```
NAME – TABISH
ROLL NO. = 23103145
BATCH – CSE B
GROUP – G3
ASSIGNMENT-11
```

//Q1) Write a program to implement a queue using an array with basic operations: enqueue, dequeue, and display. Include checks for queue overflow and underflow.

```
#include <iostream>
using namespace std;
class Queue {
private:
  int* queue;
  int front, rear, size;
public:
  Queue(int size) {
   this->size = size;
    queue = new int[size];
   front = -1;
    rear = -1;}
  bool isEmpty() {
    return front == -1;}
  bool isFull() {
    return (rear + 1) % size == front;}
  void enqueue(int value) {
    if (isFull()) {
      cout << "Queue Overflow! Cannot
add element." << endl;
   }else{
      if (front == -1) {
        front = 0;
     }
      rear = (rear + 1) % size;
      queue[rear] = value;
   }}
  void dequeue() {
    if (isEmpty()) {
```

```
cout << "Oueue Underflow! Cannot
remove element." << endl;
   } else {
      if (front == rear) {
        front = rear = -1;
     } else {
       front = (front + 1) % size;
      }}}
 void display() {
    if (isEmpty()) {
      cout << "Queue is empty." << endl;
   } else {
      int i = front;
      while (i != rear) {
        cout << queue[i] << " ";
        i = (i + 1) \% \text{ size};
      cout << queue[rear] << endl;</pre>
   }}
  ~Queue() {
    delete[] queue;
 }};
int main() {
  Queue q(5);
  q.enqueue(10);
  q.enqueue(20);
  q.enqueue(30);
  q.enqueue(40);
  q.enqueue(50);
  q.enqueue(60); // This will cause
overflow
  cout << "Queue after enqueuing
elements: ";
  q.display();
  q.dequeue();
  cout << "Queue after dequeuing an
element: ";
  q.display();
  q.dequeue();
  cout << "Queue after dequeuing
another element: ";
  q.display();
  return 0;}
```

```
OUTPUT-
                                                       cout << "Oueue Underflow! Cannot
Queue Overflow! Cannot add element.
                                                 remove element." << endl;
Queue after enqueuing elements: 10 20
                                                       return;
30 40 50
Queue after dequeuing an element: 20 30
                                                     Node* temp = front;
40 50
                                                     front = front->next;
Queue after dequeuing another element:
                                                     if (front == nullptr) {
30 40 50
                                                       rear = nullptr; // If the queue
                                                 becomes empty, rear should also be null.
// 2. Write a program to implement a
                                                     }
queue using a linked list with basic
                                                     delete temp;}
operations: enqueue, dequeue, and
                                                   void display() {
display.
                                                     if (isEmpty()) {
#include <iostream>
                                                       cout << "Queue is empty." << endl;
using namespace std;
                                                       return;}
                                                     Node* temp = front;
class Node {
public:
                                                     while (temp != nullptr) {
 int data;
                                                       cout << temp->data << " ";
  Node* next:
                                                       temp = temp->next;}
 Node(int value) {
                                                     cout << endl;}
   data = value;
                                                   ~Queue() {
   next = nullptr; }};
                                                     while (!isEmpty()) {
class Queue {
                                                       dequeue();
private:
                                                     }}};
 Node* front;
                                                 int main() {
 Node* rear;
                                                   Queue q;
public:
                                                   q.enqueue(10);
  Queue() {
                                                   q.enqueue(20);
   front = nullptr;
                                                   q.enqueue(30);
   rear = nullptr;}
                                                   q.enqueue(40);
  bool isEmpty() {
                                                   q.enqueue(50);
   return front == nullptr;}
                                                   cout << "Queue after enqueuing
 void enqueue(int value) {
                                                 elements: ";
   Node* newNode = new Node(value);
                                                   q.display();
   if (rear == nullptr) {
                                                   q.dequeue();
     front = rear = newNode;
                                                   cout << "Queue after dequeuing an
   } else {
                                                 element: ";
     rear->next = newNode;
                                                   q.display();
     rear = newNode;
                                                   q.dequeue();
                                                   cout << "Queue after dequeuing
   }
 }
                                                 another element: ";
 void dequeue() {
                                                   q.display();
   if (isEmpty()) {
                                                   return 0;}
                                                 OUTPUT-
```

```
cout << "Queue Underflow! Cannot
Queue after enqueuing elements: 10 20
30 40 50
                                                    remove element." << endl;
Queue after dequeuing an element: 20 30
                                                         return;
                                                       }
Queue after dequeuing another element:
                                                       if (front == rear) {
30 40 50
                                                         front = rear = -1;
                                                       } else {
                                                         front = (front + 1) % size;
//3. Write a program to count the number
of elements in a queue implemented
                                                       }
using either an array or a linked list.
