COEN 79

Object-Oriented Programming and Advanced Data Structures

Assignment #4 -

Name: **Date:**

- Points per question: 4

1. The node class is defined as follows:

```cpp
class node {
    public: // TYPEDEF
        typedef double value_type;

    // CONSTRUCTOR
    node(const value_type& init_data = value_type(), node* init_link = NULL) {
            data_field = init_data;
            link_field = init_link;  }

    // Member functions to set the data and link fields:
    void set_data(const value_type& new_data) { data_field = new_data; }
    void set_link(node* new_link) { link_field = new_link;  }

    // Constant member function to retrieve the current data:
    value_type data() const { return data_field; }

    // Two slightly different member functions to retreive the current link:
    const node* link() const { return link_field; }
    node* link() { return link_field; }

    private:
        value_type data_field;
        node* link_field;
};
```

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

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

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?

```
size_t list_length (const node* head_ptr)
// Precondition: head_ptr is the head pointer of a linked list.
// Postcondition: The value returned is the number of nodes in the // linked list.
{
        const node* cursor;
        size_t answer;
        answer = 0;
        for (cursor = head_ptr; cursor != NULL; cursor = cursor - > link())
            ++answer;
        return answer;
}
```

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.

9. The bag class is defined as follows:

```cpp
template < class Item >
class bag {
public:
    // TYPEDEFS and MEMBER CONSTANTS
    typedef Item value_type;
    typedef std::size_t size_type;

    static const size_type DEFAULT_CAPACITY = 30;

    typedef bag_iterator < Item > iterator;

    bag(size_type initial_capacity = DEFAULT_CAPACITY);
    bag(const bag& source);
    ~bag();

    // MODIFICATION MEMBER FUNCTIONS
    // ...

    iterator begin();
    iterator end();

private:
    Item* data;          // Pointer to partially filled dynamic array
    size_type used;      // How much of array is being used
    size_type capacity;  // Current capacity of the bag
};
```

- This class implements the following functions to create iterators:

```
template <class Item>
typename bag <Item>::iterator bag<Item>::begin() {
    return iterator(capacity, used, 0, data);
}

template <class Item>
typename bag<Item>::iterator bag<Item>::end() {
    return iterator(capacity, used, used, data);
}
```

- Please complete the implementation of the following iterator:

```
template < class Item >
class bag_iterator: public std::iterator < std::forward_iterator_tag, Item >
{
private:
    size_type capacity;
    size_type used;
    size_type current;
    Item* data;

Public:
    typedef std::size_t size_type;
    bag_iterator(size_type capacity, size_type used, size_type current, Item* data) {
        this -> capacity = capacity;
        this -> used = used;
        this -> current = current;
        this -> data = data;
    }

    Item& operator* () const {

    }

    bag_iterator& operator++()   // Prefix ++
    {

    }

    bag_iterator operator++(int) // Postfix ++
    {

    }

    bool operator == (const bag_iterator other) const {

    }

    bool operator != (const bag_iterator other) const {

    }

```