1. Implement a circular queue using an array in C++. Define a class CircularQueue with operations: enqueue(), dequeue(), front(), rear(), and isFull(). Handle circular movement using modular arithmetic. Implement error handling for circular queue overflow and underflow. Add a function size() to return the current number of elements in the queue. Implement a display() function to print all the elements of the circular queue.

SOURCE CODE

```
#include <iostream>
#include <stdexcept>
using namespace std;
// COUNT TO KEEP TRACK OF NO OF ELEMENTS, SIZE TO SHOW CAPACITY OF QUEUE.
class Tamia_Lab03 {
private:
   int* queue;
   int Size;
   int front;
   int rear;
   int count;

public:
   // Constructor
   Tamia_Lab03(int size) : Size(size), front(0), rear(-1), count(0) {
      if (size <= 0) {
         throw invalid_argument("Queue size must be positive.");
      }
      queue = new int[Size];
   }

   // Destructor
   ~Tamia_Lab03() {
```

```
    delete[] queue;
}

// ELEMENT REAR SE INSERT KARAINGAIN.
void enqueue(int N) {
    if (isFull()) {
        throw overflow_error("Queue is overflowing.");
    }
    rear = (rear + 1) % Size;
    queue[rear] = N;
    count++;
}

// ELEMENT FRONT SE DELETE KARAINGAIN.
int dequeue() {
    if (isEmpty()) {
        throw underflow_error("Queue is empty.");
    }
    int value = queue[front];
    front = (front + 1) % Size;
    count--;
    return value;
}

// Agar koi bhi front element nhi h to error throw karega otherwise front return karega.
int Front() const {
    if (isEmpty()) {
        throw underflow_error("Queue is empty.");
    }
```

```
        return queue[front];
    }

    // Agar koi bhi rear element nhi h to error throw karega otherwise rear return karega.
    int Rear() const {
        if (isEmpty()) {
            throw underflow_error("Queue is empty.");
        }
        return queue[rear];
    }

   // HUMNE ISFULL,ISEMPTY OR SIZE K FUNCTION BANAYA HE OR USE ENQUE,DEQUE,FRONT,REAR MN
CALL KIYA H.
    bool isFull() const {
        return count == Size;
    }

    bool isEmpty() const {
        return count == 0;
    }

    int size() const {
        return count;
    }

    // Display all elements in the queue
    void display() const {
        if (isEmpty()) {
            cout << "Queue is empty" << endl;
```

```
            return;
        }
        int i = front;
        int totalElemets = count;
        while (totalElemets--) {
            cout << queue[i] << " ";
            i = (i + 1) % Size;
        }
        cout << endl;
    }
};

int main() {
    try {
        Tamia_Lab03 T(4);
        T.enqueue(153);
        T.enqueue(143);
        T.enqueue(567);
        T.enqueue(95);

        cout << "Queue after enqueuing 4 elements: ";
        T.display();

        T.dequeue();
        T.dequeue();
        cout << "Queue after dequeuing 2 element: ";
        T.display();

        T.enqueue(45);
```