                                                     }
#include <iostream>
                                                     void display() {
using namespace std;
                                                       if (isEmpty()) {
                                                         cout << "Queue is empty." << endl;
class Queue {
private:
                                                         return;
  int* queue;
                                                       }
                                                       int i = front;
  int front, rear, size;
public:
                                                       while (i != rear) {
                                                         cout << queue[i] << " ";
  Queue(int size) {
   this->size = size;
                                                         i = (i + 1) \% \text{ size};
    queue = new int[size];
   front = -1;
                                                       cout << queue[rear] << endl;
    rear = -1;
                                                     int countElements() {
  }
  bool isEmpty() {
                                                       if (isEmpty()) {
    return front == -1;
                                                         return 0;
  }
                                                       }
  bool isFull() {
                                                       if (rear >= front) {
    return (rear + 1) % size == front;
                                                         return rear - front + 1;
  void enqueue(int value) {
                                                       return size - front + rear + 1;
    if (isFull()) {
      cout << "Queue Overflow! Cannot
                                                     ~Queue() {
add element." << endl;
                                                       delete[] queue;
      return;
                                                     }};
                                                   int main() {
   }
    if (front == -1) {
                                                      Queue q(5);
      front = 0;
                                                      q.enqueue(10);
   }
                                                      q.enqueue(20);
    rear = (rear + 1) \% size;
                                                      q.enqueue(30);
    queue[rear] = value;
                                                      q.enqueue(40);
                                                      q.enqueue(50);
                                                      cout << "Queue elements: ";
  void dequeue() {
    if (isEmpty()) {
                                                      q.display();
```

```
cout << "Number of elements in the
                                                     return front == nullptr;
queue: " << q.countElements() << endl;
                                                   }
  q.dequeue();
                                                   void enqueue(int value) {
  cout << "Queue after dequeuing an
                                                     Node* newNode = new Node(value);
element: ";
                                                     if (rear == nullptr) {
  q.display();
                                                       front = rear = newNode;
  cout << "Number of elements in the
                                                     } else {
queue after dequeue: " <<
                                                       rear->next = newNode;
q.countElements() << endl;</pre>
                                                       rear = newNode;
 return 0;
                                                     }}
                                                   void dequeue() {
}
                                                     if (isEmpty()) {
OUTPUT-
                                                       cout << "Queue Underflow! Cannot
Queue elements: 10 20 30 40 50
                                                  remove element." << endl;
Number of elements in the queue: 5
                                                       return;
Queue after dequeuing an element: 20 30
                                                     }
                                                     Node* temp = front;
Number of elements in the queue after
                                                     front = front->next;
dequeue: 4
                                                     if (front == nullptr) {
                                                       rear = nullptr;
//4. Write a program that implements a
                                                     }
queue and includes a peek operation to
                                                     delete temp;
display the front element of the queue
without removing it.
