Institute of Computer Technology

B. Tech. Computer Science and Engineering

Semester: III

Sub: Data Structure

Course Code: 2CSE302

Practical Number:1

Practical

1) **Write a Program to display elements of single dimensional array with their memory addresses.**

Code :

```
#include <stdio.h>

int main() {
    int arr[5] = {10, 20, 30, 40, 50};
    int *ptr = arr;
    for (int i = 0; i < 5; i++) {
        printf("Element at index %d: %d, Address: %u\n ", i, *(ptr + i), (ptr + i));
    }
    return 0;
}
```

Output :

```
Element at index 0: 10, Address: 2933010736
Element at index 1: 20, Address: 2933010740
Element at index 2: 30, Address: 2933010744
Element at index 3: 40, Address: 2933010748
Element at index 4: 50, Address: 2933010752
```

2) **Write a Program to display elements of Two-dimensional array with their memory addresses.**

Code :

```
#include <stdio.h>

int main() {
    int array_2d[2][3] = {{1, 2, 3}, {4, 5, 6}};

    printf("Address of the entire 2D array: %u\n", (void*)array_2d);
    for (int i = 0; i < 2; i++) {
        printf("Address of row %d: %u\n", i, (void*)array_2d[i]);
    }

    for (int i = 0; i < 2; i++) {
        for (int j = 0; j < 3; j++) {
            printf("Address of element at (%d, %d): %u\n", i, j,
(void*)&array_2d[i][j]);
        }
    }

    return 0;
}
```

Output :

```
Address of the entire 2D array: 1196746528
Address of row 0: 1196746528
Address of row 1: 1196746540
Address of element at (0, 0): 1196746528
Address of element at (0, 1): 1196746532
Address of element at (0, 2): 1196746536
Address of element at (1, 0): 1196746540
Address of element at (1, 1): 1196746544
Address of element at (1, 2): 1196746548
```

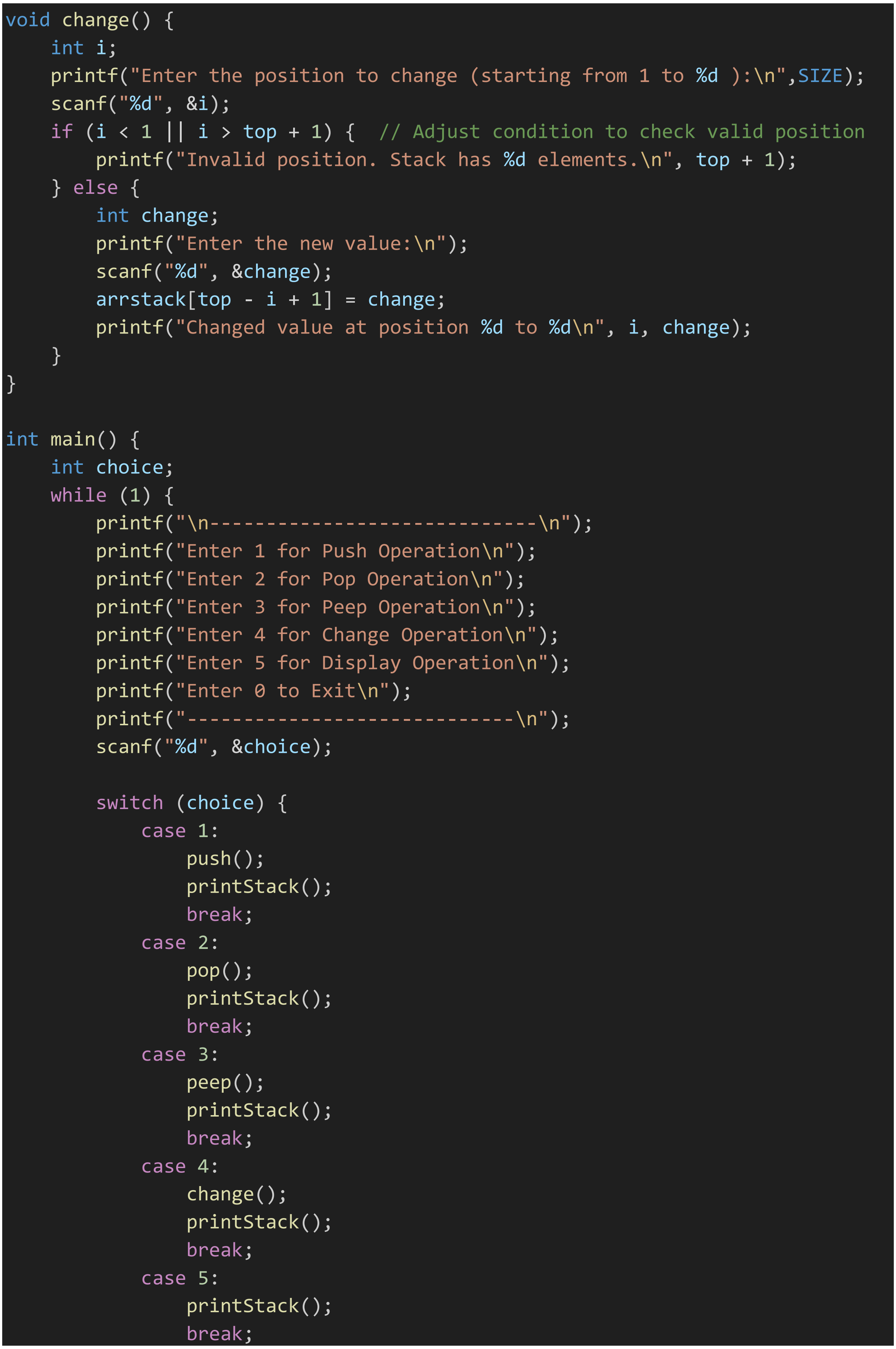
3) Write a program to implement the concept of Stack and perform following operations on Stack.

- Push
- Pop
- Peep
- Change
- Display

Code :

```
#include <stdio.h>

#define SIZE 4

int arrstack[SIZE];
int top = -1;
int value;
```

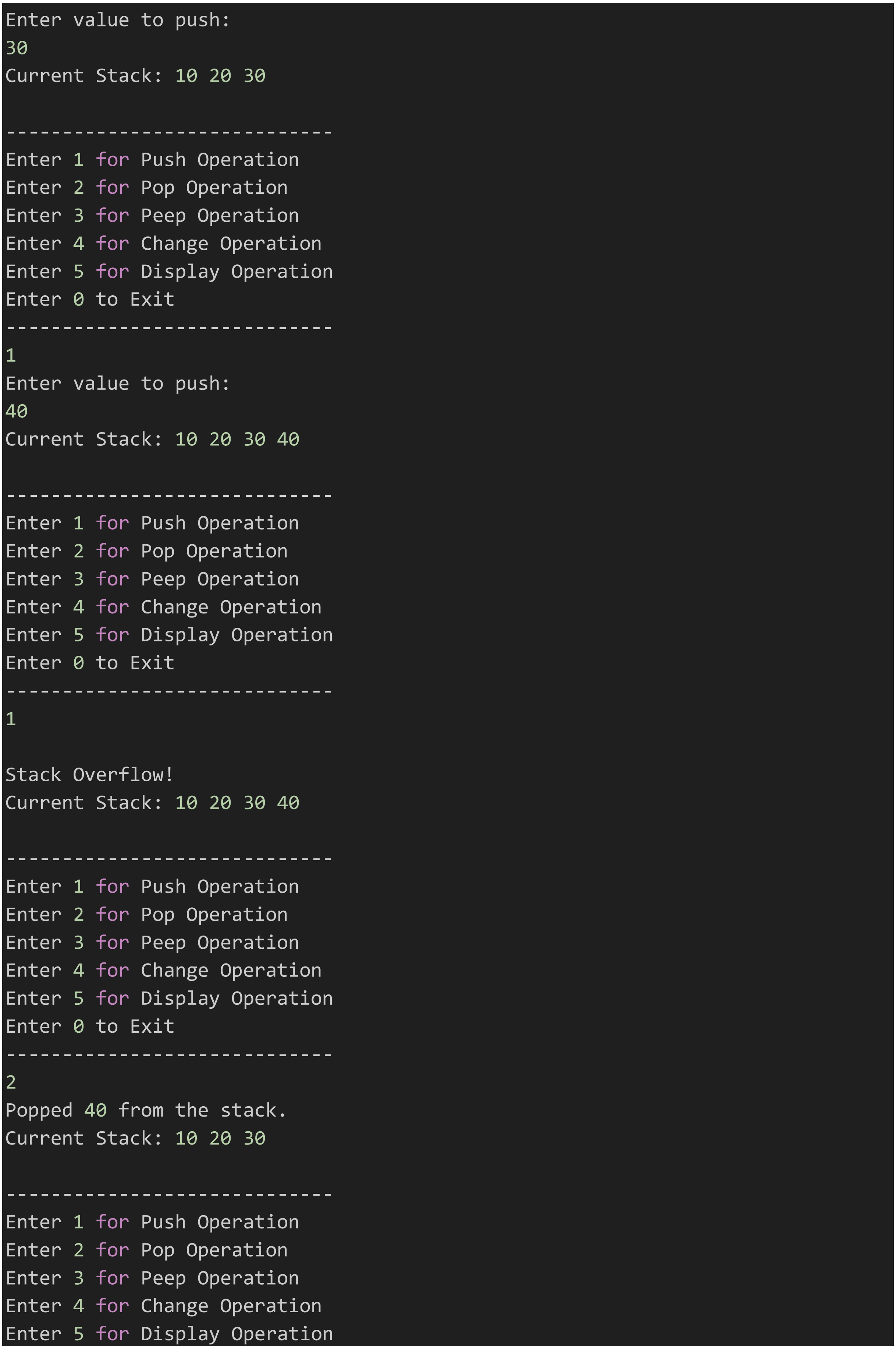
```c
// Function to print the stack
void printStack() {
    if (top == -1) {
        printf("Stack is empty\n");
    } else {
        printf("Current Stack: ");
        for (int i = 0; i <= top; i++) {
            printf("%d ", arrstack[i]);
        }
        printf("\n");
    }
}

// Function to push an element onto the stack
void push() {
    if (top >= SIZE - 1) {
        printf("\nStack Overflow!\n");
    } else {
        printf("Enter value to push:\n");
        scanf("%d", &value);
        top++;
        arrstack[top] = value;
    }
}

void pop() {
    if (top == -1) {
        printf("Stack is empty\n");
    } else {
        value = arrstack[top];
        top = top - 1;
        printf("Popped %d from the stack.\n", value);
    }
}

void peep() {
    int i;
    printf("Enter the position to peep (starting from 1 to %d):\n",SIZE);
    scanf("%d", &i);
    if (i < 1 || i > top + 1) {  // Adjust condition to check valid position
        printf("Invalid position. Stack has %d elements.\n", top + 1);
    } else {
        value = arrstack[top - i + 1];
        printf("Peeped value: %d\n", value);
    }
}
```