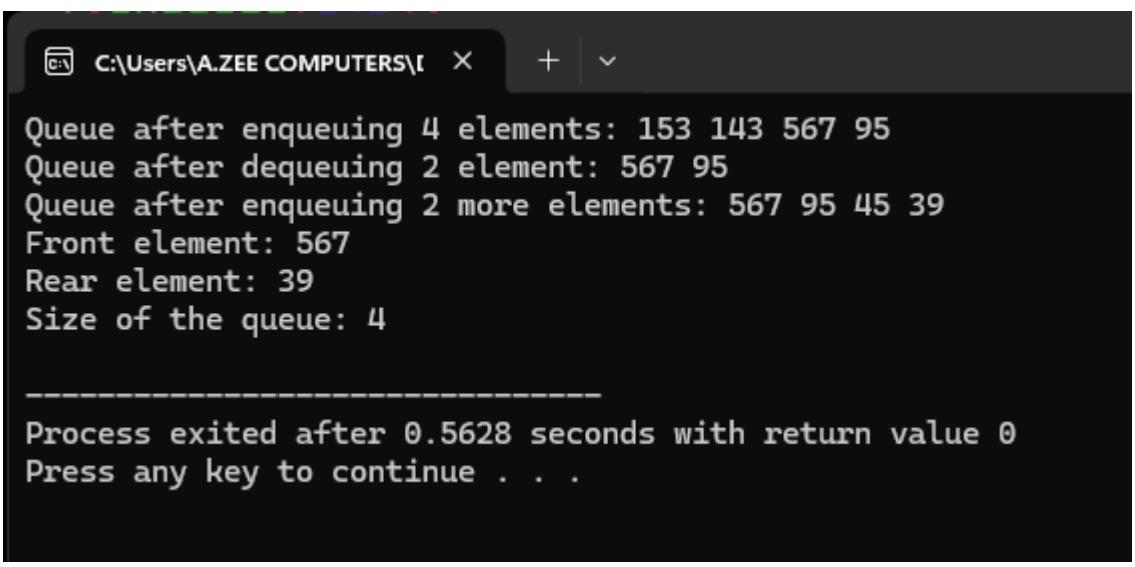
```
        T.enqueue(39);
        cout << "Queue after enqueuing 2 more elements: ";
        T.display();

        cout << "Front element: " << T.Front() << endl;
        cout << "Rear element: " << T.Rear() << endl;

        cout << "Size of the queue: " << T.size() << endl;
    } catch (const exception& e) {
        cout << e.what() << endl;
    }

    return 0;
}
```

OUTPUT



2. Define a class DEque with operations: insertFront(), insertRear(), deleteFront(), deleteRear(), and isEmpty(). Handle cases for inserting and removing elements from both ends. Add a function size() to return the current number of elements in the deque. Implement a display() function to print all the elements in the deque.

SOURCE CODE

```
#include <iostream>

#include <stdexcept>

using namespace std;


        // COUNT TO KEEP TRACK OF NO OF ELEMENTS, SIZE TO SHOW CAPACITY OF QUEUE.

class Tamia_Lab03 {

private:

   int* queue;

   int Size;

   int front;

   int rear;

   int count;


public:

   // CONSTRUCTOR

   Tamia_Lab03(int size) : Size(size), front(0), rear(-1), count(0) {

      if (size <= 0) {

         throw invalid_argument("Queue size must be positive.");

      }

      queue = new int[Size];

   }


        // DESTRUCTOR

   ~Tamia_Lab03() {

      delete[] queue;

   }


        // ELEMENT FRONT SE INSERT KARAINGAIN.
```

```
void insertFront(int n) {
    if (isFull()) {
        throw overflow_error("Queue is overflowing.");
    }
    rear = (rear + 1) % Size;
    queue[rear] = n;
    count++;
}


        // ELEMENT REAR SE INSERT KARAINGAIN.
        void insertRear(int N) {
    if (isFull()) {
        throw overflow_error("Queue is full.");
    }
    rear = (rear + 1) % Size;
    queue[rear] = N;
    count++;
}


        // ELEMENT FRONT SE DELETE KARAINGAIN.
    int deleteFront() {
    if (isEmpty()) {
        throw underflow_error("Queue is empty.");
    }
    int m = queue[front];
    front = (front + 1) % Size;
    count--;
    return m;
}
```

```
// ELEMENT BACK SE DELETE KARAINGAIN.
int deleteRear() {
    if (isEmpty()) {
        throw underflow_error("Queue is empty.");
    }
    int value = queue[rear];
    rear = (rear - 1 + Size) % Size;
    count--;
    return value;
}

// HUMNE ISFULL,ISEMPTY OR SIZE K FUNCTION BANAYA HE OR USE
ENQUE,DEQUE,FRONTELEMENT,REARELEMENT MN CALL KIYA H.
bool isEmpty() const {
    return count == 0;
}

bool isFull() const {
    return count == Size;
}

int size() const {
    return count;
}

// DISPLAY FUNCTION TO DISPLAY THE QUEUE.
void display() const {
    if (isEmpty()) {
```

```
        cout << "Queue is empty." << endl;
         return;
      }
      int i = front;
      int totalElemets = count;
      while (totalElemets--) {
         cout << queue[i] << " ";
         i = (i + 1) % Size;
      }
      cout << endl;
   }
};

int main() {
   try {
      Tamia_Lab03 T(7);
      T.insertFront(1);
      T.insertRear(31);
      T.insertFront(6);
      T.insertFront(724);
      T.insertRear(310);
      T.insertFront(76);
      T.insertRear(310);

      cout << "Queue after adding 7 elements from both front and rear: ";
      T.display();

      T.deleteRear();
      T.deleteFront();
```

```
                T.deleteRear();
    cout << "Queue after removing 3 elements at both ends: ";
    T.display();



    T.insertRear(890);
    T.insertFront(6);
    T.insertRear(91);

    cout << "Enqueing after inserting 3 more elements: ";
    T.display();

    cout << "Size of the deque: " << T.size() << endl;

  } catch (const exception& e) {
    cout << e.what() << endl;
  }

  return 0;
}
```

