C O D E :

Printer Spooler (Circular Queue):

/*In a multi-user environment, printers often use a circular queue to manage print jobs. Each print job is added to the queue, and the printer processes them in the order they arrive. Once a print job is completed, it moves out of the queue, and the next job is processed, efficiently managing the flow of print tasks. Implement the Printer Spooler system using a circular queue without using built-in queues.*/

```
#include <iostream>
#define MAXSIZE 2
#define MIN 0
using namespace std;

class Printer_spooler {
   int token[MAXSIZE];

public:
   int rear;
   int front;

   Printer_spooler() {
      rear = -1;
      front = -1;
   }

   bool isFull();
   bool isEmpty();
   void enQueue(int t);
   int deQueue();
   void display();
};

// Check if queue is full (Circular Queue Condition)
bool Printer_spooler::isFull() {
   if (((rear + 1) % MAXSIZE) == front) {
      return 1;
   } else {
      return 0;
   }
}

// Check if queue is empty
bool Printer_spooler::isEmpty() {
   if (front == -1) {
      return 1;
   } else {
      return 0;
   }
}

// Add a job to the printer queue
void Printer_spooler::enQueue(int t) {
   if (isFull()) {
      cout << "Sorry! The printer can't proceed.\nThe queue is full!" << endl;
   } else {
```

```
        if (rear == -1) {
            front = 0;
        }

        rear = (rear + 1) % MAXSIZE;
        token[rear] = t;
    }
}

// Remove a job from the printer queue (simulate printing)
int Printer_spooler::deQueue() {
    if (isEmpty()) {
        cout << "The queue is empty." << endl;
        return 0;
    }
    else if (front == rear) {
        int t = token[front];
        cout << "Printing job completed for: " << t << endl;
        front = -1;
        rear = -1;
        return t;
    }
    else {
        int t = token[front];
        cout << "Printing job completed for: " << t << endl;
        front = (front + 1) % MAXSIZE;
        return t;
    }
}

// Display all jobs in the printer queue
void Printer_spooler::display() {
    cout << "\nDisplaying the queue in printer spooler:" << endl;
    for (int i = 0; i < MAXSIZE; i++) {
        cout << "Printing job at " << i << " is: " << token[i] << endl;
    }
}

int main() {

    Printer_spooler customer;
    int choice = 0, order;

    do {
        cout << "\nPrinter job menu:" << endl;
        cout << " 1. Add a job\n 2. Delete a job \n 3. Display\n 4. Exit\nChoice: ";
        cin >> choice;

        switch (choice) {
        case 1: {
            cout << "\nEnter value to be inserted as order: ";
            cin >> order;
            customer.enQueue(order);
            cout << endl;
            break;
        }
        case 2: {
```

```
            cout << "\nPrinting:" << endl;
            customer.deQueue();
            cout << "\nPrinting job is completed." << endl;
            break;
        }
        case 3: {
            customer.display();
            cout << endl;
            break;
        }
        case 4: {
            cout << "Thank you!" << endl;
            break;
        }
        default: {
            cout << "\nIncorrect choice! Try again" << endl;
        }
        }

    } while (choice != 4);

    return 0;
}
```

OUTPUT:

```
Printer job menu:
1. Add a job
2. Delete a job
3. Display
4. Exit
Choice: 1

Enter value to be inserted as order: 34

Printer job menu:
1. Add a job
2. Delete a job
3. Display
4. Exit
Choice: 1

Enter value to be inserted as order: 35

Printer job menu:
1. Add a job
2. Delete a job
3. Display
4. Exit
Choice: 1

Enter value to be inserted as order: 36
Sorry! The printer can't proceed.
The queue is full!

Printer job menu:
1. Add a job
2. Delete a job
```

3. Display
4. Exit
Choice: 2

Printing:
Printing job completed for: 34

Printing job is completed.
Printer job menu:
1. Add a job
2. Delete a job
3. Display
4. Exit
Choice: 3

Displaying the queue in printer spooler:
Printing job at 0 is: 34
Printing job at 1 is: 35

Printer job menu:
1. Add a job
2. Delete a job
3. Display
4. Exit
Choice: 4

Thank you!

GITHUB: