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 << "Queue 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) % size;
      }
      cout << queue[rear] << endl;
    }}
  ~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-

```
Queue Overflow! Cannot add element.
Queue after enqueuing elements: 10 20 30 40 50
Queue after dequeuing an element: 20 30 40 50
Queue after dequeuing another element: 30 40 50
```

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

```
Queue after enqueuing elements: 10 20 30 40 50
Queue after dequeuing an element: 20 30 40 50
Queue after dequeuing another element: 30 40 50
```

```
//3. Write a program to count the number of elements in a queue implemented using either an array or a linked list.
#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;
      return;
    }
    if (front == -1) {
      front = 0;
    }
    rear = (rear + 1) % size;
    queue[rear] = value;
  }
  void dequeue() {
    if (isEmpty()) {
      cout << "Queue Underflow! Cannot remove element." << endl;
      return;
    }
    if (front == rear) {
      front = rear = -1;
    } else {
      front = (front + 1) % size;
    }
  }
  void display() {
    if (isEmpty()) {
      cout << "Queue is empty." << endl;
      return;
    }
    int i = front;
    while (i != rear) {
      cout << queue[i] << " ";
      i = (i + 1) % size;
    }
    cout << queue[rear] << endl;
  }
  int countElements() {
    if (isEmpty()) {
      return 0;
    }
    if (rear >= front) {
      return rear - front + 1;
    }
    return size - front + rear + 1;
  }
  ~Queue() {
    delete[] queue;
  }};
int main() {
  Queue q(5);
  q.enqueue(10);
  q.enqueue(20);
  q.enqueue(30);
  q.enqueue(40);
  q.enqueue(50);
  cout << "Queue elements: ";
  q.display();
```

```
   cout << "Number of elements in the
queue: " << q.countElements() << endl;
   q.dequeue();
   cout << "Queue after dequeuing an
element: ";
   q.display();
   cout << "Number of elements in the
queue after dequeue: " <<
q.countElements() << endl;
   return 0;
}
```

```
OUTPUT-
Queue elements: 10 20 30 40 50
Number of elements in the queue: 5
Queue after dequeuing an element: 20 30
40 50
Number of elements in the queue after
dequeue: 4
```

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

```
  }
  ~Queue() {
    while (!isEmpty()) {
      dequeue();
    }
  }};
int main() {
  Queue q;
  q.enqueue(10);
  q.enqueue(20);
  q.enqueue(30);
  q.enqueue(40);
  cout << "Queue elements: ";
  q.display();
  cout << "Front element (peek): " <<
q.peek() << endl;
  q.dequeue();
  cout << "Queue after dequeuing an
element: ";
  q.display();
  cout << "Front element (peek) after
dequeue: " << q.peek() << endl;
  return 0;}
OUTPUT-
Queue elements: 10 20 30 40
Front element (peek): 10
Queue after dequeuing an element: 20 30
40
Front element (peek) after dequeue: 20

//5. Write a program to reverse the
elements of a queue using only stack
operations.
#include <iostream>
#include <stack>
#include <queue>
using namespace std;
void reverseQueue(queue<int>& q) {
  stack<int> s;
  while (!q.empty()) {
    s.push(q.front());
    q.pop();
  }
  while (!s.empty()) {
    q.push(s.top());
    s.pop();
  }}
void displayQueue(queue<int> q) {
  while (!q.empty()) {
    cout << q.front() << " ";
    q.pop();
  }
  cout << endl;}
int main() {
  queue<int> q;
  q.push(10);
  q.push(20);
  q.push(30);
  q.push(40);
  q.push(50);
  cout << "Original Queue: ";
  displayQueue(q);
  reverseQueue(q);
  cout << "Reversed Queue: ";
  displayQueue(q);
  return 0;}
OUTPUT-
Original Queue: 10 20 30 40 50
Reversed Queue: 50 40 30 20 10

//6. Write a program to implement a
circular queue using an array. Implement
enqueue, dequeue, and display
operations, and handle circular indexing.
#include <iostream>
using namespace std;
class CircularQueue {
private:
  int* queue;
  int front, rear, size;
public:
  CircularQueue(int size) {
    this->size = size;
    queue = new int[size];
    front = -1;
    rear = -1;
  }
  bool isFull() {
        return (rear + 1) % size == front;
    }
    bool isEmpty() {
        return front == -1;
    }
    void enqueue(int value) {
        if (isFull()) {
            cout << "Queue Overflow! Cannot enqueue " << value << endl;
            return;
        }
        if (front == -1) {
            front = 0;
        }
        rear = (rear + 1) % size;
        queue[rear] = value;
        cout << "Enqueued: " << value << endl;
    }
    void dequeue() {
        if (isEmpty()) {
            cout << "Queue Underflow! Cannot dequeue" << endl;
            return;
        }
        int value = queue[front];
        cout << "Dequeued: " << value << endl;
        if (front == rear) {
            front = rear = -1;
        } else {
            front = (front + 1) % size;
        }
    }
    void display() {
        if (isEmpty()) {
            cout << "Queue is empty!" << endl;
            return;
        }
        cout << "Queue elements: ";
        int i = front;
        while (i != rear) {
            cout << queue[i] << " ";
            i = (i + 1) % size;
        }
        cout << queue[rear] << endl;
    }
    ~CircularQueue() {
        delete[] queue;
    }};
int main() {
    CircularQueue q(5);
    q.enqueue(10);
    q.enqueue(20);
    q.enqueue(30);
    q.enqueue(40);
    q.enqueue(50);
    q.display();
    q.enqueue(60);
    q.dequeue();
    q.dequeue();
    q.display();
    q.enqueue(60);
    q.enqueue(70);
    q.display();
    return 0;}
```