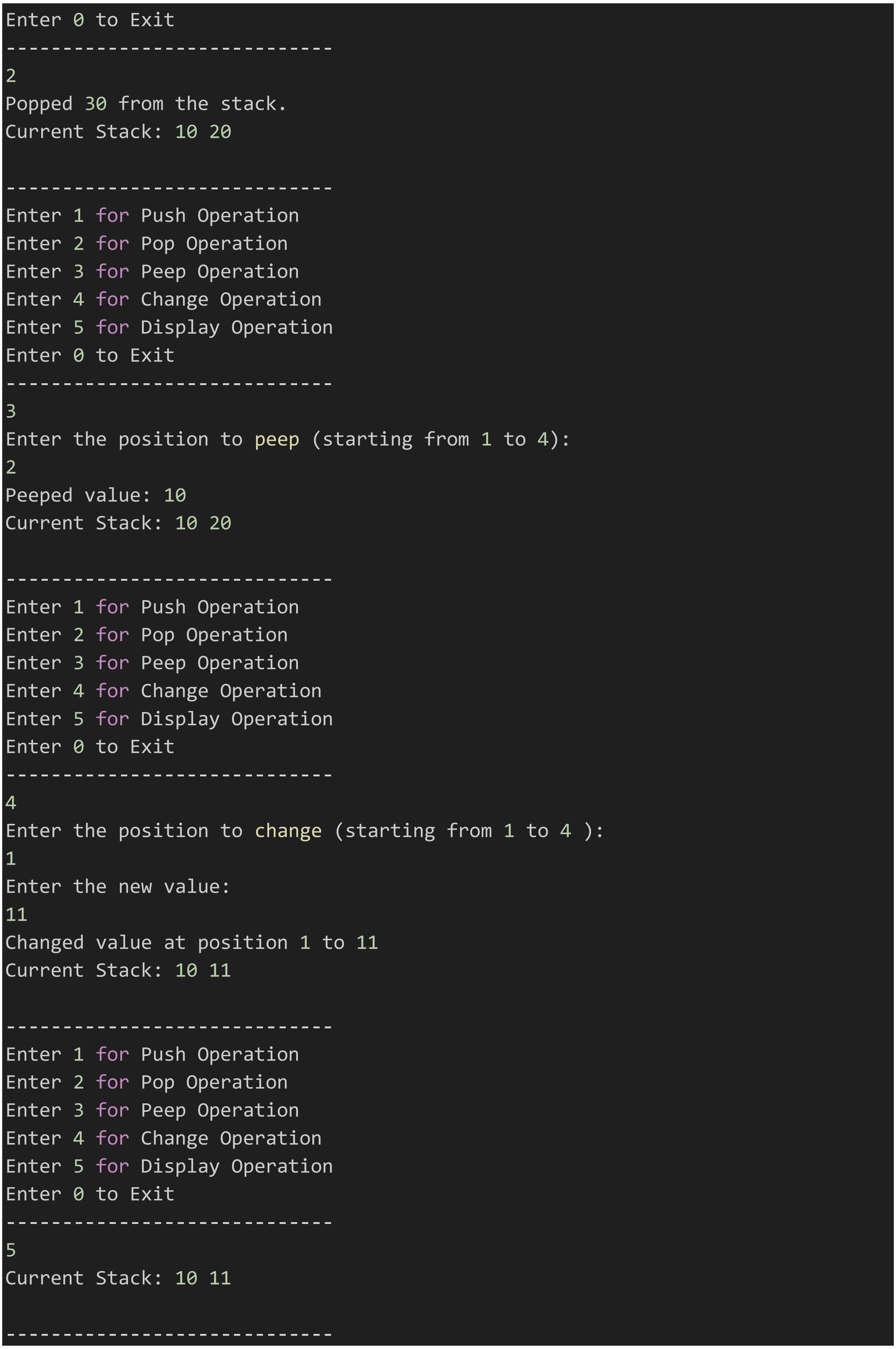
```
void change() {
    int i;
    printf("Enter the position to change (starting from 1 to %d ):\n",SIZE);
    scanf("%d", &i);
    if (i < 1 || i > top + 1) {  // Adjust condition to check valid position
        printf("Invalid position. Stack has %d elements.\n", top + 1);
    } else {
        int change;
        printf("Enter the new value:\n");
        scanf("%d", &change);
        arrstack[top - i + 1] = change;
        printf("Changed value at position %d to %d\n", i, change);
    }
}

int main() {
    int choice;
    while (1) {
        printf("\n----------------------------\n");
        printf("Enter 1 for Push Operation\n");
        printf("Enter 2 for Pop Operation\n");
        printf("Enter 3 for Peep Operation\n");
        printf("Enter 4 for Change Operation\n");
        printf("Enter 5 for Display Operation\n");
        printf("Enter 0 to Exit\n");
        printf("----------------------------\n");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                push();
                printStack();
                break;
            case 2:
                pop();
                printStack();
                break;
            case 3:
                peep();
                printStack();
                break;
            case 4:
                change();
                printStack();
                break;
            case 5:
                printStack();
                break;
            case 0:
                printf("Exiting...\n");
                return 0;
            default:
                printf("Invalid choice, please try again.\n");
        }
    }

    return 0;
}
```

Output :

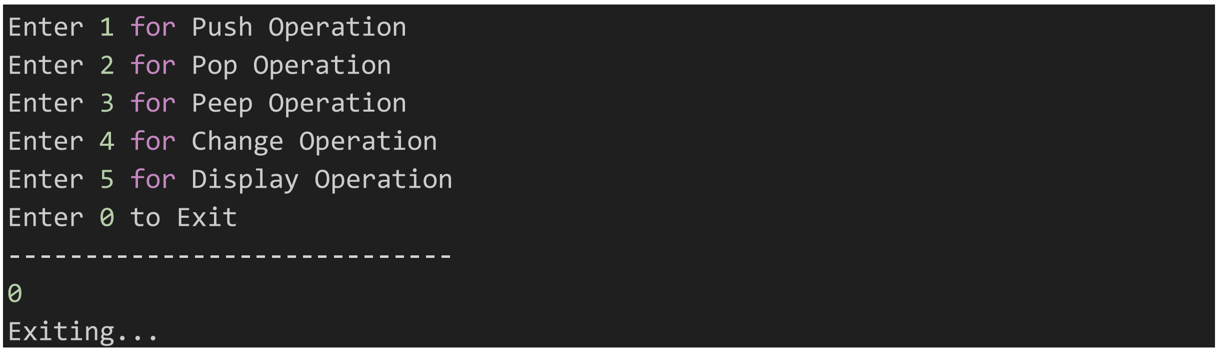
```
-----------------------------
Enter 1 for Push Operation
Enter 2 for Pop Operation
Enter 3 for Peep Operation
Enter 4 for Change Operation
Enter 5 for Display Operation
Enter 0 to Exit
-----------------------------
1
Enter value to push:
10
Current Stack: 10

-----------------------------
Enter 1 for Push Operation
Enter 2 for Pop Operation
Enter 3 for Peep Operation
Enter 4 for Change Operation
Enter 5 for Display Operation
Enter 0 to Exit
-----------------------------
1
Enter value to push:
20
Current Stack: 10 20

-----------------------------
Enter 1 for Push Operation
Enter 2 for Pop Operation
Enter 3 for Peep Operation
Enter 4 for Change Operation
Enter 5 for Display Operation
Enter 0 to Exit
-----------------------------
1
```



```
Enter value to push:
30
Current Stack: 10 20 30

-----------------------------
Enter 1 for Push Operation
Enter 2 for Pop Operation
Enter 3 for Peep Operation
Enter 4 for Change Operation
Enter 5 for Display Operation
Enter 0 to Exit
-----------------------------
1
Enter value to push:
40
Current Stack: 10 20 30 40

-----------------------------
Enter 1 for Push Operation
Enter 2 for Pop Operation
Enter 3 for Peep Operation
Enter 4 for Change Operation
Enter 5 for Display Operation
Enter 0 to Exit
-----------------------------
1

Stack Overflow!
Current Stack: 10 20 30 40

-----------------------------
Enter 1 for Push Operation
Enter 2 for Pop Operation
Enter 3 for Peep Operation
Enter 4 for Change Operation
Enter 5 for Display Operation
Enter 0 to Exit
-----------------------------
2
Popped 40 from the stack.
Current Stack: 10 20 30

-----------------------------
Enter 1 for Push Operation
Enter 2 for Pop Operation
Enter 3 for Peep Operation
Enter 4 for Change Operation
Enter 5 for Display Operation
```



