Name:

1. (5 points) What is the definition of a recursive function?

2. (5 points) Write a recursive Fibonacci function, where the Fibonacci function is mathematically defined as

$$f(0) = 0$$
$$f(1) = 1$$
$$f(n) = f(n-1) + f(n-2)$$

3. (5 points) Rewrite the function from Problem 1 to use iteration and an explicit stack.

4. (10 points) Assume a linked list implementation identical to Homework 1, and write a method named `find` which accepts a value as an argument, and returns the index of the first occurrence of the argument in the list. The function should return -1 if no such index exists.

```
int LinkedListNode::find(T item)
{




}
```

5. (10 points) For each pair of operations, circle the operation that has the worst performance. Assume that the linked list is identical to our Homework 1 implementation.

   A. Get the i-th element of a vector.
   B. Get the i-th element of a linked list.
   C. These operations have the same performance.

   A. `push_front` on a vector.
   B. `push_front` on a linked list.
   C. These operations have the same performance.

   A. Binary search a vector.
   B. Binary search a linked list.
   C. These operations have the same performance.

   A. `push_back` on a linked list.
   B. `push_front` on a linked list.
   C. These operations have the same performance.

   A. `pop` on a stack.
   B. `push` on a stack.
   C. These operations have the same performance.

6. (5 points) The following program prints the string "baz". Is the mystery object acting like a stack or a queue?

```
MysteryObject<char*> myStery;
myStery.in("foo");
myStery.in("bar");
myStery.in("foo");
myStery.in("baz");
cout << myStery.out();
```

A. Stack. B. Queue. C. Neither.

7. (5 points) Assuming the default copy constructor, draw `foo` and `bar` at the end of the following operations.

```
LinkedList<int> foo(100);
foo.push_back(300);
foo.push_back(200);
LinkedList<int> bar = foo;
foo.push_back(800);
bar.push_back(400);
```

8. (20 points) Consider the following infix expression.

$$(3 - 1) * 6/(3 * 3) - (4 + 4)$$

(a) Convert the infix expression into the equivalent prefix expression.

(b) Convert the infix expression into the equivalent postfix expression.

(c) Using the postfix expression from part (a), draw a stack performing each step of the computation of the result.

1. If the current token is a number, push it onto the stack. (draw the stack.)
2. If the current token is an operator, pop two operands off of the stack and perform the correct operation. (draw the stack.)
3. Push the result onto the stack. (draw the stack.)

9. (10 points) Assume a linked list implementation identical to Homework 1. Write a **function** which takes a linked list (node) as an argument, and returns the reversed linked list (nodes). This can be implemented recursively or iteratively. You may assume that this function has access to the private and protected members of the `LinkedListNode` class.

   For example if the items of the linked list argument are $1, 2, 3, 4$ the returned list will contain the items $4, 3, 2, 1$.

   Note: While it is not necessary to use any of the functions we defined as part of Homework 1, you are free to do so.

```
LinkedListNode<T>* reverse(LinkedListNode<T>* list)
{




}
```

10. (10 points) Given `array1` and `array2` write two `for` loops to insert `array1` into `array2` at position 2. The first loop will move `E, F, G` to the correct positions, and the second will insert `C, D` (as shown in the diagram).

    Do not use any hard-coded values; assume that the variables `size1` and `size2` are in scope, and set to the sizes of `array1` and `array2` respectively, and that the variable `offset` is in scope and set to 2.