OUTPUT-

```
Enqueued: 10
Enqueued: 20
Enqueued: 30
Enqueued: 40
Enqueued: 50
Queue elements: 10 20 30 40 50
Queue Overflow! Cannot enqueue 60
Dequeued: 10
Dequeued: 20
Queue elements: 30 40 50
Enqueued: 60
Enqueued: 70
Queue elements: 30 40 50 60 70
```

//7. Write a program that finds the minimum element in a queue without altering the queue’s content. Display the minimum element without dequeuing it.

```
#include <iostream>
#include <queue>
using namespace std;
```

```
int findMin(queue<int>& q) {
  if (q.empty()) {
    cout << "Queue is empty!" << endl;
    return -1;
  }
  int minElement = q.front();
  queue<int> tempQueue;
  while (!q.empty()) {
    int current = q.front();
    q.pop();
    if (current < minElement) {
      minElement = current;
    }
    tempQueue.push(current);
  }
  while (!tempQueue.empty()) {
    q.push(tempQueue.front());
    tempQueue.pop();
  }
  return minElement;  }
void displayQueue(queue<int>& q) {
  if (q.empty()) {
    cout << "Queue is empty!" << endl;
    return;
  }
  cout << "Queue elements: ";
  queue<int> tempQueue = q;
  while (!tempQueue.empty()) {
    cout << tempQueue.front() << " ";
    tempQueue.pop();
  }
  cout << endl;}
int main() {
  queue<int> q;
  q.push(10);
  q.push(20);
  q.push(5);
  q.push(30);
  q.push(15);
  cout << "Original Queue: ";
  displayQueue(q);
  int minElement = findMin(q);
  cout << "Minimum element in the
queue: " << minElement << endl;
  cout << "Queue after finding the
minimum element: ";
  displayQueue(q);
  return 0;}
```

OUTPUT-

```
Original Queue: Queue elements: 10 20 5
30 15
Minimum element in the queue: 5
Queue after finding the minimum
element: Queue elements: 10 20 5 30 15
```

```
//8. Write a program to merge two queues
into a third queue. The resulting queue
should contain elements from both
queues in their original order
#include <iostream>
#include <queue>
using namespace std;
void mergeQueues(queue<int>& q1,
queue<int>& q2, queue<int>&
mergedQueue) {
  while (!q1.empty()) {
    mergedQueue.push(q1.front());
    q1.pop();
  }
  while (!q2.empty()) {
    mergedQueue.push(q2.front());
    q2.pop();
  }
}
void displayQueue(queue<int>& q) {
  if (q.empty()) {
    cout << "Queue is empty!" << endl;
    return;
  }
  cout << "Queue elements: ";
  queue<int> tempQueue = q;
  while (!tempQueue.empty()) {
    cout << tempQueue.front() << " ";
    tempQueue.pop();
  }
  cout << endl;
}
int main() {
  queue<int> q1, q2, mergedQueue;
  q1.push(10);
  q1.push(20);
  q1.push(30);
  q2.push(40);
  q2.push(50);
  q2.push(60);
  cout << "First Queue: ";
  displayQueue(q1);
  cout << "Second Queue: ";
  displayQueue(q2);
  mergeQueues(q1, q2, mergedQueue);
  cout << "Merged Queue: ";
  displayQueue(mergedQueue);
  return 0;}
```

OUTPUT-

```
First Queue: Queue elements: 10 20 30
Second Queue: Queue elements: 40 50
60
Merged Queue: Queue elements: 10 20
30 40 50 60
```

