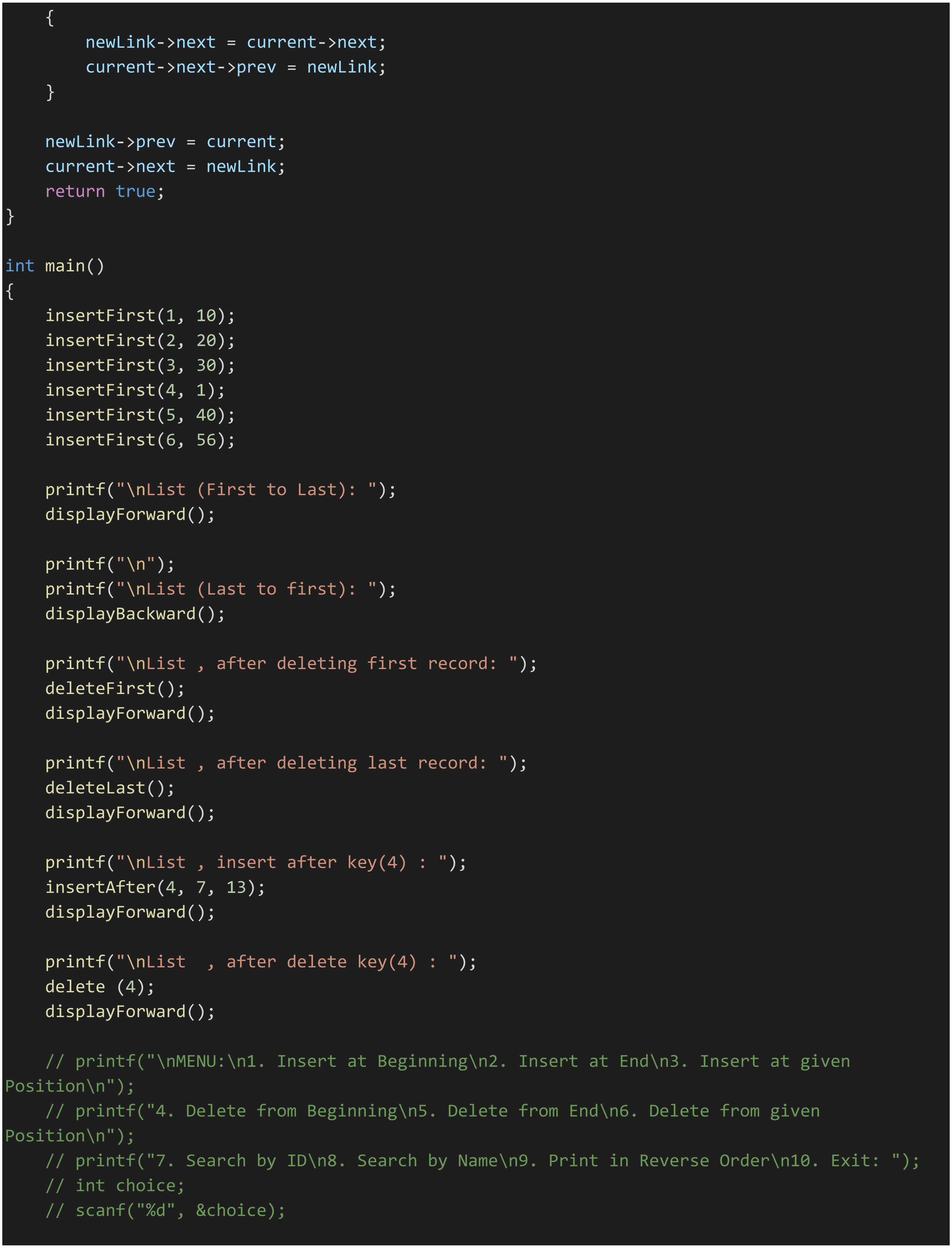
```c
#include <stdlib.h>
#include <stdbool.h>

struct node
{
    int data;
    int key;

    struct node *next;
    struct node *prev;
};

struct node *head = NULL;   // will always point to first node
struct node *last = NULL;   // will always point to last node
struct node *current = NULL;

bool isEmpty()  // is list empty
{
    return head == NULL;
}

int length()
{
    int length = 0;
    struct node *current;

    for (current = head; current != NULL; current = current->next)
    {
        length++;
    }
    return length;
}

void displayForward()   // display the list in from first to last
{
    struct node *ptr = head;    // start from the beginning
    printf("\n[ "); // navigate till the end of the list
    while (ptr != NULL)
    {
        printf("(%d,%d) ", ptr->key, ptr->data);
        ptr = ptr->next;
    }
    printf(" ]");
}

void displayBackward()  // display the list from last to first (i.e; reverse order)
{
    struct node *ptr = last;    // start from the last
    printf("\n[ "); // navigate till the start of the list
```



```c
    while (ptr != NULL)
    {
        printf("(%d,%d) ", ptr->key, ptr->data);
        ptr = ptr->prev;
    }
    printf(" ]");

}

void insertFirst(int key, int data) // insert link at the first location
{
    struct node *link = (struct node *)malloc(sizeof(struct node)); // create a link
    link->key = key;
    link->data = data;

    if (isEmpty())
    {
        last = link;    // make it the last link
    }
    else
    {
        head->prev = link;  // update first prev link
    }

    link->next = head;      // point it to old first link
    head = link;    // point first to new first link
}

void insertLast(int key, int data)  // insert link at the last location
{

    // create a link
    struct node *link = (struct node *)malloc(sizeof(struct node));
    link->key = key;
    link->data = data;

    if (isEmpty())
    {
        last = link;    // make it the last link
    }
    else
    {
        last->next = link;  // make link a new last link
        link->prev = last; // mark old last node as prev of new link
    }
    last = link; // point last to new last node
}

struct node *deleteFirst()  // delete first item
{
```

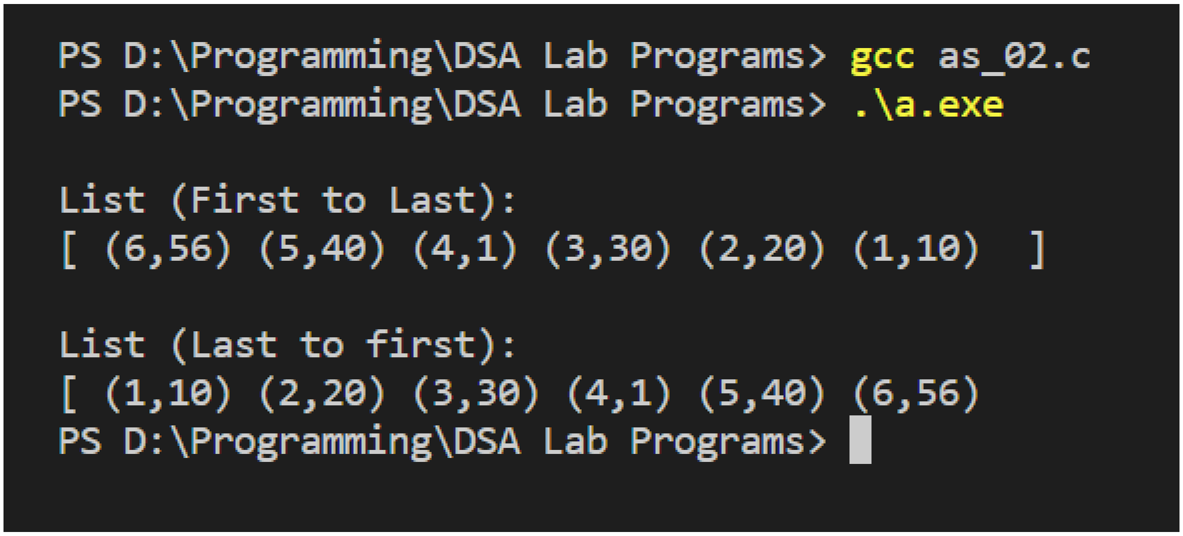


```
    struct node *tempLink = head;   // save reference to first link
    if (head->next == NULL) // if only one link
    {
        last = NULL;
    }
    else
    {
        head->next->prev = NULL;
    }
    head = head->next;
    return tempLink;    // return the deleted link
}

struct node *deleteLast()   // delete link at the last location
{
    struct node *tempLink = last;   // save reference to last link
    if (head->next == NULL) // if only one link
    {
        head = NULL;
    }
    else
    {
        last->prev->next = NULL;
    }
    last = last->prev;
    return tempLink;    // return the deleted link
}

struct node *delete (int key)
{
    struct node *current = head;    // start from the first link
    struct node *previous = NULL;

    if (head == NULL)   // if list is empty
    {
        return NULL;
    }
    while (current->key != key) // navigate through list
    {

        if (current->next == NULL)
        {
            return NULL;
        }
        else
        {
            previous = current; // store reference to current link
            current = current->next;    // move to next link
        }
    }
    if (current == head)
    {
        head = head->next;  // change first to point to next link
    }
    else
    {
        current->prev->next = current->next;    // bypass the current link
    }

    if (current == last)
    {
        last = current->prev;   // change last to point to prev link
    }
    else
    {
        current->next->prev = current->prev;
    }
    return current;
}

bool insertAfter(int key, int newKey, int data)
{
    struct node *current = head;    // start from the first link
    if (head == NULL)   // if list is empty
    {
        return false;
    }
    while (current->key != key) // navigate through list
    {
        if (current->next == NULL)  // if it is last node
        {
            return false;
        }
        else
        {
            current = current->next;    // move to next link
        }
