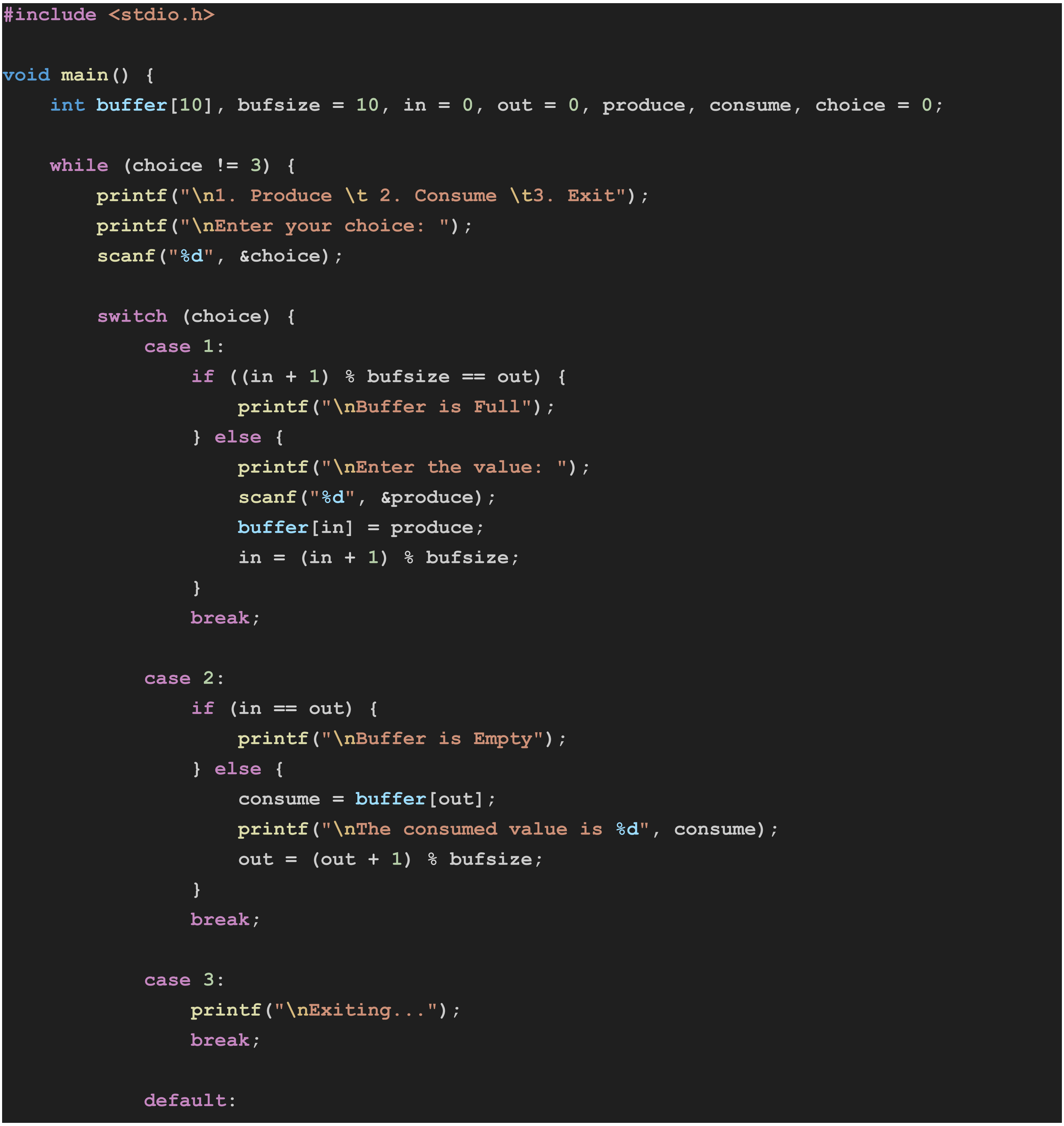
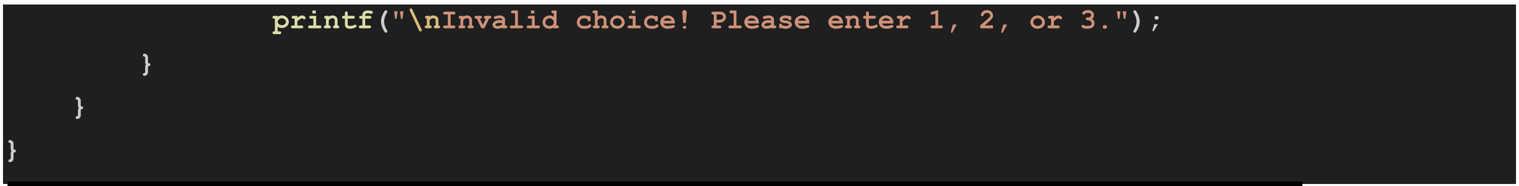
NAME: MUHAMMAD SAIM NOMANI
ROLL NO: DT-22030
SUBJECT: OPERATING SYSTEM
CODE: CT-353
LAB: 04
DATA SCIENCE
THIRD YEAR

Exercise:

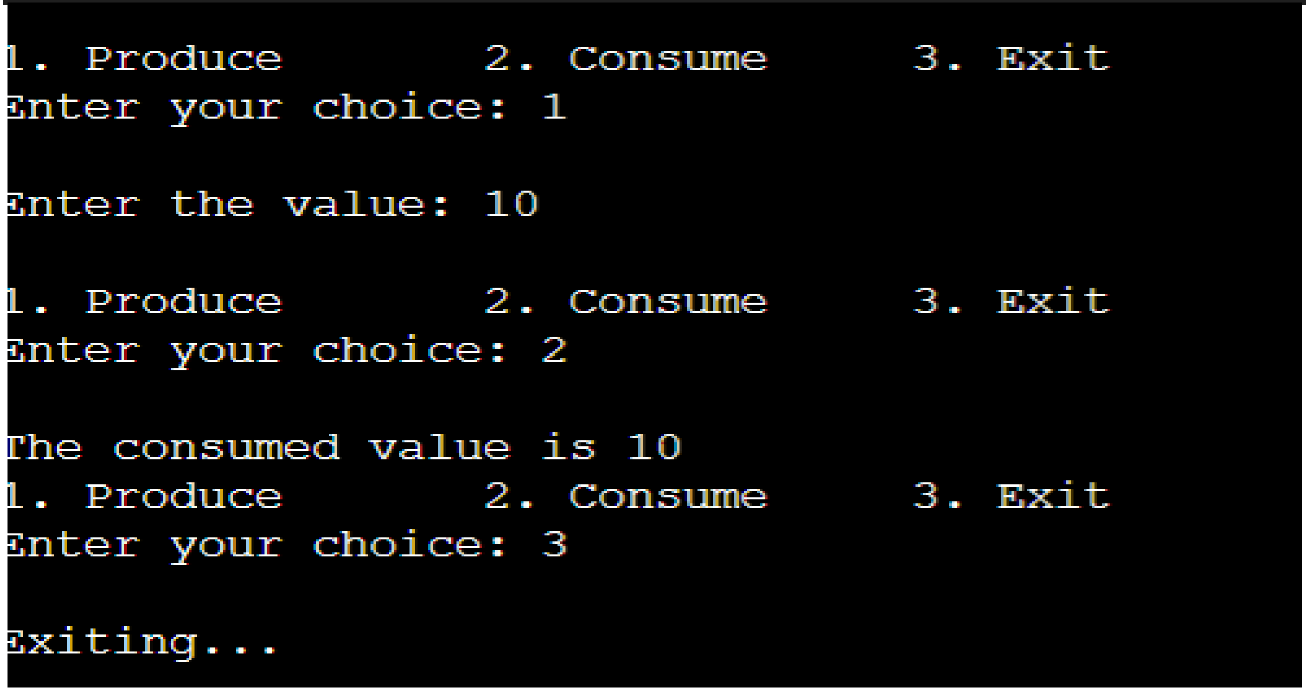
1) Implement the above code and paste the screen shot of the output.



```
#include <stdio.h>

void main() {
    int buffer[10], bufsize = 10, in = 0, out = 0, produce, consume, choice = 0;

    while (choice != 3) {
        printf("\n1. Produce \t 2. Consume \t3. Exit");
        printf("\nEnter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                if ((in + 1) % bufsize == out) {
                    printf("\nBuffer is Full");
                } else {
                    printf("\nEnter the value: ");
                    scanf("%d", &produce);
                    buffer[in] = produce;
                    in = (in + 1) % bufsize;
                }
                break;

            case 2:
                if (in == out) {
                    printf("\nBuffer is Empty");
                } else {
                    consume = buffer[out];
                    printf("\nThe consumed value is %d", consume);
                    out = (out + 1) % bufsize;
                }
                break;

            case 3:
                printf("\nExiting...");
                break;

            default:
```