                                                   void display() {
#include <iostream>
                                                     if (isEmpty()) {
using namespace std;
                                                       cout << "Queue is empty." << endl;
class Node {
                                                       return;
public:
                                                     }
                                                     Node* temp = front;
 int data;
 Node* next;
                                                     while (temp != nullptr) {
                                                       cout << temp->data << " ";
 Node(int value) {
                                                       temp = temp->next;
    data = value;
                                                     }
   next = nullptr;
                                                     cout << endl;
 }};
class Queue {
                                                   }
private:
                                                   int peek() {
  Node* front;
                                                     if (isEmpty()) {
 Node* rear;
                                                       cout << "Queue is empty. Cannot
                                                  peek." << endl;
public:
                                                       return -1; // Return -1 if the queue is
  Queue() {
   front = nullptr;
                                                  empty
   rear = nullptr;
                                                     return front->data: // Return the data
                                                  of the front node
  bool isEmpty() {
```

```
q.push(s.top());
  ~Queue() {
                                                     s.pop();
   while (!isEmpty()) {
                                                   }}
     dequeue();
                                                 void displayQueue(queue<int> q) {
                                                   while (!q.empty()) {
   }
                                                     cout << q.front() << " ";
 }};
int main() {
                                                     q.pop();
  Queue q;
                                                   }
  q.enqueue(10);
                                                   cout << endl;}
  q.enqueue(20);
                                                 int main() {
                                                   queue<int> q;
  q.enqueue(30);
  q.enqueue(40);
                                                   q.push(10);
  cout << "Queue elements: ";
                                                   q.push(20);
  q.display();
                                                   q.push(30);
  cout << "Front element (peek): " <<
                                                   q.push(40);
q.peek() << endl;</pre>
                                                   q.push(50);
                                                   cout << "Original Queue: ";
  q.dequeue();
  cout << "Queue after dequeuing an
                                                   displayQueue(q);
element: ";
                                                   reverseQueue(q);
                                                   cout << "Reversed Queue: ";
  q.display();
  cout << "Front element (peek) after
                                                   displayQueue(q);
dequeue: " << q.peek() << endl;
                                                   return 0;}
  return 0;}
                                                 OUTPUT-
OUTPUT-
                                                 Original Queue: 10 20 30 40 50
                                                 Reversed Queue: 50 40 30 20 10
Queue elements: 10 20 30 40
Front element (peek): 10
Queue after dequeuing an element: 20 30
                                                 //6. Write a program to implement a
                                                 circular queue using an array. Implement
Front element (peek) after dequeue: 20
                                                 enqueue, dequeue, and display
                                                 operations, and handle circular indexing.
//5. Write a program to reverse the
                                                 #include <iostream>
elements of a queue using only stack
                                                 using namespace std;
                                                 class CircularQueue {
operations.
#include <iostream>
                                                 private:
#include <stack>
                                                   int* queue;
#include <queue>
                                                   int front, rear, size;
using namespace std;
                                                 public:
void reverseQueue(queue<int>& q) {
                                                   CircularQueue(int size) {
                                                     this->size = size;
  stack<int> s;
 while (!q.empty()) {
                                                     queue = new int[size];
   s.push(q.front());
                                                     front = -1;
                                                     rear = -1;
   q.pop();
 }
                                                   }
 while (!s.empty()) {
                                                   bool isFull() {
```

```
return (rear + 1) % size == front;
                                                     }
 }
                                                      cout << queue[rear] << endl;</pre>
  bool isEmpty() {
                                                    ~CircularQueue() {
    return front == -1;
                                                      delete[] queue;
 void enqueue(int value) {
                                                   }};
                                                  int main() {
   if (isFull()) {
                                                    CircularQueue q(5);
     cout << "Queue Overflow! Cannot
enqueue " << value << endl;
                                                    q.enqueue(10);
     return;
                                                    q.enqueue(20);
   }
                                                    q.enqueue(30);
    if (front == -1) {
                                                    q.enqueue(40);
     front = 0;
                                                    q.enqueue(50);
                                                    q.display();
    rear = (rear + 1) \% size;
                                                    q.enqueue(60);
    queue[rear] = value;
                                                    q.dequeue();
    cout << "Enqueued: " << value <<
                                                    q.dequeue();
endl;
                                                    q.display();
 }
                                                    q.enqueue(60);
 void dequeue() {
                                                    q.enqueue(70);
    if (isEmpty()) {
                                                    q.display();
     cout << "Queue Underflow! Cannot
                                                    return 0;}
dequeue" << endl;
                                                  OUTPUT-
     return;
                                                  Enqueued: 10
   }
                                                  Enqueued: 20
    int value = queue[front];
                                                  Enqueued: 30
    cout << "Dequeued: " << value <<
                                                  Enqueued: 40
endl;
                                                  Enqueued: 50
                                                  Queue elements: 10 20 30 40 50
   if (front == rear) {
     front = rear = -1;
                                                  Queue Overflow! Cannot enqueue 60
                                                  Dequeued: 10
   } else {
     front = (front + 1) % size;
                                                  Dequeued: 20
   }
                                                  Queue elements: 30 40 50
                                                  Enqueued: 60
 void display() {
                                                  Enqueued: 70
   if (isEmpty()) {
                                                  Queue elements: 30 40 50 60 70
     cout << "Queue is empty!" << endl;
                                                  //7. Write a program that finds the
     return;
   }
                                                  minimum element in a queue without
    cout << "Queue elements: ";
                                                  altering the queue's content. Display the
   int i = front;
                                                  minimum element without dequeuing it.
   while (i != rear) {
                                                  #include <iostream>
     cout << queue[i] << " ";
                                                  #include <queue>
     i = (i + 1) \% \text{ size};
                                                  using namespace std;
```

```
int findMin(queue<int>& q) {
                                                  cout << "Queue after finding the
                                                minimum element: ";
  if (q.empty()) {
   cout << "Queue is empty!" << endl;
                                                  displayQueue(q);
   return -1;
                                                  return 0;}
                                                OUTPUT-
                                                Original Queue: Queue elements: 10 20 5
  int minElement = q.front();
  queue<int> tempQueue;
  while (!q.empty()) {
                                                Minimum element in the queue: 5
   int current = q.front();
                                                Queue after finding the minimum
                                                element: Queue elements: 10 20 5 30 15
   q.pop();
   if (current < minElement) {</pre>
     minElement = current;
                                                //8. Write a program to merge two queues
                                                into a third queue. The resulting queue
   tempQueue.push(current);
                                                should contain elements from both
                                                queues in their original order
 while (!tempQueue.empty()) {
                                                #include <iostream>
   q.push(tempQueue.front());
                                                #include <queue>
   tempQueue.pop();
                                                using namespace std;
                                                void mergeQueues(queue<int>& q1,
 }
                                                queue<int>& q2, queue<int>&
 return minElement; }
void displayQueue(queue<int>& q) {
                                                mergedQueue) {
 if (q.empty()) {
                                                  while (!q1.empty()) {
   cout << "Queue is empty!" << endl;
                                                    mergedQueue.push(q1.front());
   return;
                                                    q1.pop();
 }
                                                  }
  cout << "Queue elements: ";
                                                  while (!q2.empty()) {
  queue<int> tempQueue = q;
                                                    mergedQueue.push(q2.front());
 while (!tempQueue.empty()) {
                                                    q2.pop();
   cout << tempQueue.front() << " ";
                                                  }
   tempQueue.pop();
                                                void displayQueue(queue<int>& q) {
 cout << endl;}
                                                  if (q.empty()) {
                                                    cout << "Queue is empty!" << endl;</pre>
int main() {
  queue<int> q;
                                                    return;
                                                  }
  q.push(10);
  q.push(20);
                                                  cout << "Queue elements: ";
  q.push(5);
                                                  queue<int> tempQueue = q;
                                                  while (!tempQueue.empty()) {