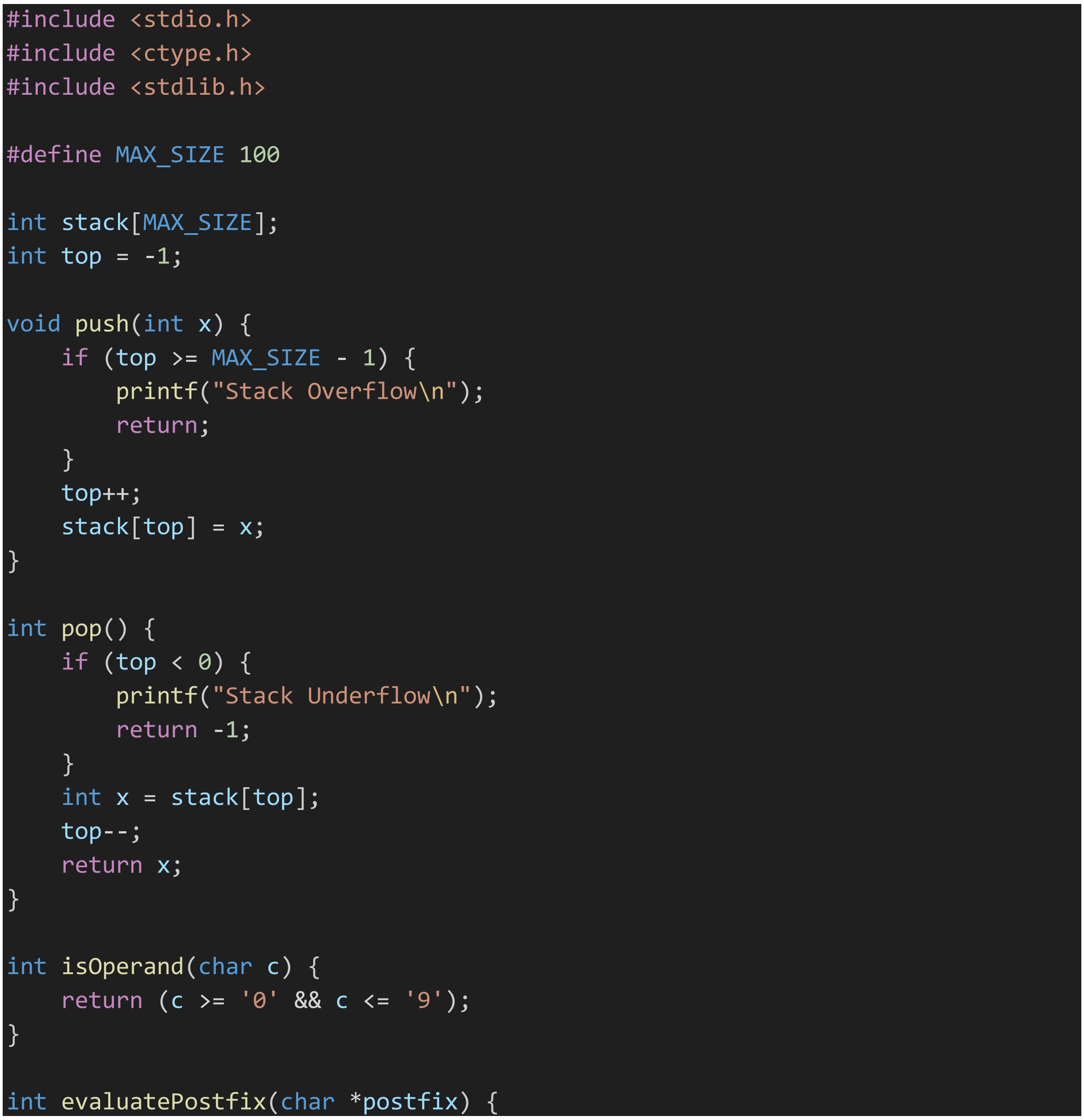
```
Enter 0 to Exit
-----------------------------
2
Popped 30 from the stack.
Current Stack: 10 20

-----------------------------
Enter 1 for Push Operation
Enter 2 for Pop Operation
Enter 3 for Peep Operation
Enter 4 for Change Operation
Enter 5 for Display Operation
Enter 0 to Exit
-----------------------------
3
Enter the position to peep (starting from 1 to 4):
2
Peeped value: 10
Current Stack: 10 20

-----------------------------
Enter 1 for Push Operation
Enter 2 for Pop Operation
Enter 3 for Peep Operation
Enter 4 for Change Operation
Enter 5 for Display Operation
Enter 0 to Exit
-----------------------------
4
Enter the position to change (starting from 1 to 4 ):
1
Enter the new value:
11
Changed value at position 1 to 11
Current Stack: 10 11

-----------------------------
Enter 1 for Push Operation
Enter 2 for Pop Operation
Enter 3 for Peep Operation
Enter 4 for Change Operation
Enter 5 for Display Operation
Enter 0 to Exit
-----------------------------
5
Current Stack: 10 11

-----------------------------
```



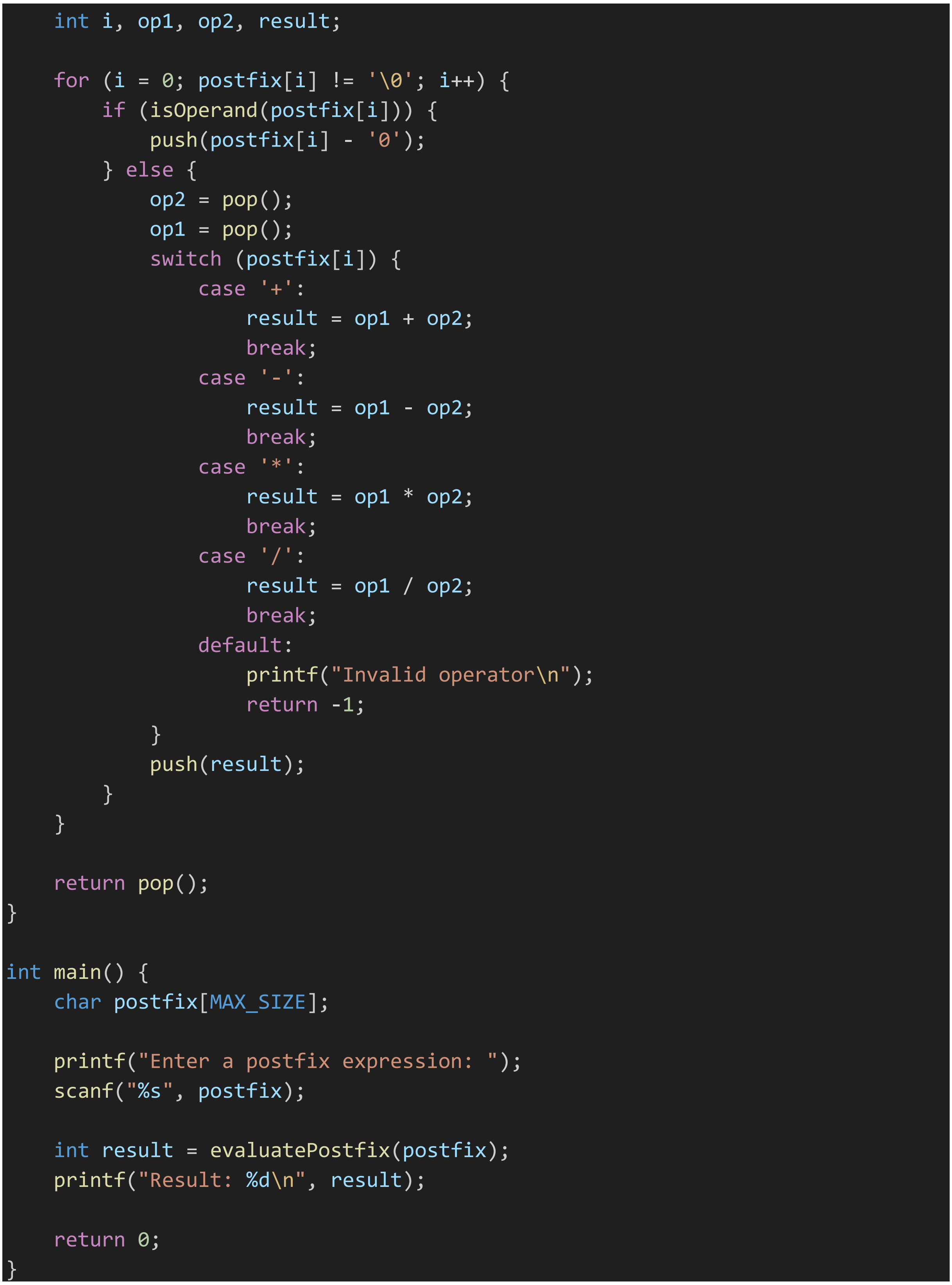
```
Enter 1 for Push Operation
Enter 2 for Pop Operation
Enter 3 for Peep Operation
Enter 4 for Change Operation
Enter 5 for Display Operation
Enter 0 to Exit
-----------------------------
0
Exiting...
```

4) Write a program to evaluate a postfix expression using stack.

