LAB-04

Exercise:

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

PROGRAM:

```c
#include <stdio.h>

void main() {
    int buffer[10], bufsize, in, out, produce, consume, choice = 0;
    in = 0;
    out = 0;
    bufsize = 10;
    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 try again.");
        }
    }}
```

OUTPUT:

```
PS D:\OS labs> cd "d:\OS labs\" ; if ($?) { gcc lab_4_1.c -o lab_4_1 } ; if ($?) { .\lab_4_1 }

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...
PS D:\OS labs>
```

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.

PROGRAM:

```c
#include <stdio.h>
#include <stdlib.h>
struct Node
{
    int data;
    struct Node *next;
};
void produce();
void consume();
void display();
// Global variables
struct Node *front = NULL;
struct Node *rear = NULL;
int count = 0;
int bufferSize = 10;

void main()
{
    int choice;

    while (1)
    {
        printf("\n1. Produce \t 2. Consume \t 3. Display \t 4. Exit");
        printf("\nEnter your choice: ");
        scanf("%d", &choice);

        switch (choice)
        {
        case 1:
            produce();
```

```
            break;
        case 2:
            consume();
            break;
        case 3:
            display();
            break;
        case 4:
            printf("\nExiting...");
            exit(0);
        default:
            printf("\nInvalid choice, please try again.");
        }
    }
}
void produce()
{
    if (count == bufferSize)
    {
        printf("\nBuffer is Full. Cannot produce.");
        return;
    }

    int value;
    printf("\nEnter the value to produce: ");
    scanf("%d", &value);
    struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->next = NULL;

    if (front == NULL && rear == NULL)
    {
        front = rear = newNode;
    }
    else
    {
        rear->next = newNode;
        rear = newNode;
    }
    count++;
    printf("\nProduced: %d", value);
}
void consume()
{
    if (front == NULL)
    {
        printf("\nBuffer is Empty. Cannot consume.");
        return;
    }
```

```
    struct Node *temp = front;
    int value = temp->data;
    front = front->next;

    if (front == NULL)
    {
        rear = NULL;
    }
    free(temp);
    count--;
    printf("\nConsumed: %d", value);
}

// Display buffer contents
void display()
{
    if (front == NULL)
    {
        printf("\nBuffer is Empty.");
        return;
    }

    struct Node *temp = front;
    printf("\nBuffer contents: ");
    while (temp != NULL)
    {
        printf("%d ", temp->data);
        temp = temp->next;
    }
}
```

OUTPUT:

```
PS D:\OS labs> cd "d:\OS labs\" ; if ($?) { gcc lab_4_2.c -o lab_4_2 } ; if ($?) { .\lab_4_2 }

1. Produce      2. Consume      3. Display      4. Exit
Enter your choice: 1

Enter the value to produce: 2

Produced: 2
1. Produce      2. Consume      3. Display      4. Exit
Enter your choice: 2

Consumed: 2
1. Produce      2. Consume      3. Display      4. Exit
Enter your choice: 3

Buffer is Empty.
1. Produce      2. Consume      3. Display      4. Exit
Enter your choice: 4
```

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

Using a stack instead of an array (or, more commonly, a queue) fundamentally changes the order in which items are processed:

- **Order of Processing**:
  - **Array/Queue (FIFO)**: Items are consumed in the order they are produced. This is important when the sequence of events matters.
  - **Stack (LIFO)**: The most recently produced item is the first to be consumed. This reverses the natural order of production, which can lead to earlier items waiting indefinitely if new items continue to arrive.
- **Implications**:
  - In many producer-consumer scenarios, maintaining the production order is critical (e.g., processing tasks in the order they were submitted). Switching to a stack would violate this order and might cause logical errors in applications expecting FIFO behavior.
  - In certain cases, LIFO processing might be desirable (for example, when the latest data is most relevant), but this is not typical for standard producer-consumer problems.
- **Performance Considerations**:
  - Both stacks and circular arrays (queues) can offer O(1) operations for insertion and removal. However, the difference lies in their behavior—order of retrieval—rather than performance.

In summary, using a stack changes the behavior from FIFO to LIFO, which may not meet the requirements of applications that rely on processing items in the order they were produced.