  q.push(30);
                                                    cout << tempQueue.front() << " ";</pre>
  q.push(15);
                                                    tempQueue.pop();
  cout << "Original Queue: ";
  displayQueue(q);
 int minElement = findMin(q);
                                                  cout << endl;
  cout << "Minimum element in the
queue: " << minElement << endl;
                                                int main() {
```

```
queue<int> q1, q2, mergedQueue;
                                                        return;
  q1.push(10);
                                                      }
  q1.push(20);
                                                      while (!stack1.empty()) {
  q1.push(30);
                                                        stack2.push(stack1.top());
  q2.push(40);
                                                        stack1.pop();
  q2.push(50);
                                                      }}
  q2.push(60);
                                                    if (!stack2.empty()) {
  cout << "First Queue: ";
                                                      cout << "Dequeued: " <<
  displayQueue(q1);
                                                stack2.top() << endl;
  cout << "Second Queue: ";
                                                      stack2.pop();
  displayQueue(q2);
                                                    }}
  mergeQueues(q1, q2, mergedQueue);
                                                  void display() {
  cout << "Merged Queue: ";
                                                    if (stack1.empty() && stack2.empty())
  displayQueue(mergedQueue);
                                                {
                                                      cout << "Queue is empty!" << endl;
  return 0;}
OUTPUT-
                                                      return;
First Queue: Queue elements: 10 20 30
                                                    }
Second Queue: Queue elements: 40 50
                                                    cout << "Queue elements: ";</pre>
60
                                                    stack<int> tempStack = stack2;
Merged Queue: Queue elements: 10 20
                                                    while (!tempStack.empty()) {
30 40 50 60
                                                      cout << tempStack.top() << " ";
                                                      tempStack.pop();
//9. Write a program to implement a
                                                    }
                                                    tempStack = stack1;
queue using two stacks. Implement
enqueue and dequeue operations.
                                                    stack<int> reversedStack;
#include <iostream>
                                                    while (!tempStack.empty()) {
#include <stack>
using namespace std;
                                                reversedStack.push(tempStack.top());
class QueueUsingTwoStacks {
                                                      tempStack.pop();
private:
                                                    }
 stack<int> stack1;
                                                    while (!reversedStack.empty()) {
  stack<int> stack2;
                                                      cout << reversedStack.top() << " ";</pre>
                                                      reversedStack.pop();
public:
 void enqueue(int value) {
                                                    cout << endl;
   stack1.push(value);
                                                  }};
   cout << "Enqueued: " << value <<
                                                int main() {
endl;
                                                  QueueUsingTwoStacks q;
 }
                                                  q.enqueue(10);
 void dequeue() {
                                                  q.enqueue(20);
   if (stack2.empty()) {
                                                  q.enqueue(30);
     if (stack1.empty()) {
                                                  q.display();
       cout << "Queue Underflow!
                                                  q.dequeue();
Cannot dequeue" << endl;
                                                  q.display();
```

```
q.enqueue(40);
                                                   q.pop();
 q.enqueue(50);
                                                 }
 q.display();
                                                 while (!tempQueue.empty()) {
 q.dequeue();
                                                   q.push(tempQueue.front());
 q.dequeue();
                                                   tempQueue.pop();
 q.display();
 q.dequeue();
                                                 return sum;
 q.dequeue();
 q.display();
                                               void displayQueue(queue<int>& q) {
 return 0;}
                                                 if (q.empty()) {
OUTPUT-
                                                   cout << "Queue is empty!" << endl;
Enqueued: 10
                                                   return;
Enqueued: 20
                                                 }
Enqueued: 30
                                                 cout << "Queue elements: ";
Queue elements: 10 20 30
                                                 queue<int> tempQueue = q;
Dequeued: 10
                                                 while (!tempQueue.empty()) {
Queue elements: 20 30
                                                   cout << tempQueue.front() << " ";</pre>
Enqueued: 40
                                                   tempQueue.pop();
Enqueued: 50
                                                 }
Queue elements: 20 30 40 50
                                                 cout << endl;
Dequeued: 20
Dequeued: 30
                                               int main() {
Queue elements: 40 50
                                                 queue<int> q;
Dequeued: 40
                                                 q.push(10);
Dequeued: 50
                                                 q.push(20);
Queue is empty!