Code :

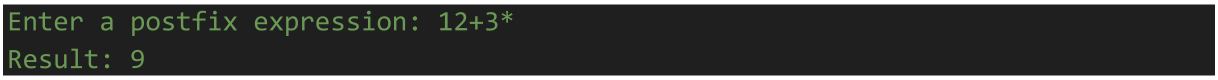


```c
#include <stdio.h>
#include <ctype.h>
#include <stdlib.h>

#define MAX_SIZE 100

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

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

int pop() {
    if (top < 0) {
        printf("Stack Underflow\n");
        return -1;
    }
    int x = stack[top];
    top--;
    return x;
}

int isOperand(char c) {
    return (c >= '0' && c <= '9');
}

int evaluatePostfix(char *postfix) {
```



```
    int i, op1, op2, result;

    for (i = 0; postfix[i] != '\0'; i++) {
        if (isOperand(postfix[i])) {
            push(postfix[i] - '0');
        } else {
            op2 = pop();
            op1 = pop();
            switch (postfix[i]) {
                case '+':
                    result = op1 + op2;
                    break;
                case '-':
                    result = op1 - op2;
                    break;
                case '*':
                    result = op1 * op2;
                    break;
                case '/':
                    result = op1 / op2;
                    break;
                default:
                    printf("Invalid operator\n");
                    return -1;
            }
            push(result);
        }
    }

    return pop();
}

int main() {
    char postfix[MAX_SIZE];

    printf("Enter a postfix expression: ");
    scanf("%s", postfix);

    int result = evaluatePostfix(postfix);
    printf("Result: %d\n", result);

    return 0;
}
```

Output :



```
Enter a postfix expression: 12+3*
Result: 9
```

5) Write a program to find GCD of two numbers using recursion.

Code :

```
#include <stdio.h>

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

int main() {
    int num1, num2;

    printf("Enter two integers: ");
    scanf("%d %d", &num1, &num2);

    int result = gcd(num1, num2);
    printf("GCD of %d and %d is %d\n", num1, num2, result);

    return 0;
}
```

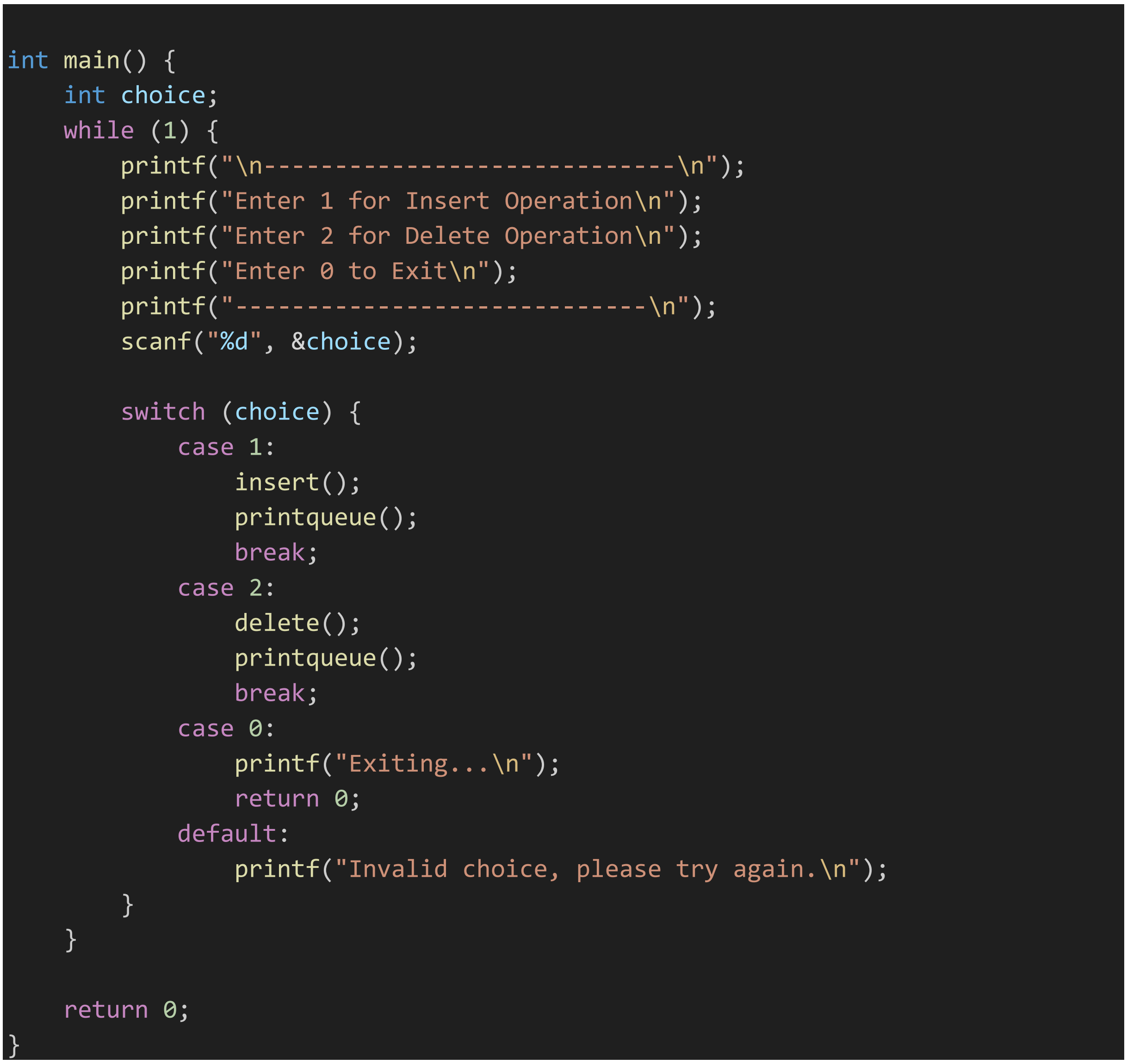
Output :

```
Enter two integers: 2 5
GCD of 2 and 5 is 1
```

6) Write a program to implement the concept of Simple Queue and perform insert and delete operations on simple queue.

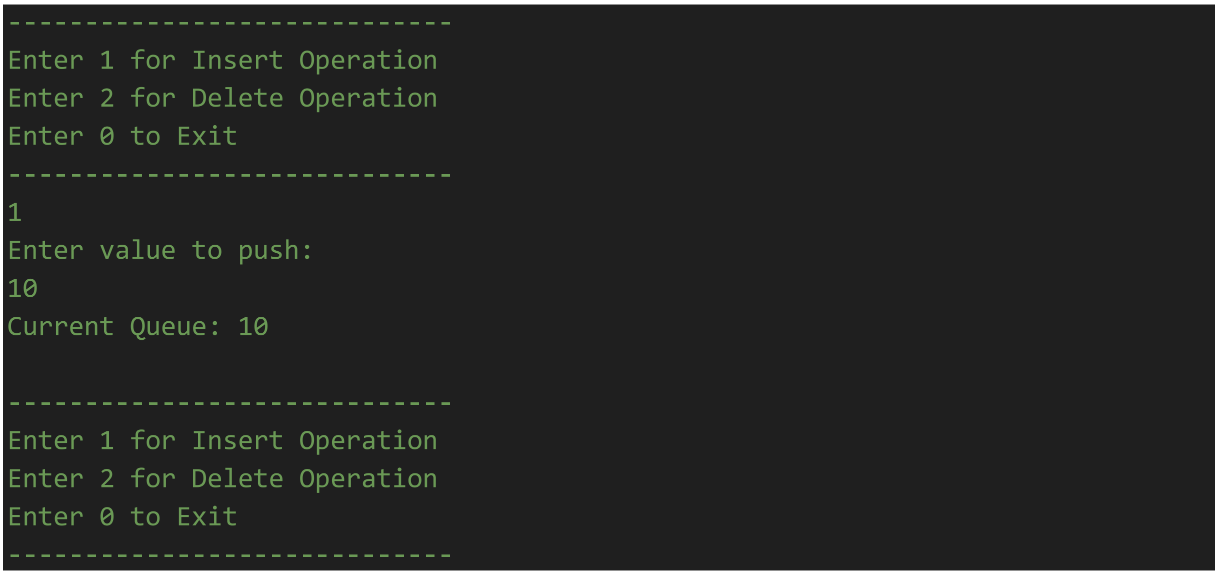
Code :

```
#include <stdio.h>

#define SIZE 5

int queue[SIZE];
int front = -1;
int rear = -1;
int value;

// Function to print the Queue
void printqueue() {
    if (front == -1) {
        printf("Queue is empty\n");
    } else {
        printf("Current Queue: ");
        for (int i = front; i <= rear; i++) {
            printf("%d ", queue[i]);
        }
        printf("\n");
    }
}

// Function to insert an element into the Queue
void insert() {
    if (rear >= SIZE - 1) {  // Check if the queue is full
        printf("\nQueue Overflow!\n");
    } else {
        printf("Enter value to push:\n");
        scanf("%d", &value);

        if (front == -1) {
            // If this is the first element to be added
            front = 0;
        }

        rear = rear + 1;  // Move the rear pointer
        queue[rear] = value;  // Insert the value into the queue
    }
}

// Function to delete an element from the Queue
void delete() {
    if (front == -1) {
        printf("Queue is empty\n");
    } else {
        value = queue[front];
        if (front == rear) {
            // If the queue becomes empty after this deletion
            front = rear = -1;
        } else {
            front = front + 1;
        }
        printf("Deleted %d from the queue.\n", value);
    }
}
```