    }

    struct node *newLink = (struct node *)malloc(sizeof(struct node));  // create a link
    newLink->key = newKey;
    newLink->data = data;

    if (current == last)
    {
        newLink->next = NULL;
        last = newLink;
    }
    else
```



```
    {
        newLink->next = current->next;
        current->next->prev = newLink;
    }

    newLink->prev = current;
    current->next = newLink;
    return true;
}

int main()
{
    insertFirst(1, 10);
    insertFirst(2, 20);
    insertFirst(3, 30);
    insertFirst(4, 1);
    insertFirst(5, 40);
    insertFirst(6, 56);

    printf("\nList (First to Last): ");
    displayForward();

    printf("\n");
    printf("\nList (Last to first): ");
    displayBackward();

    printf("\nList , after deleting first record: ");
    deleteFirst();
    displayForward();

    printf("\nList , after deleting last record: ");
    deleteLast();
    displayForward();

    printf("\nList , insert after key(4) : ");
    insertAfter(4, 7, 13);
    displayForward();

    printf("\nList  , after delete key(4) : ");
    delete (4);
    displayForward();

    // printf("\nMENU:\n1. Insert at Beginning\n2. Insert at End\n3. Insert at given 
Position\n");
    // printf("4. Delete from Beginning\n5. Delete from End\n6. Delete from given 
Position\n");
    // printf("7. Search by ID\n8. Search by Name\n9. Print in Reverse Order\n10. Exit: ");
    // int choice;
    // scanf("%d", &choice);
    // switch (choice)
    // {
    // }
    return 0;
}
```

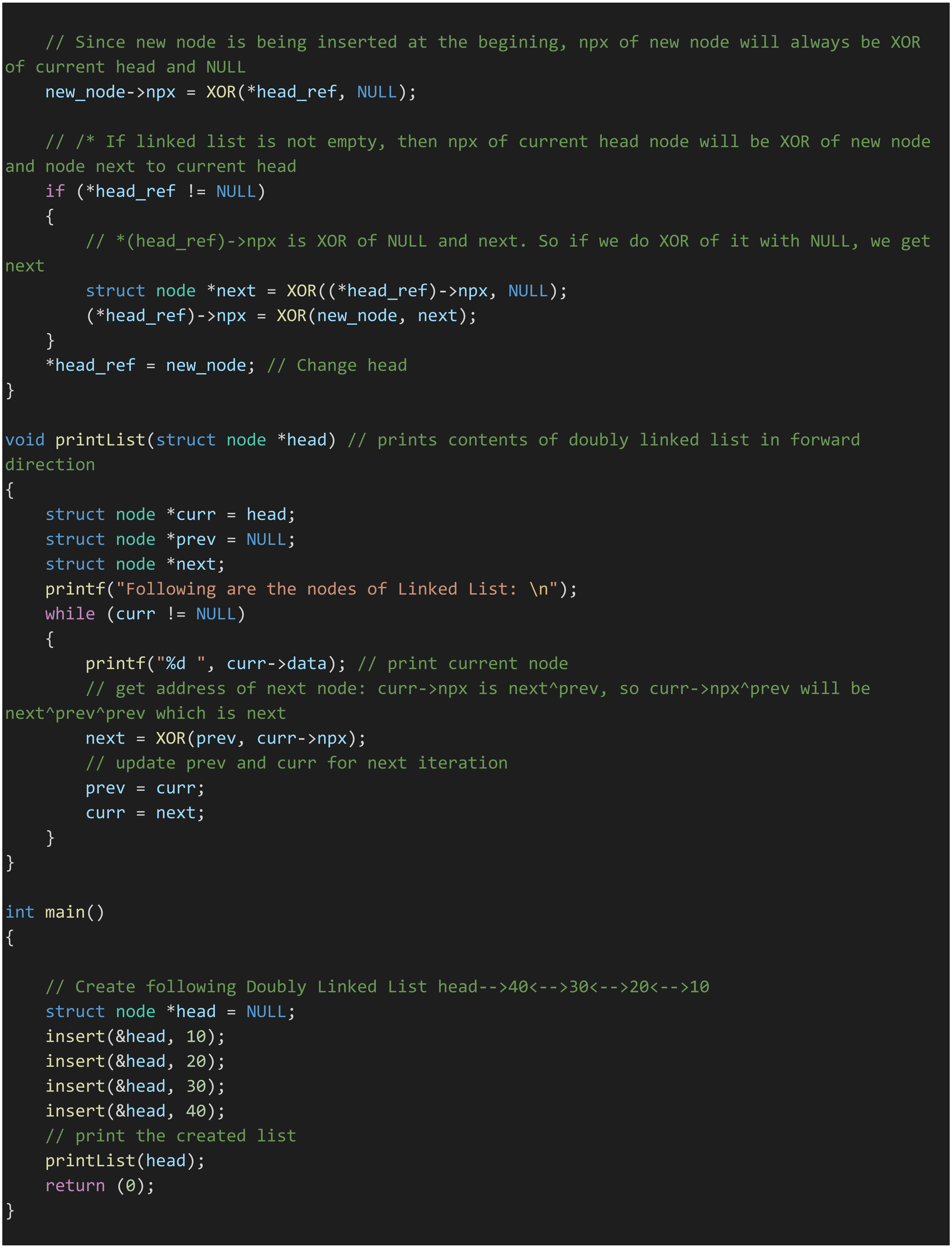
Assignment Program Output: 2



```
PS D:\Programming\DSA Lab Programs> gcc as_02.c
PS D:\Programming\DSA Lab Programs> .\a.exe

List (First to Last):
[ (6,56) (5,40) (4,1) (3,30) (2,20) (1,10)  ]

List (Last to first):
[ (1,10) (2,20) (3,30) (4,1) (5,40) (6,56)
PS D:\Programming\DSA Lab Programs> 
```

ASSIGNMENT PROGRAM: 3

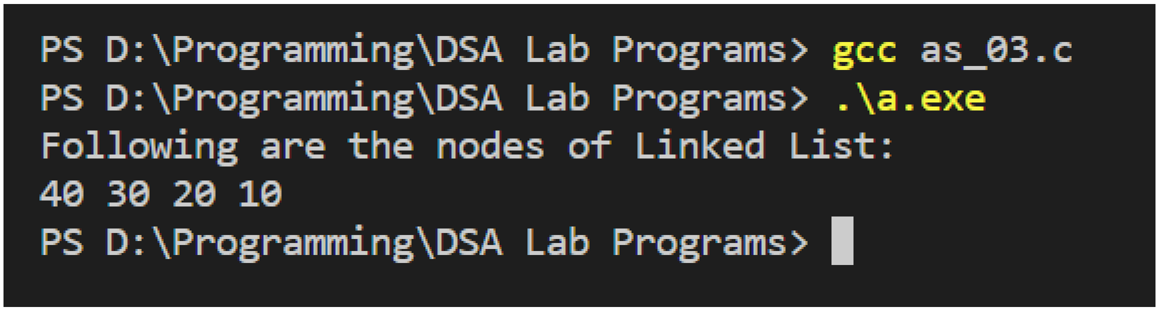
```
// Memory Efficient (XOR) Linked List

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

struct node
{
    int data;
    struct node *npx; /* XOR of next and previous node */
};

