Data Structures and Fundamentals of Programming

Problem #1

In C++ implement a generic class, called Queue<T>, that uses a single-linked list implementation. This should implement the queue ADT. It should be generic on the type of the data to be stored. It must be implemented using a dynamically allocated linked list with all allocation and de-allocation done explicitly. Give all class definitions and implement the following for Queue:

- Default constructor
- Destructor
- Copy-constructor
- Assignment operator
- enqueue(T) – takes an parameter of type T and adds it to the end of the queue
- T dequeue() – removes a node from the front of the queue

Note: Your implementation can NOT use STL or any other libraries (standard or otherwise).

Problem #2

Implement a function, to convert a fully parenthesized infix expression into the corresponding postfix expression. You can assume the expression is correct. The infix expression will be passed into the function as a character array (null terminating) or string. The binary operators +, -, *, / with standard precedence are to be supported. You do not need to support unary operators. Additionally, you can assume that a generic class stack<T> exists with push and pop defined as normal and you may also use a built in string class.

```c
char expr1[] = "(2*{(3+7)-10})";
string expr2 = "(16*{(4+23)-7})";
```

Problem #3

Implement the function int G(int m, int n) defined by

\[
G(m, n) = \begin{cases} 
  n + 1, & \text{if } m = 0 \\
  G(m-1,1), & \text{if } m > 0 \text{ and } n = 0 \\
  G(m-1,G(m,n-1)), & \text{if } m > 0 \text{ and } n > 0 
\end{cases}
\]

(a) First, using system recursion.
(b) Second, using only the ADT stack (i.e without using system recursion, vectors, queues, maps, etc).
Problem #4

Given the following:

```c
struct cellT {
    int val;
    cellT *next;
};

bool contains(cellT *list, cellT *sub);
```

Write a function that given two linked lists will determine whether the second list is a subsequence of the first. To be a subsequence, every value of the second must appear within the first list and in the same order, but there may be additional values interspersed in the first list. A list contains itself; the NULL list is contained in any list.

Here are some examples:

<table>
<thead>
<tr>
<th>list</th>
<th>sub</th>
<th>Contains(list, sub)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1→4→2→9</td>
<td>1→4</td>
<td>true</td>
</tr>
<tr>
<td>1→4→2→9</td>
<td>9→4</td>
<td>false</td>
</tr>
<tr>
<td>1→4→2→9</td>
<td>1→9</td>
<td>true</td>
</tr>
<tr>
<td>1→4→2→9</td>
<td>1→1→4</td>
<td>false</td>
</tr>
<tr>
<td>1→4→2→9</td>
<td>2→9→10</td>
<td>false</td>
</tr>
</tbody>
</table>