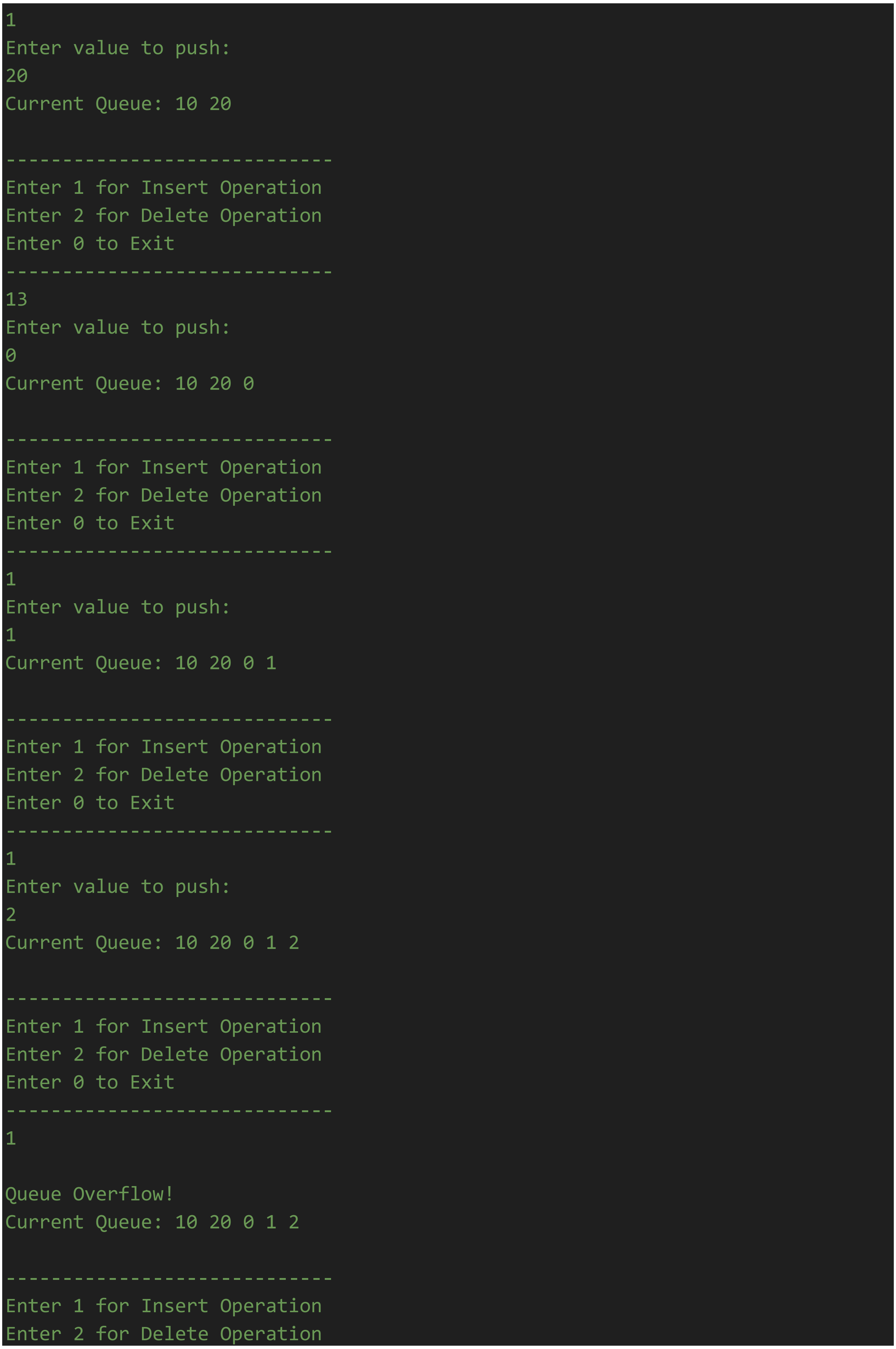


```c
int main() {
    int choice;
    while (1) {
        printf("\n-----------------------------\n");
        printf("Enter 1 for Insert Operation\n");
        printf("Enter 2 for Delete Operation\n");
        printf("Enter 0 to Exit\n");
        printf("-----------------------------\n");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                insert();
                printqueue();
                break;
            case 2:
                delete();
                printqueue();
                break;
            case 0:
                printf("Exiting...\n");
                return 0;
            default:
                printf("Invalid choice, please try again.\n");
        }
    }

    return 0;
}
```

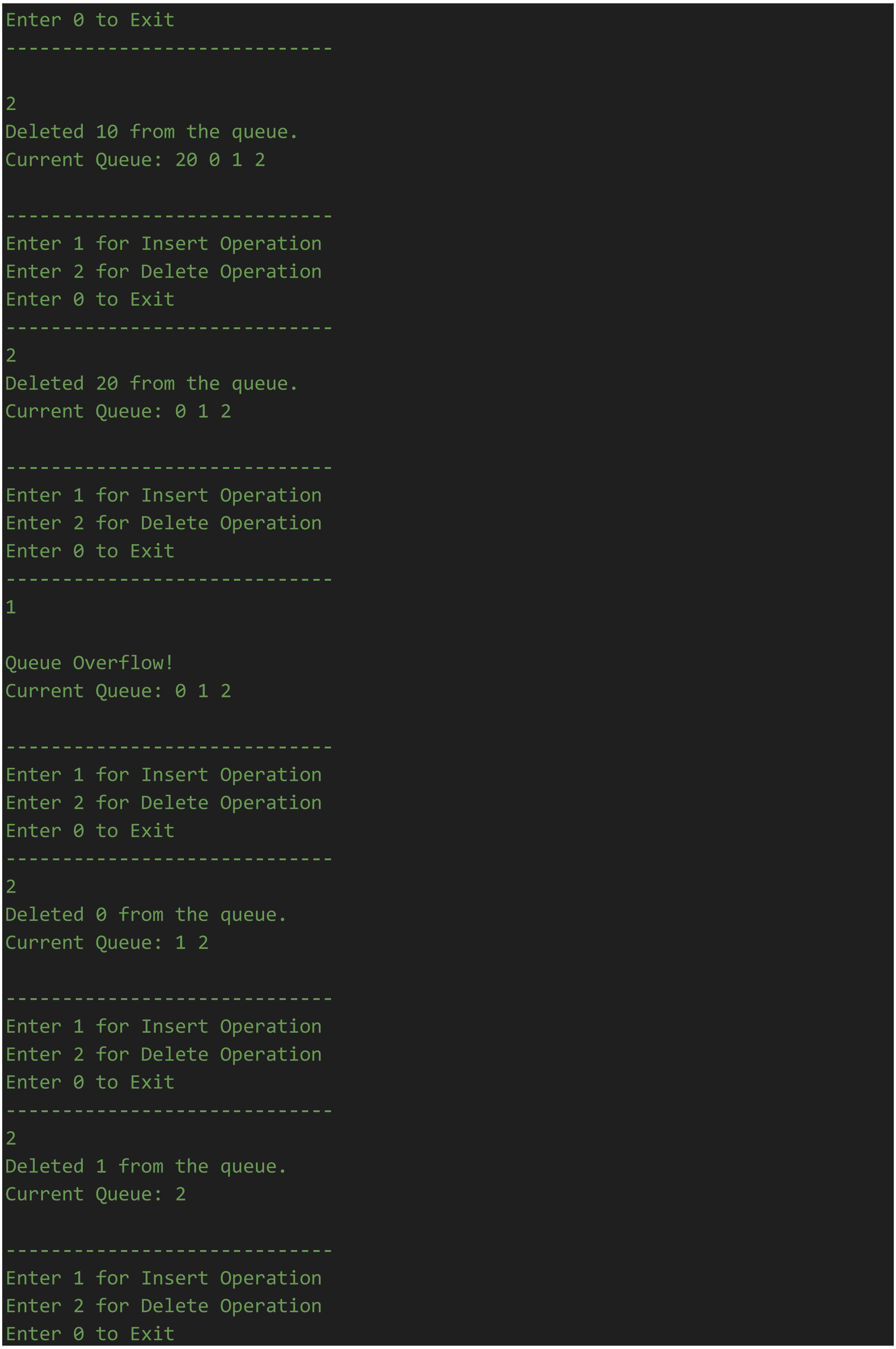
Output :



```
-----------------------------
Enter 1 for Insert Operation
Enter 2 for Delete Operation
Enter 0 to Exit
-----------------------------
1
Enter value to push:
10
Current Queue: 10

-----------------------------
Enter 1 for Insert Operation
Enter 2 for Delete Operation
Enter 0 to Exit
-----------------------------
```



```
1
Enter value to push:
20
Current Queue: 10 20

-----------------------------
Enter 1 for Insert Operation
Enter 2 for Delete Operation
Enter 0 to Exit
-----------------------------
13
Enter value to push:
0
Current Queue: 10 20 0

-----------------------------
Enter 1 for Insert Operation
Enter 2 for Delete Operation
Enter 0 to Exit
-----------------------------
1
Enter value to push:
1
Current Queue: 10 20 0 1

-----------------------------
Enter 1 for Insert Operation
Enter 2 for Delete Operation
Enter 0 to Exit
-----------------------------
1
Enter value to push:
2
Current Queue: 10 20 0 1 2

-----------------------------
Enter 1 for Insert Operation
Enter 2 for Delete Operation
Enter 0 to Exit
-----------------------------
1

Queue Overflow!
Current Queue: 10 20 0 1 2

-----------------------------
Enter 1 for Insert Operation
Enter 2 for Delete Operation
```



```
Enter 0 to Exit
-----------------------------

2
Deleted 10 from the queue.
Current Queue: 20 0 1 2

-----------------------------
Enter 1 for Insert Operation
Enter 2 for Delete Operation
Enter 0 to Exit
-----------------------------
2
Deleted 20 from the queue.
Current Queue: 0 1 2

-----------------------------
Enter 1 for Insert Operation
Enter 2 for Delete Operation
Enter 0 to Exit
-----------------------------
1

Queue Overflow!
Current Queue: 0 1 2

-----------------------------
Enter 1 for Insert Operation
Enter 2 for Delete Operation
Enter 0 to Exit
-----------------------------
2
Deleted 0 from the queue.
Current Queue: 1 2

-----------------------------
Enter 1 for Insert Operation
Enter 2 for Delete Operation
Enter 0 to Exit
-----------------------------
2
Deleted 1 from the queue.
Current Queue: 2

-----------------------------
Enter 1 for Insert Operation
Enter 2 for Delete Operation
Enter 0 to Exit
```

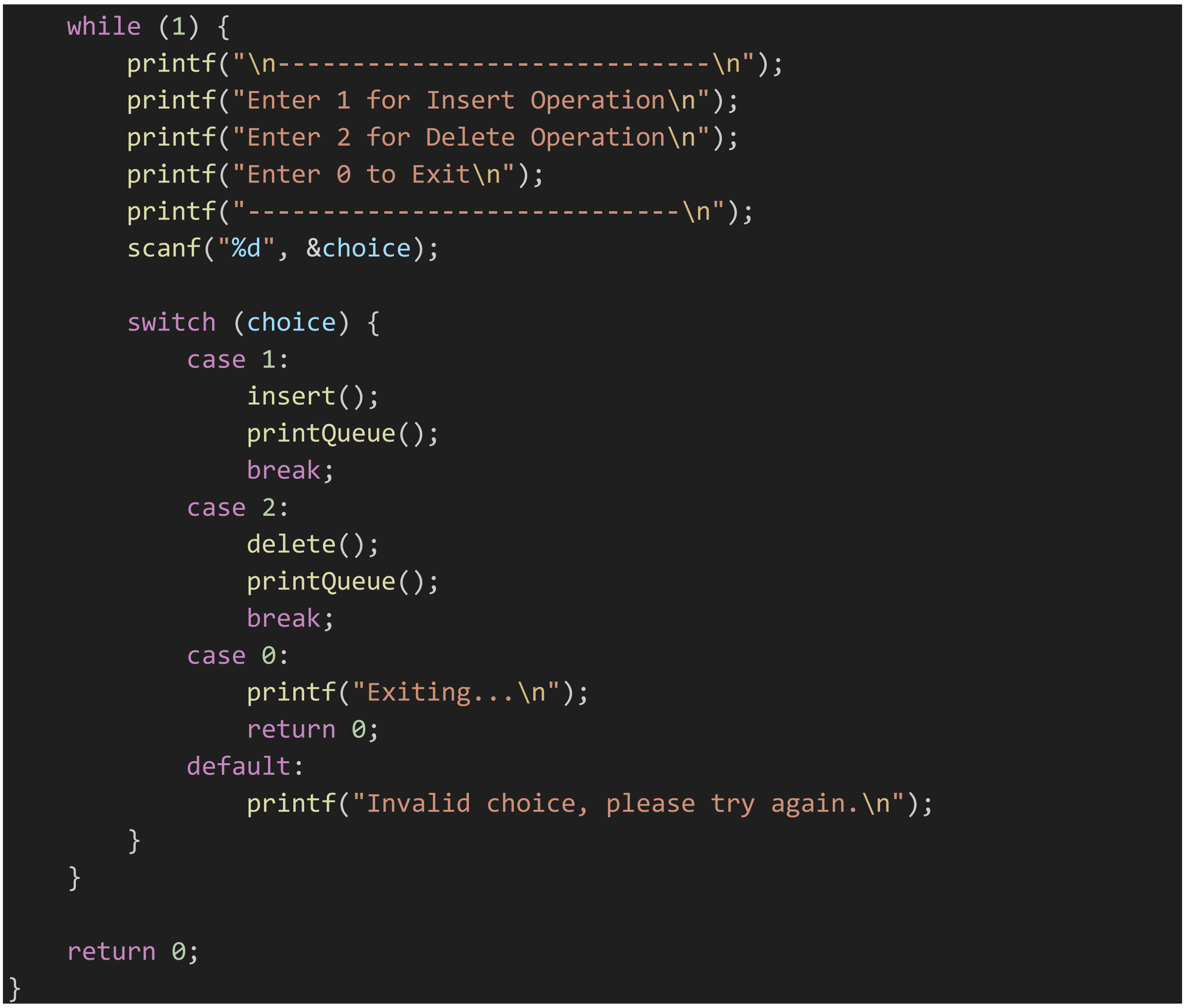
```
-----------------------------
2
Deleted 2 from the queue.
Queue is empty

-----------------------------
Enter 1 for Insert Operation
Enter 2 for Delete Operation
Enter 0 to Exit
-----------------------------
0
Exiting...
```

7) Write a program to implement the concept of Circular Queue and perform insert and delete operations on circular queue.

Code :

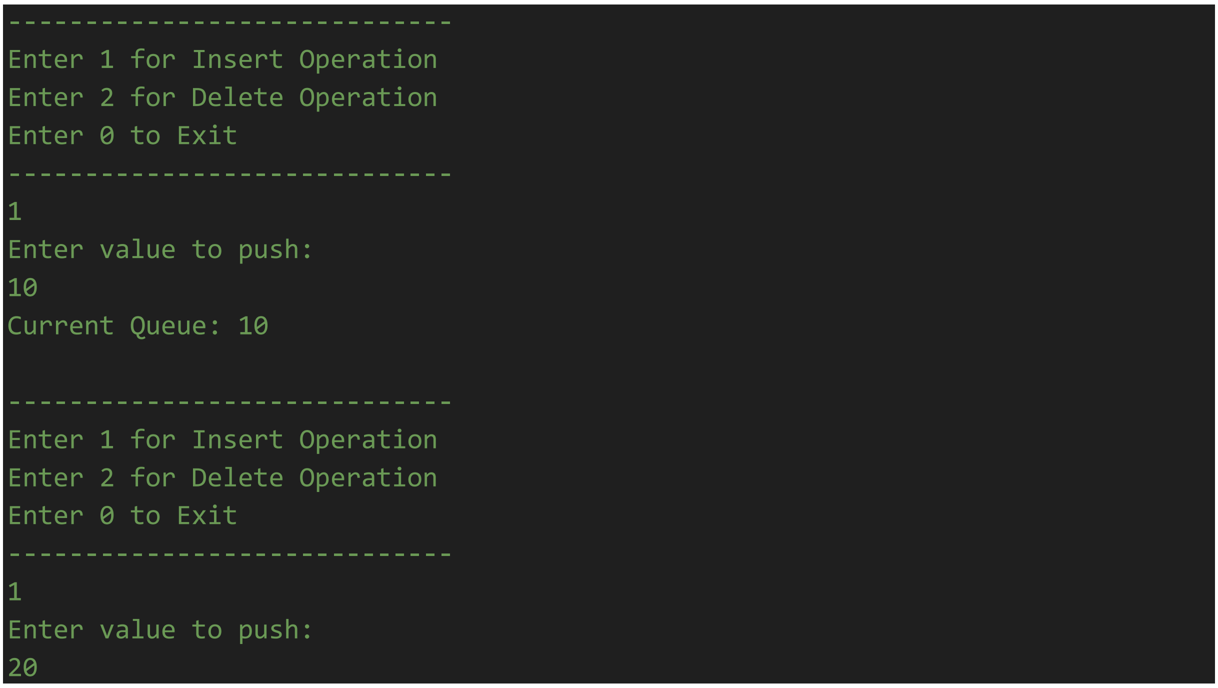
```
#include <stdio.h>

#define SIZE 3

int queue[SIZE];
int front = -1;
int rear = -1;

// Function to print the Queue
void printQueue() {
    if (front == -1) {
        printf("Queue is empty\n");
    } else {
        printf("Current Queue: ");
        if (rear >= front) {
            for (int i = front; i <= rear; i++) {
                printf("%d ", queue[i]);
            }
        } else {
            for (int i = front; i < SIZE; i++) {
                printf("%d ", queue[i]);
            }
            for (int i = 0; i <= rear; i++) {
                printf("%d ", queue[i]);
            }
        }
        printf("\n");
    }
}
```

```
// Function to insert an element into the Queue
void insert() {
    if ((front == 0 && rear == SIZE - 1) || (front == rear + 1)) {
        printf("\nQueue Overflow!\n");
        return;
    }

    if (rear == SIZE - 1 && front != 0) {
        rear = 0;
    } else if (rear == -1) {
        front = rear = 0;
    } else {
        rear++;
    }

    printf("Enter value to push:\n");
    scanf("%d", &queue[rear]);
}

// Function to delete an element from the Queue
void delete() {
    if (front == -1) {
        printf("Queue is empty\n");
        return;
    }

    int deletedValue = queue[front];
    printf("Deleted %d from the queue.\n", deletedValue);

    // Ensure the deleted value is not reinserted
    for (int i = front; i < SIZE; i++) {
        if (queue[i] == deletedValue) {
            queue[i] = 0; // Mark the deleted value as 0
        }
    }

    if (front == rear) {
        front = rear = -1;
    } else if (front == SIZE - 1) {
        front = 0;
    } else {
        front++;
    }
}

int main() {
    int choice;
```