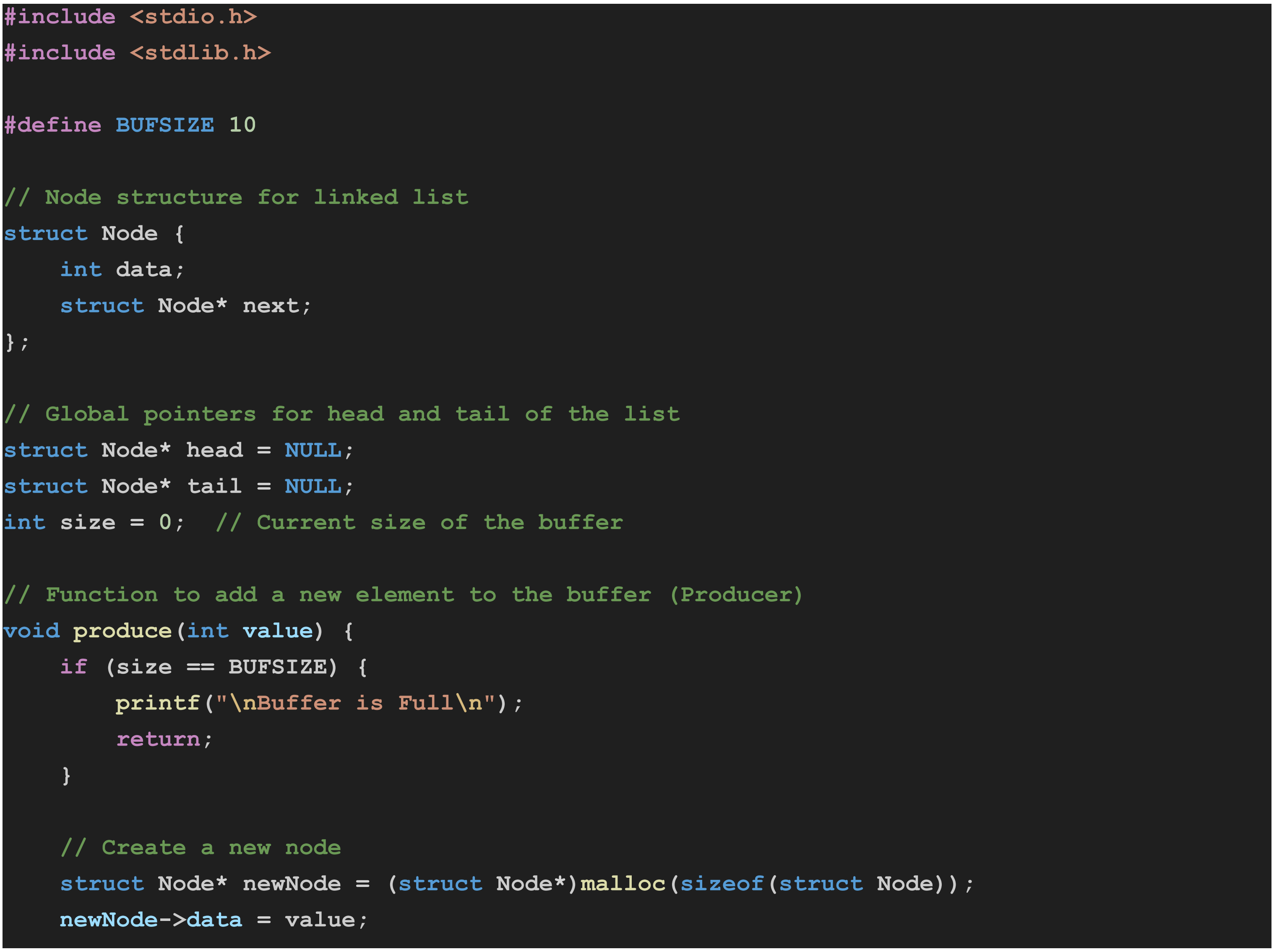


```
                printf("\nInvalid choice! Please enter 1, 2, or 3.");
        }
    }
}
```

