Data Structures & Algorithms

55412

Muhammad Anas
9-20-2024

Lab 05

1) Implement Class Queue, its data members, and operation listed below.

• Queue ()

A non-parameterized constructor that creates an empty queue. Where should the front and rear of an empty queue point to?

• enqueue ()

Inserts the element at the rear of queue.

• dequeue ()

Removes the element from the front of queue.

• isEmpty ()

Returns True if queue is empty else returns False.

• display ()

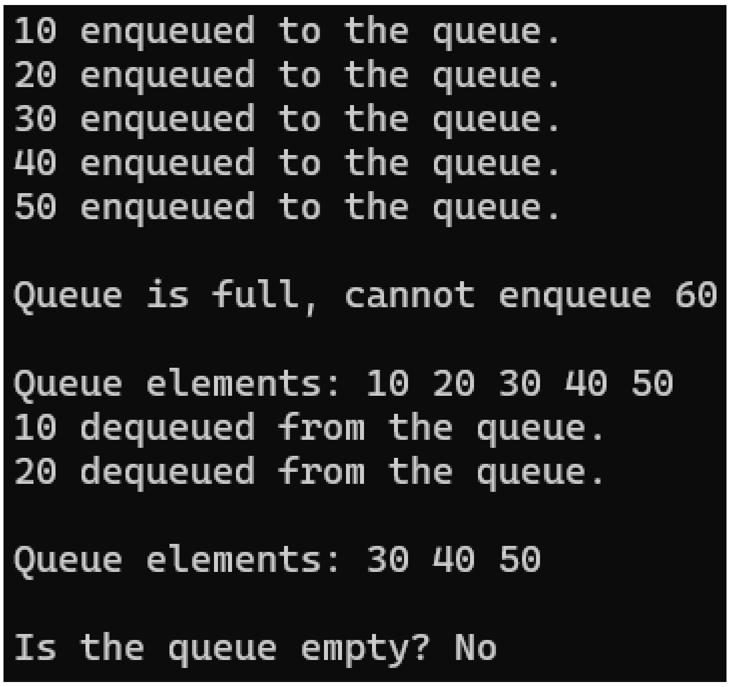
Display all the elements of Queue

```
#include <iostream>
using namespace std;
class Queue
{
private:
    int front, rear;
    int size;
    int* arr;
public:
    // Non-parameterized constructor to create an empty queue
    Queue()
    {
        size = 5;
        front = rear = -1;
        arr = new int[size];
    }
    void enqueue(int value)
    {
        if (rear == size - 1)
        {
            cout << "\n Queue is full, cannot enqueue " << value << endl;
            return;
        }
        if (front == -1)
        {
            front = 0;  // Reset front if queue was empty
        }
        arr[++rear] = value;  // Increment rear and add the new element
        cout << " " << value << " enqueued to the queue." << endl;
    }
    // remove an element from the front of the queue
    void dequeue()
    {
        if (isEmpty())
        {
```

```
        cout << "\n Queue is empty, cannot dequeue." << endl;
        return;
    }
    cout << " " << arr[front] << " dequeued from the queue." << endl;
    front++;
    if (front > rear)
    {
        // if the front passes the rear, reset the queue
        front = rear = -1;
    }
}
bool isEmpty()
{
    return (front == -1);
}
void display()
{
    if (isEmpty())
    {
        cout << " Queue is empty." << endl;
        return;
    }
    cout << "\n Queue elements: ";
    for (int i = front; i <= rear; i++)
    {
        cout << arr[i] << " ";
    }
    cout << endl;
}
```

```
    ~Queue()
    {
        delete[] arr;
    }
};
int main()
{
    Queue q;  // Creating an empty queue
    q.enqueue(10);
    q.enqueue(20);
    q.enqueue(30);
    q.enqueue(40);
    q.enqueue(50);
    q.enqueue(60);  // This should show "Queue is full"
    q.display();
    q.dequeue();
    q.dequeue();
    q.display();  // Display current queue elements again
    cout << "\n Is the queue empty? " << (q.isEmpty() ? "Yes" : "No") << endl;
    return 0;
}
```

Output


```
10 enqueued to the queue.
20 enqueued to the queue.
30 enqueued to the queue.
40 enqueued to the queue.
50 enqueued to the queue.

Queue is full, cannot enqueue 60

Queue elements: 10 20 30 40 50
10 dequeued from the queue.
20 dequeued from the queue.

Queue elements: 30 40 50

Is the queue empty? No
```

---

2) Take a single string as input. Using this input string, you have to create multiple queues in which each queue will comprise of separate word appeared in input string. At the end, you will again concatenate all queues to a single queue and return it to user.

Example:

String = "Data Structure and Algorithms"

Q1 = D → a → t → a

Q2 = S → t → r → u → c → t → u → r → e

Q3 = a → n → d

Q4 = A → l → g → o

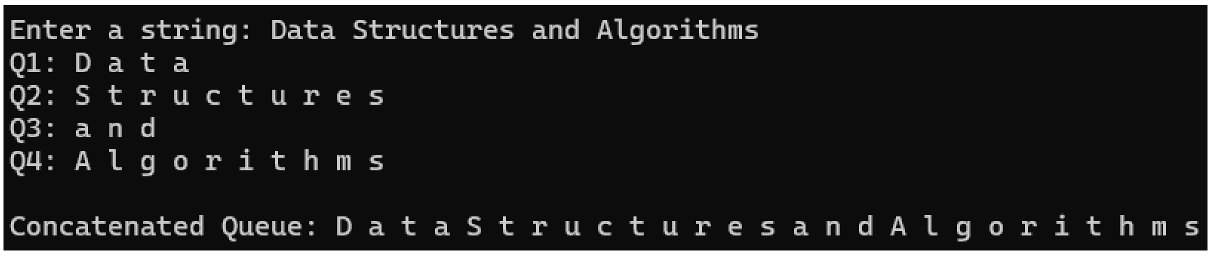
At the end concatenate all queues.

Q1→Q2→Q3→Q4

```
#include <iostream>
#include <string>
using namespace std;
#define MAX 100 // Define a maximum size for our arrays (queues)
class Queue
{
   char arr[MAX];
   int front, rear;
public:
   Queue()
   {
      front = -1;
      rear = -1;
   }
   bool isEmpty()
   {
      return front == -1 || front > rear;
   }
   // Enqueue a character to the rear of the queue
   void enqueue(char value)
   {
      if (rear == MAX - 1)
      {
         cout << " Queue is full!" << endl; // Overflow condition
      }
      else
      {
         if (front == -1) front = 0; // Initialize front if first element
         arr[++rear] = value;
```

```
    }
}
char dequeue()
{
    if (isEmpty())
    {
        cout << " Queue is empty!" << endl;
        return '\0'; // Return null character if queue is empty
    }
    return arr[front++];
}
void display()
{
    if (isEmpty())
    {
        cout << " Queue is empty" << endl;
        return;
    }
    for (int i = front; i <= rear; i++)
    {
        cout << arr[i] << " ";
    }
    cout << endl;
}
// Concatenate another queue into this one
void concatenate(Queue& q)
{
    while (!q.isEmpty())
    {
            enqueue(q.dequeue());
        }
    }
};
int main()
{
    string input;
    cout << "\n Enter a string: ";
    getline(cin, input); // Get input string
    Queue queues[10];
    int queueIndex = 0;
    Queue currentQueue;
    for (int i = 0; i < input.length(); i++)
    {
        if (input[i] != ' ')
        {
            currentQueue.enqueue(input[i]); // Enqueue each character of the word
        }
        else
        {
            if (!currentQueue.isEmpty())
            {
                queues[queueIndex++] = currentQueue; // Store the queue of the word
                currentQueue = Queue();              // Reset the current queue for the
next word
            }
        }
    }
    // Handle the last word if any
    if (!currentQueue.isEmpty())
    {
        queues[queueIndex++] = currentQueue;
    }
    for (int i = 0; i < queueIndex; i++)
    {
        cout << " Q" << i + 1 << ": ";
        queues[i].display();
    }
    Queue resultQueue = queues[0];
    for (int i = 1; i < queueIndex; i++)
    {
        resultQueue.concatenate(queues[i]);
    }
    cout << "\n Concatenated Queue: ";
    resultQueue.display();
    return 0;
}
```

Output



```
Enter a string: Data Structures and Algorithms
Q1: D a t a
Q2: S t r u c t u r e s
Q3: a n d
Q4: A l g o r i t h m s

Concatenated Queue: D a t a S t r u c t u r e s a n d A l g o r i t h m s
```