OUTPUT

```
C:\Users\A.ZEE COMPUTERS\[

Queue after adding 7 elements from both front and rear: 1 31 6 724 310 76 310
Queue after removing 3 elements at both ends: 31 6 724 310
Enqueing after inserting 3 more elements: 31 6 724 310 890 6 91
Size of the deque: 7

--------------------------------
Process exited after 0.5473 seconds with return value 0
Press any key to continue . . .
```

3.Design your implementation of the circular double-ended queue (DEque). Implement the

MyCircularDeque class:

- MyCircularDeque(int k) Initializes the deque with a maximum size of k. - boolean

insertFront() Adds an item at the front of Deque. Returns true if the operation is successful, or

false otherwise.

- boolean insertLast() Adds an item at the rear of Deque. Returns true if the operation is

successful, or false otherwise.

- boolean deleteFront() Deletes an item from the front of Deque. Returns true if the operation

is successful, or false otherwise.

- boolean deleteLast() Deletes an item from the rear of Deque. Returns true if the operation is

successful, or false otherwise.

- int getFront() Returns the front item from the Deque. Returns -1 if the deque is empty. -

int getRear() Returns the last item from Deque. Returns -1 if the deque is empty. - booleanisEmpty() Returns true if the deque is empty, or false otherwise. - boolean isFull()

Returns true if the deque is full, or false otherwise.

- Main Program:

Int main( ){

MyCircularDeque myCircularDeque = new MyCircularDeque(3);

myCircularDeque.insertLast(1); // return True

myCircularDeque.insertLast(2); // return True

myCircularDeque.insertFront(3); // return True

myCircularDeque.insertFront(4); // return False, the queue is full.

myCircularDeque.getRear(); // return 2

myCircularDeque.isFull(); // return True

myCircularDeque.deleteLast(); // return True

myCircularDeque.insertFront(4); // return True

myCircularDeque.getFront(); // return 4 }

SOURCE CODE

```
#include <iostream>
```

```
#include <stdexcept>

using namespace std;


class Tamia_Lab03 {

private:

    int* queue;

    int Size;

    int front;

    int rear;

    int count;


public:

    // Constructor

    Tamia_Lab03(int k) : Size(k), front(0), rear(-1), count(0) {

        if (k <= 0) {

            throw invalid_argument("Queue size must be positive.");

        }

        queue = new int[k];

    }

    // Destructor

    ~Tamia_Lab03() {

        delete[] queue;

    }

    bool insertFront(int n) {

        if (isFull()) {

            return false;

        }

        front = (front - 1 + Size) % Size;

        queue[front] = n;
```

```
        count++;
        return true;
    }
    bool insertLast(int n) {
        if (isFull()) {
            return false;
        }
        rear = (rear + 1) % Size;
        queue[rear] = n;
        count++;
        return true;
    }
    bool deleteFront() {
        if (isEmpty()) {
            return false;
        }
        front = (front + 1) % Size;
        count--;
        return true;
    }
    bool deleteLast() {
        if (isEmpty()) {
            return false;
        }
        rear = (rear - 1 + Size) % Size;
        count--;
        return true;
    }
    int getFront() const {
```