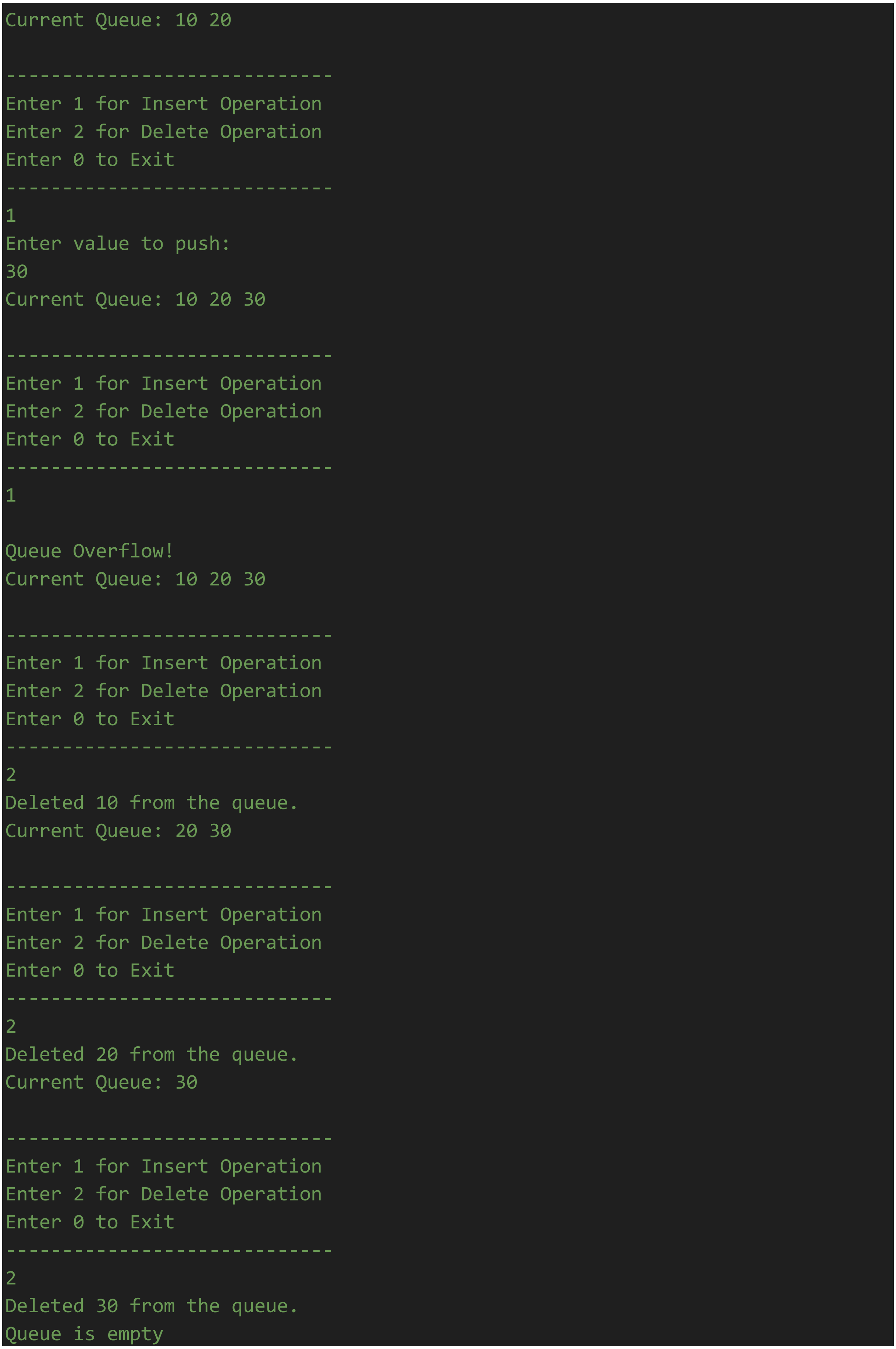


```
    while (1) {
        printf("\n-----------------------------\n");
        printf("Enter 1 for Insert Operation\n");
        printf("Enter 2 for Delete Operation\n");
        printf("Enter 0 to Exit\n");
        printf("-----------------------------\n");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                insert();
                printQueue();
                break;
            case 2:
                delete();
                printQueue();
                break;
            case 0:
                printf("Exiting...\n");
                return 0;
            default:
                printf("Invalid choice, please try again.\n");
        }
    }

    return 0;
}
```

Output :



```
-----------------------------
Enter 1 for Insert Operation
Enter 2 for Delete Operation
Enter 0 to Exit
-----------------------------
1
Enter value to push:
10
Current Queue: 10

-----------------------------
Enter 1 for Insert Operation
Enter 2 for Delete Operation
Enter 0 to Exit
-----------------------------
1
Enter value to push:
20
```



```
Current Queue: 10 20

-----------------------------
Enter 1 for Insert Operation
Enter 2 for Delete Operation
Enter 0 to Exit
-----------------------------
1
Enter value to push:
30
Current Queue: 10 20 30

-----------------------------
Enter 1 for Insert Operation
Enter 2 for Delete Operation
Enter 0 to Exit
-----------------------------
1

Queue Overflow!
Current Queue: 10 20 30

-----------------------------
Enter 1 for Insert Operation
Enter 2 for Delete Operation
Enter 0 to Exit
-----------------------------
2
Deleted 10 from the queue.
Current Queue: 20 30

-----------------------------
Enter 1 for Insert Operation
Enter 2 for Delete Operation
Enter 0 to Exit
-----------------------------
2
Deleted 20 from the queue.
Current Queue: 30

-----------------------------
Enter 1 for Insert Operation
Enter 2 for Delete Operation
Enter 0 to Exit
-----------------------------
2
Deleted 30 from the queue.
Queue is empty
```

```

-----------------------------
Enter 1 for Insert Operation
Enter 2 for Delete Operation
Enter 0 to Exit
-----------------------------
0
Exiting...
```

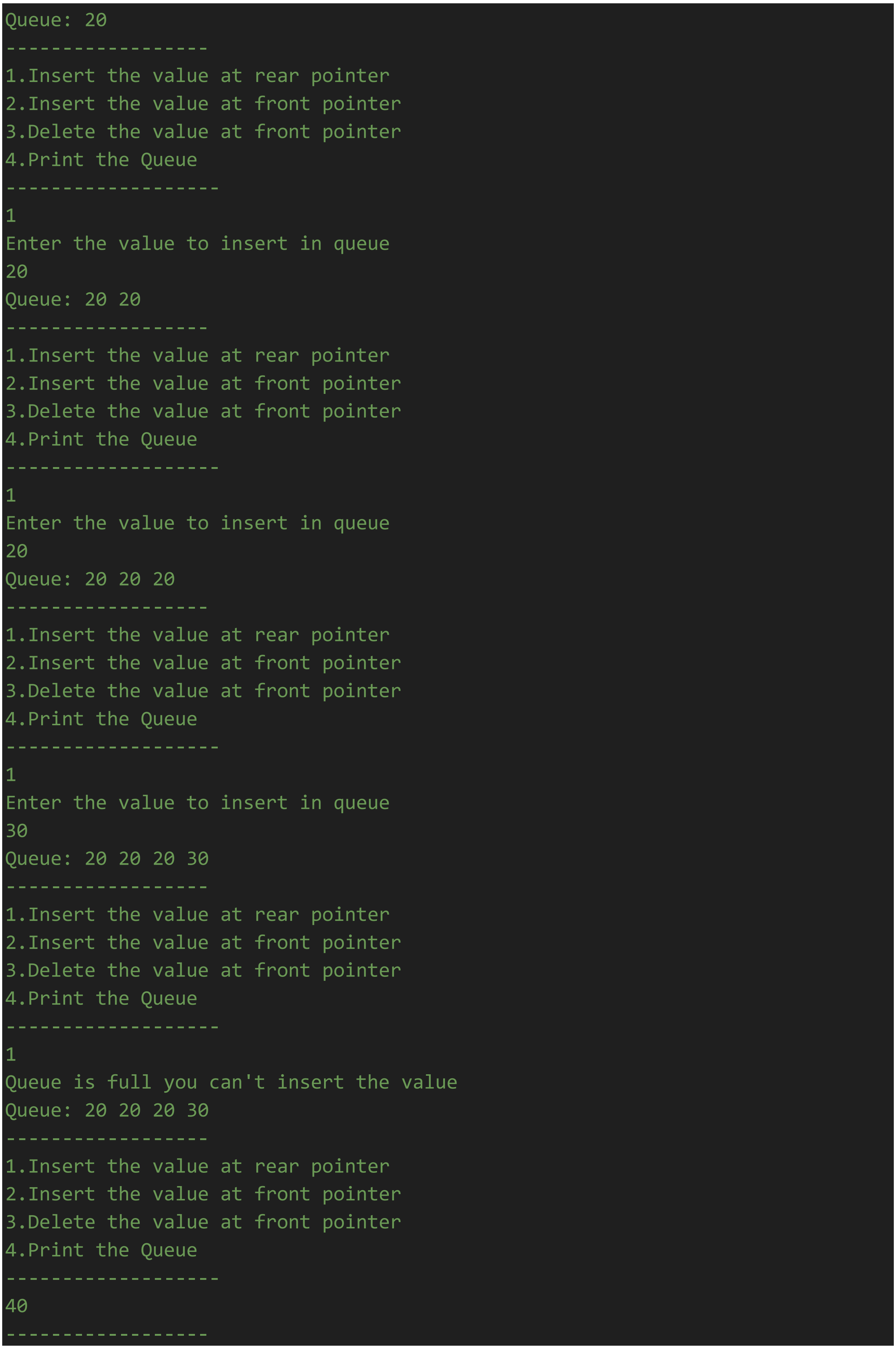
8) **Write a program to implement the concept of Deque and perform insert and delete operations on deque.**

