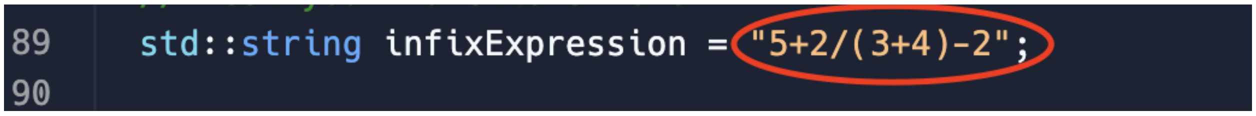
Q1 Assumptions:

1. This code assumes that all operands are single digit integers.
2. All infix inputs are assumed to include every operator, even if not mathematically required.
    - Ex: NOT 2(2+3), but 2*(2+3)
3. The code does not handle spaces in expressions and expects a space-free input. It treats expressions without spaces between operands and operators.
4. The code assumes that the primary concern is unbalanced parentheses, and it reports an error message when encountered. It does not provide detailed information about other potential errors, such as invalid operators.
5. The code assumes that the input infix expression consists of valid mathematical expressions with numbers, operators (+, -, *, /, %), and parentheses. It may not handle other special characters or expressions outside the standard mathematical syntax.
6. The code assumes that the input infix expression consists of valid mathematical expressions with numbers, operators (+, -, *, /, %), and parentheses. It may not handle other special characters or expressions outside the standard mathematical syntax.
7. The code assumes standard operator precedence rules. It correctly converts the infix expression to postfix notation, considering the precedence of operators (*, /, %) over (+, -).
8. The code does not explicitly handle empty expressions. It assumes that expressions passed to the functions are non-empty.

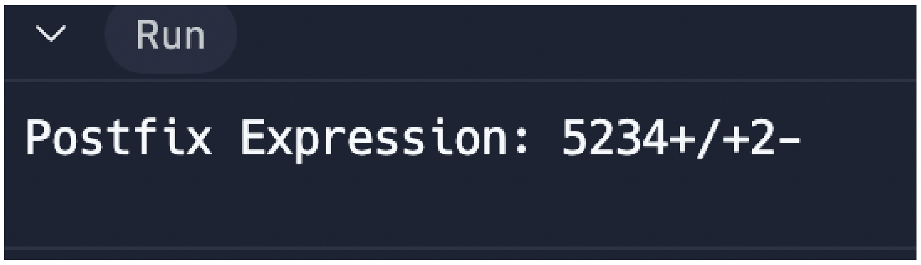
Q1 Instructions:

1. Open Assignment_3_Q1.cpp and Assignment_3_Q1.h, and make sure they are in the same working directory
2. On Line 89 of Assignment_3_Q1.cpp, enter in your own infix expression, or leave the expression already between the quotes.
3. Click the RUN button, the result will be displayed in the console.



```
89    std::string infixExpression = "5+2/(3+4)-2";
90
```

Q1 Outputs:



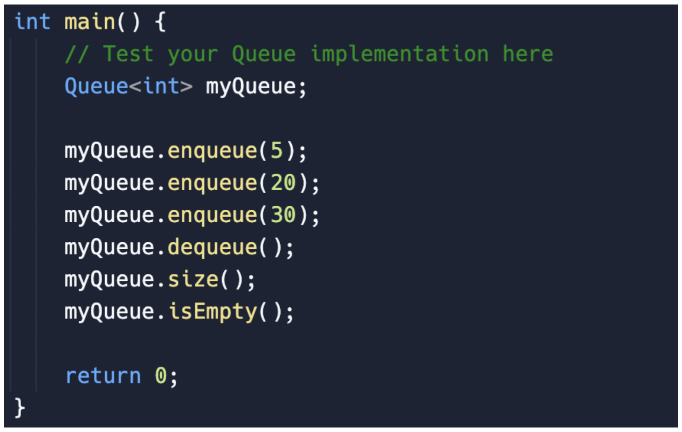
```
Postfix Expression: 5234+/+2-
```

Q2 Assumptions:

1. This code assumes that all inputs for the queue will be of int type.

2. The code assumes that queue operations (enqueue, dequeue, front, isEmpty, and size) are performed in a sequential manner, following the intended order of operations. Deviating from this order might lead to unexpected results.
3. The code assumes that the Queue class will be instantiated with a valid data type (T) for the template. It is designed to work with any data type, and operations depend on the properties of that type.

Q2 Instructions:

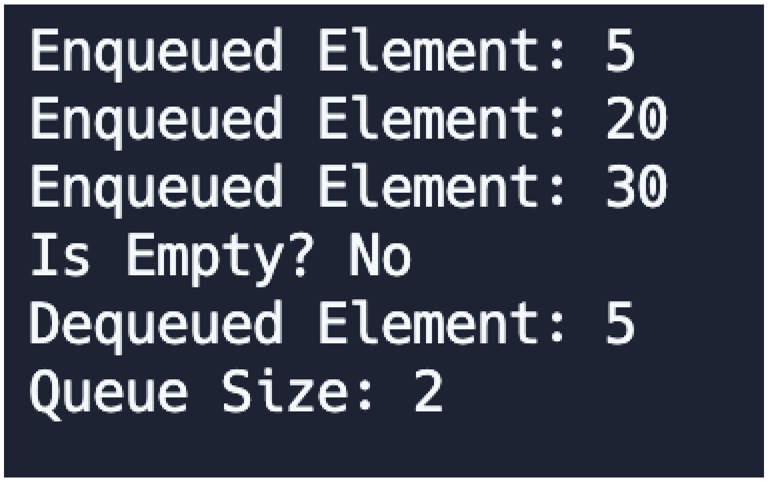
1. Open Assignment_3_Q2.cpp and Assignment_3_Q2.h in the same working directory.
2. Scroll down to the main() function in Assignment_3_Q2.cpp, and create an instance of the class Queue, for example… Queue<int> myQueue;
3. From main(), call whatever functions desired, being, myQueue.enqueue(int), myQueue.dequeue(), myQueue.isEmpty(), myQueue.size().
4. 

```
int main() {
    // Test your Queue implementation here
    Queue<int> myQueue;

    myQueue.enqueue(5);
    myQueue.enqueue(20);
    myQueue.enqueue(30);
    myQueue.dequeue();
    myQueue.size();
    myQueue.isEmpty();

    return 0;
}
```

Q2 Outputs:



```
Enqueued Element: 5
Enqueued Element: 20
Enqueued Element: 30
Is Empty? No
Dequeued Element: 5
Queue Size: 2
```