struct node *XOR(struct node *a, struct node *b) // returns XORed value of the node addresses
{
    return ((struct node *)((unsigned long long)(a) ^ (unsigned long long)(b)));
}

void insert(struct node **head_ref, int data) // Insert a node at the begining of the XORed
linked list and makes the newly inserted node as head
{
    struct node *new_node = (struct node *)malloc(sizeof(struct node)); // Allocate memory
for new node
    new_node->data = data;
```



```
    // Since new node is being inserted at the begining, npx of new node will always be XOR
of current head and NULL
    new_node->npx = XOR(*head_ref, NULL);

    // /* If linked list is not empty, then npx of current head node will be XOR of new node
and node next to current head
    if (*head_ref != NULL)
    {
        // *(head_ref)->npx is XOR of NULL and next. So if we do XOR of it with NULL, we get
next
        struct node *next = XOR((*head_ref)->npx, NULL);
        (*head_ref)->npx = XOR(new_node, next);
    }
    *head_ref = new_node; // Change head
}

void printList(struct node *head) // prints contents of doubly linked list in forward
direction
{
    struct node *curr = head;
    struct node *prev = NULL;
    struct node *next;
    printf("Following are the nodes of Linked List: \n");
    while (curr != NULL)
    {
        printf("%d ", curr->data); // print current node
        // get address of next node: curr->npx is next^prev, so curr->npx^prev will be
next^prev^prev which is next
        next = XOR(prev, curr->npx);
        // update prev and curr for next iteration
        prev = curr;
        curr = next;
    }
}

int main()
{

    // Create following Doubly Linked List head-->40<-->30<-->20<-->10
    struct node *head = NULL;
    insert(&head, 10);
    insert(&head, 20);
    insert(&head, 30);
    insert(&head, 40);
    // print the created list
    printList(head);
    return (0);
}
```

```
/*Following are the nodes of Linked List:
40 30 20 10*/
```

Assignment Program Output: 3



```
PS D:\Programming\DSA Lab Programs> gcc as_03.c
PS D:\Programming\DSA Lab Programs> .\a.exe
Following are the nodes of Linked List:
40 30 20 10
PS D:\Programming\DSA Lab Programs> 
```