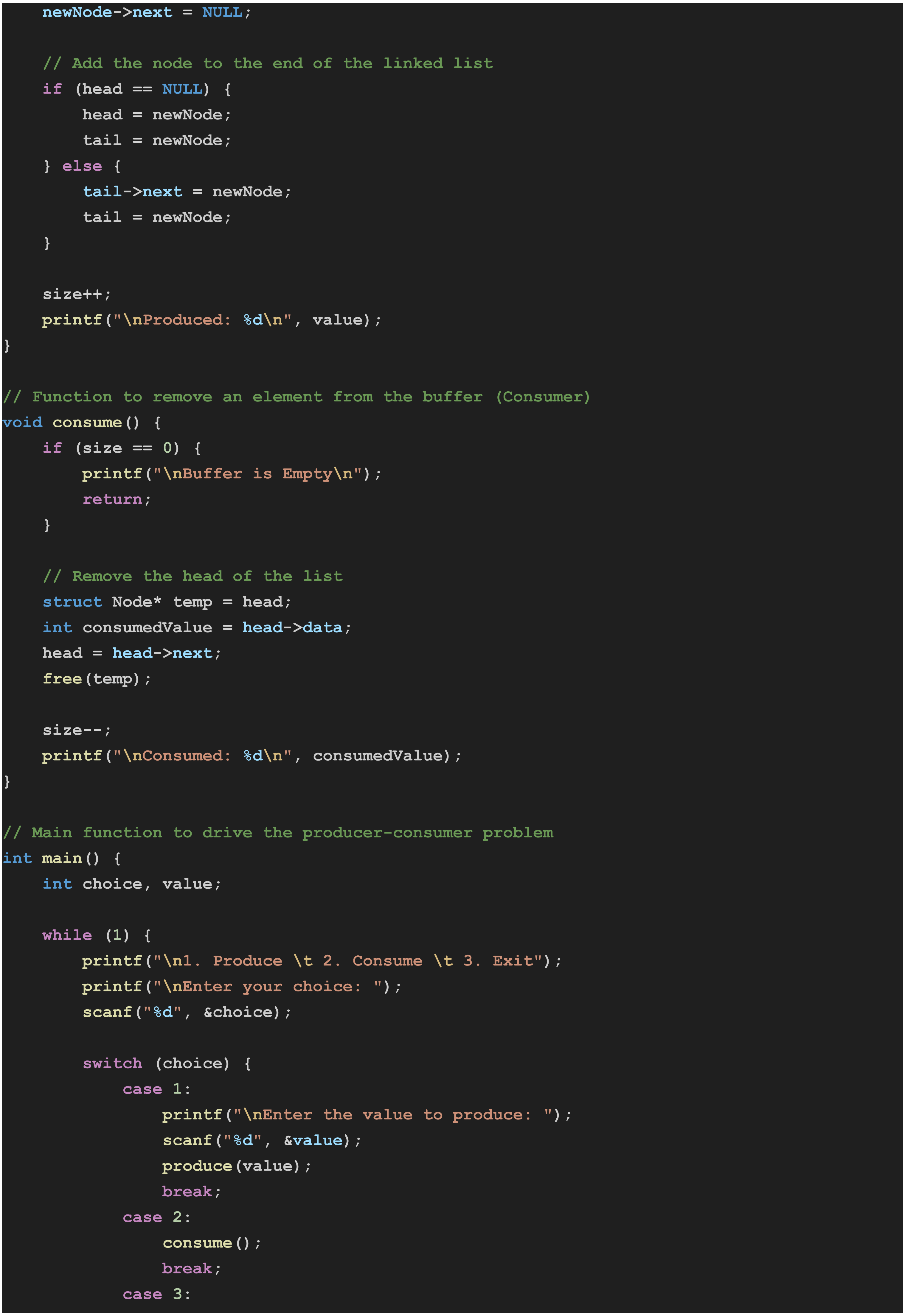


```
1. Produce          2. Consume       3. Exit
Enter your choice: 1

Enter the value: 10

1. Produce          2. Consume       3. Exit
Enter your choice: 2

The consumed value is 10
1. Produce          2. Consume       3. Exit
Enter your choice: 3

Exiting...
```

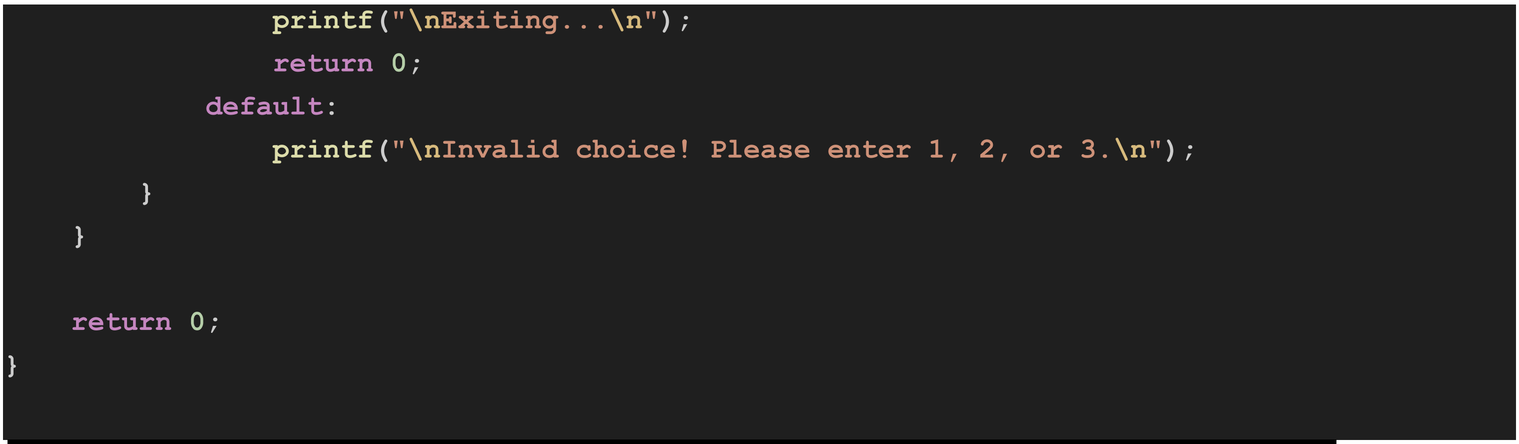
2) Solve the producer-consumer problem using linked list. (You can perform this task using any programming language)
Note: Keep the buffer size to 10 places.