                                                 q.push(30);
                                                 q.push(40);
//10. Write a program to find and display
                                                 q.push(50);
the sum of all elements in a queue
                                                 cout << "Original Queue: ";
without modifying the queue's content.
                                                 displayQueue(q);
#include <iostream>
                                                 int sum = sumQueue(q);
                                                 cout << "Sum of all elements in the
#include <queue>
                                                queue: " << sum << endl;
using namespace std;
int sumQueue(queue<int>& q) {
                                                 cout << "Queue after finding the sum: ";
 if (q.empty()) {
                                                 displayQueue(q);
   cout << "Queue is empty!" << endl;
                                                 return 0;}
                                                OUTPUT-
   return 0;
 }
                                                Original Queue: Queue elements: 10 20
                                                30 40 50
 int sum = 0;
                                                Sum of all elements in the queue: 150
 queue<int> tempQueue;
 while (!q.empty()) {
                                                Queue after finding the sum: Queue
                                                elements: 10 20 30 40 50
   int current = q.front();
   sum += current;
   tempQueue.push(current);
```

```
//11. Write a program that uses a queue
                                                   using namespace std;
to check if a string is a palindrome.
                                                   void reverseFirstKElements(queue<int>&
Enqueue each character, then dequeue
                                                   q, int k) {
to verify the order.
                                                     if (k > q.size() || k <= 0) {
#include <iostream>
                                                       cout << "Invalid value of k." << endl;
#include <queue>
                                                       return;
#include <string>
                                                     }
using namespace std;
                                                     stack<int>s;
bool isPalindrome(const string& str) {
                                                     for (int i = 0; i < k; i++) {
  queue<char> q;
                                                       s.push(q.front());
 for (char ch: str) {
                                                       q.pop();
    q.push(ch);
                                                     }
                                                     while (!s.empty()) {
 }
 int length = str.length();
                                                       q.push(s.top());
 for (int i = length-1; i > = 0; i - -) {
                                                       s.pop();
   if (q.front() != str[i]) {
     return false;
                                                     int size = q.size();
   }
                                                     for (int i = 0; i < size - k; i++) {
    q.pop();
                                                       q.push(q.front());
                                                       q.pop();
 return true; }
                                                     }}
int main() {
                                                   void displayQueue(queue<int>& q) {
  string str;
                                                     if (q.empty()) {
                                                       cout << "Queue is empty!" << endl;
  cout << "Enter a string: ";
  cin >> str;
                                                       return;
 if (isPalindrome(str)) {
                                                     }
    cout << "The string is a palindrome."
                                                     cout << "Queue elements: ";
<< endl;
                                                     queue<int> tempQueue = q;
                                                     while (!tempQueue.empty()) {
 } else {
    cout << "The string is not a
                                                       cout << tempQueue.front() << " ";
palindrome." << endl;
                                                       tempQueue.pop();
 }
                                                     cout << endl;}
 return 0;}
OUTPUT-
                                                   int main() {
Enter a string: TANISH
                                                     queue<int> q;
The string is not a palindrome.
                                                     int k;
                                                     q.push(1);
//12. Write a program that takes a queue
                                                     q.push(2);
and an integer k as input and reverses the
                                                     q.push(3);
first k elements of the queue, leaving the
                                                     q.push(4);
rest in the same order.
                                                     q.push(5);
#include <iostream>
                                                     cout << "Enter the value of k: ";
#include <queue>
                                                     cin >> k;
#include <stack>
                                                     cout << "Original Queue: ";
```

```
displayQueue(q);
reverseFirstKElements(q, k);
cout << "Queue after reversing the first"
<< k << " elements: ";
displayQueue(q);
return 0;}
OUTPUT-
Enter the value of k: 5
Original Queue: Queue elements: 1 2 3 4
5
Queue after reversing the first 5
elements: Queue elements: 5 4 3 2 1
```