

Assignment #4 -**Name:****Date:**

- Points per question: 4
-

1. The node class is defined as follows:

```
1. class node {
2.     public: // TYPEDEF
3.         typedef double value_type;
4.
5.         // CONSTRUCTOR
6.         node(const value_type& init_data = value_type(), node* init_link = NULL) {
7.             data_field = init_data;
8.             link_field = init_link; }
9.
10.        // Member functions to set the data and link fields:
11.        void set_data(const value_type& new_data) { data_field = new_data; }
12.        void set_link(node* new_link) { link_field = new_link; }
13.
14.        // Constant member function to retrieve the current data:
15.        value_type data() const { return data_field; }
16.
17.        // Two slightly different member functions to retrieve the current link:
18.        const node* link() const { return link_field; }
19.        node* link() { return link_field; }
20.
21.    private:
22.        value_type data_field;
23.        node* link_field;
24.};
```

Implement the following function. (Note: *No toolkit function is available.* Only the node class is available.)

```
1. void list_copy (const node* source_ptr, node*& head_ptr, node*& tail_ptr)
2. // Precondition: source_ptr is the head pointer of a linked list.
3. // Postcondition: head_ptr and tail_ptr are the head and tail pointers for a new list that
4. // contains the same items as the list pointed to by source_ptr
5. {
```

2. Please justify why the linked list toolkit functions are not member functions of the node class?

3. In the following function, why `cursor` has been declared as a `const` variable? What happens if you change it to a non-`const` variable?

```
1. size_t list_length (const node* head_ptr)
2. // Precondition: head_ptr is the head pointer of a linked list.
3. // Postcondition: The value returned is the number of nodes in the // linked list.
4. {
5.     const node* cursor;
6.     size_t answer;
7.     answer = 0;
8.     for (cursor = head_ptr; cursor != NULL; cursor = cursor -> link())
9.         ++answer;
10.    return answer;
11. }
```

4. We are interested in implemented the back and forward arrow functionality in a mobile app. When back arrow is touched, the previous screen is shown, and when forward arrow is pressed (if applicable), then the forward screen is shown. Please justify what type of data structure you would use.

5. Why does our node class have two versions of the link member function?
 - A. One is public, the other is private.
 - B. One is to use with a const pointer, the other with a regular pointer.
 - C. One returns the forward link, the other returns the backward link.
 - D. One returns the data, the other returns a pointer to the next node.

6. What are the *iterator invalidation rules* for a data structure that stores items in a *linked list*?

7. What are the *iterator invalidation rules* for STL's *vector* class?

8. What are the features of a *random access iterator*? Present the name of two STL data structures that offer random access iterators.

|