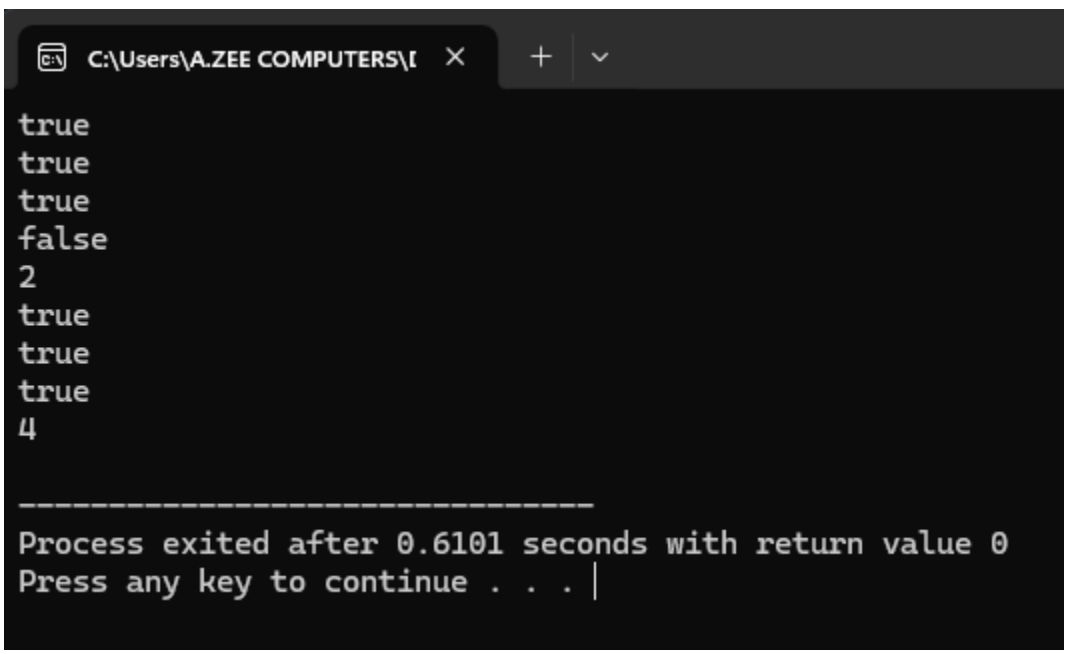
```
        if (isEmpty()) {
            return -1;
        }
        return queue[front];
    }
    int getRear() const {
        if (isEmpty()) {
            return -1;
        }
        return queue[rear];
    }
    bool isEmpty() const {
        return count == 0;
    }
    bool isFull() const {
        return count == Size;
    }
};
int main() {
    try {
        Tamia_Lab03 myCircularDeque(3);


        //In order to print true/false instead of 0/1 i have to add a manipulator known as boolalpha which
will print values as true/false.
        cout << boolalpha;
                        cout << myCircularDeque.insertLast(1) << endl;  // return True
        cout << myCircularDeque.insertLast(2) << endl;  // return True
        cout << myCircularDeque.insertFront(3) << endl; // return True
        cout << myCircularDeque.insertFront(4) << endl; // return False, the queue is full.
```

```
        cout << myCircularDeque.getRear() << endl;      // return 2
        cout << myCircularDeque.isFull() << endl;       // return True
        cout << myCircularDeque.deleteLast() << endl;   // return True
        cout << myCircularDeque.insertFront(4) << endl; // return True
        cout << myCircularDeque.getFront() << endl;     // return 4

    } catch (const exception& e) {
        cout << e.what() << endl;
    }
    return 0;
}
```

OUTPUT



4.There are n friends that are playing a game. The friends are sitting in a circle and are numbered from 1 to n in clockwise order. More formally, moving clockwise from the ith friend brings you to the (i+1)th friend for 1 <= i < n, and moving clockwise from the nth friend brings you to the 1st friend. The rules of the game are as follows:

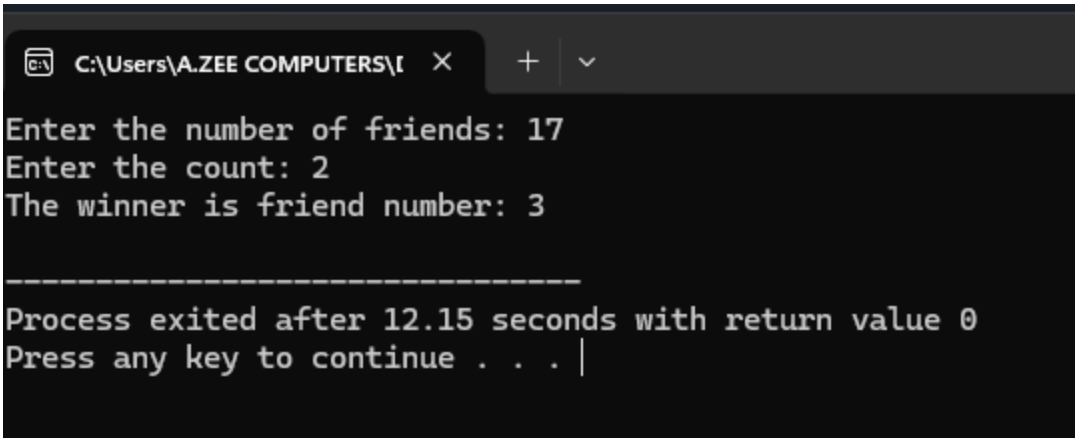
- Start at the 1st friend.
- Count the next k friends in the clockwise direction including the friend you started at. The counting wraps around the circle and may count some friends more than once.
- The last friend you counted leaves the circle and loses the game.
- If there is still more than one friend in the circle, go back to step 2 starting from the friend immediately clockwise of the friend who just lost and repeat.
- Else, the last friend in the circle wins the game.
- Given the number of friends, n, and an integer k, return the winner of the game.

SOURCE CODE

```
#include <iostream>
#include <queue>
using namespace std;
class Tamia_Lab03 {
private:
   int n;
   int k;
public:
   // Constructor
   Tamia_Lab03(int N, int C) : n(N), k(C) {}
   int findWinner() const {
      queue<int> q;
      // Initiazation
      for (int i = 1; i <= n; i++) {
         q.push(i);
      }
      // Jb tk queue 1 tk nhi ajata tb tk front element ko k-1 time rear mn shift karain gain
      while (q.size() > 1) {
         for (int i = 0; i < k - 1; ++i) {
            q.push(q.front());
```

```
            q.pop();
        }
        // K-th element ko eleminate kardein gain
        q.pop();
    }
    // Aakhir mn jo bache ga wohi winner hoga.
    return q.front();
  }
};
int main() {
  int n,k;
  cout << "Enter the number of friends: ";
  cin >> n;
  cout << "Enter the count: ";
  cin >> k;
  Tamia_Lab03 T(n,k);
  int w = T.findWinner();
  cout << "The winner is friend number: " << w << endl;
  return 0;
}
```

OUTPUT



5. For a stream of integers, implement a data structure that checks if the last k integers parsed in the stream are equal to value. Implement the DataStream class:

- DataStream(int value, int k) Initializes the object with an empty integer stream and the two integers value and k.
- boolean consec(int num) Adds num to the stream of integers. Returns true if the last k integers are equal to value, and false otherwise. If there are less than k integers, the condition does not hold true, so returns false.
- Main Program:

```
void main(){
DataStream dataStream = new DataStream(4, 3); //value = 4, k = 3
dataStream.consec(4); // Only 1 integer is parsed, so returns False.
dataStream.consec(4); // Only 2 integers are parsed.
// Since 2 is less than k, returns False.
dataStream.consec(4); // The 3 integers parsed are all equal to value, so returns True.
dataStream.consec(3); // The last k integers parsed in the stream are [4,4,3].
// Since 3 is not equal to value, it return False.
```

SOURCE CODE

```
#include <iostream>
#include <queue>
using namespace std;
class Tamia_Lab03{
private:
   int value;
   int k;
   queue<int> Q;
   int count;

public:
   // Constructor
```

```
    Tamia_Lab03(int value,int k):value(value),k(k),count(0){}


    bool consec(int N) {
        if (Q.size() == k) {
            int removed = Q.front();
            Q.pop();
            if (removed == value) {
                count--;
            }
        }
        Q.push(N);
        if (N == value) {
            count++;
        }
        return Q.size() == k && count == k;
    }
};
int main() {
    Tamia_Lab03 T(4, 3);
    cout << boolalpha;
    cout << T.consec(4) << endl;
    cout << T.consec(4) << endl;
    cout << T.consec(4) << endl;
    cout << T.consec(3) << endl;
    return 0;
}
```

OUTPUT

```
C:\Users\A.ZEE COMPUTERS\[

false
false
true
false

--------------------------------
Process exited after 0.4003 seconds with return value 0
Press any key to continue . . .
```