CS 240 Fall 2017 **Quiz 2** **Name: _____KEY_____**

(1) [2 pts] What, if anything, is wrong with the following code?

```
Node *makeNode(int value) {
      Node *newnode;
      Node myNode(value); // assume the value constructor sets the node's
                          // next field to null, and its data to value
      newnode = &myNode;
      return newnode;
}
```

The function returns a pointer to an object (myNode) that resides on the stack. That space will be reused for the next function call, and so that space will no longer contain the node value.

(2) Consider a container data structure that allows insertion and removal at the front and at the back of the container. In other words, it supports `insertFront()`, `removeFront()`, `insertBack()`, and `removeBack()`.

[1 pt] If you expect a very large number of elements in your container, and lots of updates (i.e. insertions and removals), then for efficiency would you use a *linked list* or a *C++ array* for your implementation?

Linked list! You can only insert into one end of an array without moving everything over. You can insert into either end of a linked list. (There is a case to be made for allocating a larger array than you need and putting the elements in the "middle" so that you could usually insert and remove from either end without moving everything, but that's a lot to explain and you could still run out of space on either side, eventually.)

[1 pt] What private *data members* would you put in a C++ class to implement this container efficiently? Declare them in C++ below.

```
class DoubleEndedContainer {
      private:  // show data members only!
            Node *head;  // or "first"… whatever
            Node *tail;  // or "last".  This assumes a DLL by the way.

      // other stuff would go below… but don't show anything else…
};
```

(3) [2 pts] Which of the following is possible with C++ templates. Circle the letters for all that are possible:

(a) To write *functions* that can operate on different *user-defined types* of data, such as `FBLUser` and `Fraction`.
(b) To write *functions* that can operate on different *primitive* C++ data types, such as strings, ints, floats, and doubles.
(c) To write *classes* that can contain different *user-defined types* of data, such as `FBLUser` and `Fraction`.
(d) To write *classes* that can contain different *primitive* C++ data types, such as strings, ints, floats, and doubles.
(ALL ARE POSSIBLE)

(4) [2 pts] Which of the following best describes a container's iterator in the C++ STL.
(a) An iterator is an integer that ranges from 0 through the number of elements in the container.
(b) An iterator is a function or algorithm that "visits" every element in the container, whenever it is called.
(c) An iterator is a C++ pointer type that points to the first element of the container.
(d) An iterator is intended to refer to some element in the container, or to a value that means we have reached the end of the container.

(5) [2 pts] What are the two parts of a recursive function? Name them, or (if you're not sure of the names) describe them briefly, or both.
Base case: *the case where the function knows and simply returns the answer without making a call*
Recursive case: *the case where the function is called one or more times with a "smaller problem"*