Code :

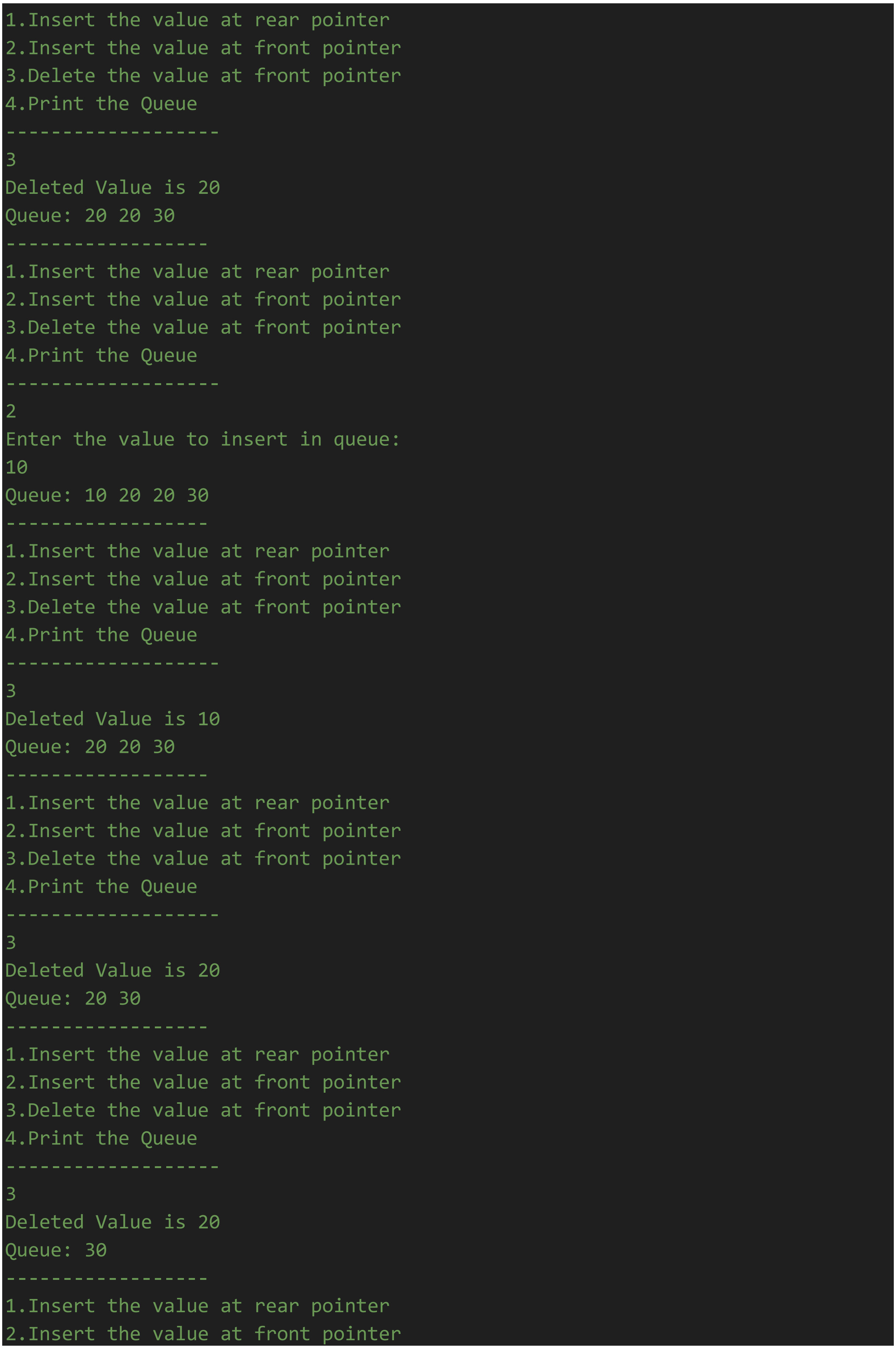
```
#include <stdio.h>
#define SIZE 4

int rear = -1;
int front = -1;
int Queue[SIZE];
int value;

void insertRear(){
    if ((rear + 1) % SIZE == front)
    {
        printf("Queue is full you can't insert the value\n");
    }
    else{
        if (front==-1)
        {
            front=0;
            rear=0;
        }
        else{
            rear=(rear+1)%SIZE;
        }
        printf("Enter the value to insert in queue\n");
        scanf("%d",&value);
        Queue[rear]=value;
    }
}

void insertFront(){
    if ((front == 0 && rear == SIZE - 1) || (front == rear + 1)) {
        printf("Queue Overflow\n");
        return;
    } else {
        if (front == -1) {
            front = 0;
            rear = 0;
        } else if (front == 0) {
            front = SIZE - 1;
        } else {
            front = front - 1;
        }
        printf("Enter the value to insert in queue:\n");
        scanf("%d", &value);
        Queue[front] = value;
    }
}

void deleteFront(){
    if (front==-1)
    {
        printf("Queue is Empty\n");
    }
    else{
        value=Queue[front];
        printf("Deleted Value is %d\n",value);
        if (front==rear)
        {
            front=rear=-1;
        }
        else{
            front = (front + 1) % SIZE;
        }
    }
}

void printQueue() {
    if (front == -1) {
        printf("Queue is empty.\n");
    } else {
        printf("Queue: ");
        int i = front;
        while (i != rear) {
            printf("%d ", Queue[i]);
            i = (i + 1) % SIZE;
        }
        printf("%d\n", Queue[rear]);
    }
}

int main(){
    int choice;
    while (1)
    {
        printf("------------------\n");
        printf("1.Insert the value at rear pointer\n");
        printf("2.Insert the value at front pointer\n");
        printf("3.Delete the value at front pointer\n");
        printf("4.Print the Queue\n");
        printf("-------------------\n");
        scanf("%d",&choice);

        switch (choice){
        case 1:
            insertRear();
            printQueue();
            break;

        case 2:
            insertFront();
            printQueue();
            break;

        case 3:
            deleteFront();
            printQueue();
            break;

        case 4:
            printQueue();
            break;

        default:
            break;
        }
    }
    return 0;
}
```

Output :

```
------------------
1.Insert the value at rear pointer
2.Insert the value at front pointer
3.Delete the value at front pointer
4.Print the Queue
-------------------
1
Enter the value to insert in queue
20
```



```
Queue: 20
------------------
1.Insert the value at rear pointer
2.Insert the value at front pointer
3.Delete the value at front pointer
4.Print the Queue
-------------------
1
Enter the value to insert in queue
20
Queue: 20 20
------------------
1.Insert the value at rear pointer
2.Insert the value at front pointer
3.Delete the value at front pointer
4.Print the Queue
-------------------
1
Enter the value to insert in queue
20
Queue: 20 20 20
------------------
1.Insert the value at rear pointer
2.Insert the value at front pointer
3.Delete the value at front pointer
4.Print the Queue
-------------------
1
Enter the value to insert in queue
30
Queue: 20 20 20 30
------------------
1.Insert the value at rear pointer
2.Insert the value at front pointer
3.Delete the value at front pointer
4.Print the Queue
-------------------
1
Queue is full you can't insert the value
Queue: 20 20 20 30
------------------
1.Insert the value at rear pointer
2.Insert the value at front pointer
3.Delete the value at front pointer
4.Print the Queue
-------------------
40
------------------
```



```
1.Insert the value at rear pointer
2.Insert the value at front pointer
3.Delete the value at front pointer
4.Print the Queue
-------------------
3
Deleted Value is 20
Queue: 20 20 30
------------------
1.Insert the value at rear pointer
2.Insert the value at front pointer
3.Delete the value at front pointer
4.Print the Queue
-------------------
2
Enter the value to insert in queue:
10
Queue: 10 20 20 30
------------------
1.Insert the value at rear pointer
2.Insert the value at front pointer
3.Delete the value at front pointer
4.Print the Queue
-------------------
3
Deleted Value is 10
Queue: 20 20 30
------------------
1.Insert the value at rear pointer
2.Insert the value at front pointer
3.Delete the value at front pointer
4.Print the Queue
-------------------
3
Deleted Value is 20
Queue: 20 30
------------------
1.Insert the value at rear pointer
2.Insert the value at front pointer
3.Delete the value at front pointer
4.Print the Queue
-------------------
3
Deleted Value is 20
Queue: 30
------------------
1.Insert the value at rear pointer
2.Insert the value at front pointer
```

```
3.Delete the value at front pointer
4.Print the Queue
-------------------
3
Deleted Value is 30
Queue is empty.
```