//9. Write a program to implement a queue using two stacks. Implement enqueue and dequeue operations.

```
#include <iostream>
#include <stack>
using namespace std;
class QueueUsingTwoStacks {
private:
  stack<int> stack1;
  stack<int> stack2;

public:
  void enqueue(int value) {
    stack1.push(value);
    cout << "Enqueued: " << value <<
endl;
  }
  void dequeue() {
    if (stack2.empty()) {
      if (stack1.empty()) {
        cout << "Queue Underflow!
Cannot dequeue" << endl;
        return;
      }
      while (!stack1.empty()) {
        stack2.push(stack1.top());
        stack1.pop();
      }}
    if (!stack2.empty()) {
      cout << "Dequeued: " <<
stack2.top() << endl;
      stack2.pop();
    }}
  void display() {
    if (stack1.empty() && stack2.empty())
{
      cout << "Queue is empty!" << endl;
      return;
    }
    cout << "Queue elements: ";
    stack<int> tempStack = stack2;
    while (!tempStack.empty()) {
      cout << tempStack.top() << " ";
      tempStack.pop();
    }
    tempStack = stack1;
    stack<int> reversedStack;
    while (!tempStack.empty()) {

reversedStack.push(tempStack.top());
      tempStack.pop();
    }
    while (!reversedStack.empty()) {
      cout << reversedStack.top() << " ";
      reversedStack.pop();
    }
    cout << endl;
  }};
int main() {
  QueueUsingTwoStacks q;
  q.enqueue(10);
  q.enqueue(20);
  q.enqueue(30);
  q.display();
  q.dequeue();
  q.display();
```

```
    q.enqueue(40);
    q.enqueue(50);
    q.display();
    q.dequeue();
    q.dequeue();
    q.display();
    q.dequeue();
    q.dequeue();
    q.display();
    return 0;}
```

OUTPUT-

```
Enqueued: 10
Enqueued: 20
Enqueued: 30
Queue elements: 10 20 30
Dequeued: 10
Queue elements: 20 30
Enqueued: 40
Enqueued: 50
Queue elements: 20 30 40 50
Dequeued: 20
Dequeued: 30
Queue elements: 40 50
Dequeued: 40
Dequeued: 50
Queue is empty!
```

//10. Write a program to find and display the sum of all elements in a queue without modifying the queue's content.

```
#include <iostream>
#include <queue>
using namespace std;
int sumQueue(queue<int>& q) {
    if (q.empty()) {
        cout << "Queue is empty!" << endl;
        return 0;
    }
    int sum = 0;
    queue<int> tempQueue;
    while (!q.empty()) {
        int current = q.front();
        sum += current;
        tempQueue.push(current);
        q.pop();
    }
    while (!tempQueue.empty()) {
        q.push(tempQueue.front());
        tempQueue.pop();
    }
    return sum;
}
void displayQueue(queue<int>& q) {
    if (q.empty()) {
        cout << "Queue is empty!" << endl;
        return;
    }
    cout << "Queue elements: ";
    queue<int> tempQueue = q;
    while (!tempQueue.empty()) {
        cout << tempQueue.front() << " ";
        tempQueue.pop();
    }
    cout << endl;
}
int main() {
    queue<int> q;
    q.push(10);
    q.push(20);
    q.push(30);
    q.push(40);
    q.push(50);
    cout << "Original Queue: ";
    displayQueue(q);
    int sum = sumQueue(q);
    cout << "Sum of all elements in the
queue: " << sum << endl;
    cout << "Queue after finding the sum: ";
    displayQueue(q);
    return 0;}
```

OUTPUT-

```
Original Queue: Queue elements: 10 20
30 40 50
Sum of all elements in the queue: 150
Queue after finding the sum: Queue
elements: 10 20 30 40 50
```

```
//11. Write a program that uses a queue
to check if a string is a palindrome.
Enqueue each character, then dequeue
to verify the order.
#include <iostream>
#include <queue>
#include <string>
using namespace std;
bool isPalindrome(const string& str) {
  queue<char> q;
  for (char ch : str) {
    q.push(ch);
  }
  int length = str.length();
  for (int i = length-1; i>=0; i--) {
    if (q.front() != str[i]) {
      return false;
    }
    q.pop();
  }
  return true;  }
int main() {
  string str;
  cout << "Enter a string: ";
  cin >> str;
  if (isPalindrome(str)) {
    cout << "The string is a palindrome."
<< endl;
  } else {
    cout << "The string is not a
palindrome." << endl;
  }
  return 0;}
OUTPUT-
Enter a string: TANISH
The string is not a palindrome.
```

```
//12. Write a program that takes a queue
and an integer k as input and reverses the
first k elements of the queue, leaving the
rest in the same order.
#include <iostream>
#include <queue>
#include <stack>
using namespace std;
void reverseFirstKElements(queue<int>&
q, int k) {
  if (k > q.size() || k <= 0) {
    cout << "Invalid value of k." << endl;
    return;
  }
  stack<int> s;
  for (int i = 0; i < k; i++) {
    s.push(q.front());
    q.pop();
  }
  while (!s.empty()) {
    q.push(s.top());
    s.pop();
  }
  int size = q.size();
  for (int i = 0; i < size - k; i++) {
    q.push(q.front());
    q.pop();
  }}
void displayQueue(queue<int>& q) {
  if (q.empty()) {
    cout << "Queue is empty!" << endl;
    return;
  }
  cout << "Queue elements: ";
  queue<int> tempQueue = q;
  while (!tempQueue.empty()) {
    cout << tempQueue.front() << " ";
    tempQueue.pop();
  }
  cout << endl;}
int main() {
  queue<int> q;
  int k;
  q.push(1);
  q.push(2);
  q.push(3);
  q.push(4);
  q.push(5);
  cout << "Enter the value of k: ";
  cin >> k;
  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
```