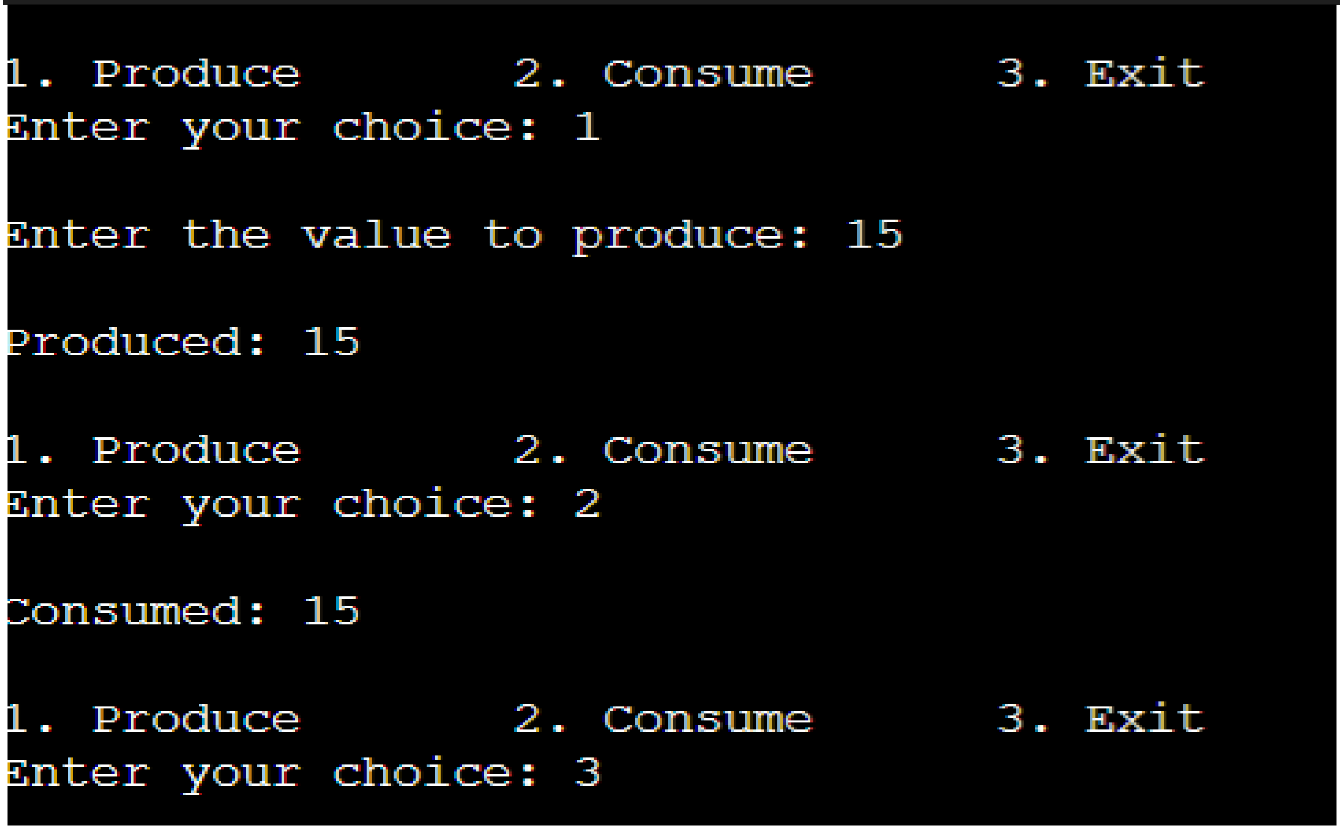
```c
#include <stdio.h>
#include <stdlib.h>

#define BUFSIZE 10

// Node structure for linked list
struct Node {
    int data;
    struct Node* next;
};

// Global pointers for head and tail of the list
struct Node* head = NULL;
struct Node* tail = NULL;
int size = 0;  // Current size of the buffer

// Function to add a new element to the buffer (Producer)
void produce(int value) {
    if (size == BUFSIZE) {
        printf("\nBuffer is Full\n");
        return;
    }

    // Create a new node
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
```



```
    newNode->next = NULL;

    // Add the node to the end of the linked list
    if (head == NULL) {
        head = newNode;
        tail = newNode;
    } else {
        tail->next = newNode;
        tail = newNode;
    }

    size++;
    printf("\nProduced: %d\n", value);
}

// Function to remove an element from the buffer (Consumer)
void consume() {
    if (size == 0) {
        printf("\nBuffer is Empty\n");
        return;
    }

    // Remove the head of the list
    struct Node* temp = head;
    int consumedValue = head->data;
    head = head->next;
    free(temp);

    size--;
    printf("\nConsumed: %d\n", consumedValue);
}

// Main function to drive the producer-consumer problem
int main() {
    int choice, value;

    while (1) {
        printf("\n1. Produce \t 2. Consume \t 3. Exit");
        printf("\nEnter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("\nEnter the value to produce: ");
                scanf("%d", &value);
                produce(value);
                break;
            case 2:
                consume();
                break;
            case 3:
```



```
                printf("\nExiting...\n");
                return 0;
            default:
                printf("\nInvalid choice! Please enter 1, 2, or 3.\n");
        }
    }

    return 0;
}
```



```
1. Produce            2. Consume            3. Exit
Enter your choice: 1

Enter the value to produce: 15

Produced: 15

1. Produce            2. Consume            3. Exit
Enter your choice: 2

Consumed: 15

1. Produce            2. Consume            3. Exit
Enter your choice: 3
```

3) In producer-consumer problem what difference will it make if we utilize stack for the buffer rather than an array?

1. Behavioral Difference

- **Array (or Queue)**:
  Typically, an array implements the buffer in a **FIFO (First In, First Out)** manner. The producer adds items at the end of the array, and the consumer removes items from the front.
  - **Order:** Items are consumed in the order they were produced.
- **Stack**:
  A stack operates in a **LIFO (Last In, First Out)** manner. The producer pushes items onto the top of the stack, and the consumer pops items from the top.
  - **Order:** The most recently produced item is consumed first.

2. Impact on the Problem

- **Real-World Use Case Differences**:

    - **Queue** (array): Simulates real-world scenarios like printing jobs, packet processing, or task scheduling, where processing order matters.
    - **Stack**: May be suitable for scenarios like undo functionality or recursive function calls, where the latest item is consumed first. However, it’s less common for typical Producer-Consumer scenarios.
- **Potential Starvation**:
    - Using a stack could lead to older items remaining in the buffer for a long time (or forever), as newer items are always consumed first.

3.Pros and Cons of Using a Stack

- **Pros**:
    - Easier to implement than a circular buffer (no need to manage front and rear pointers).
    - More efficient for scenarios where LIFO behavior is desired.
- **Cons**:
    - Breaks FIFO ordering, which may not be desirable in many Producer-Consumer scenarios.
    - Could lead to starvation of older items if new items keep arriving rapidly.