CSCI 3110 Review for test 1

Topics covered:

- Function parameter: value, reference, constant reference
- Function return type: value, reference, constant reference
- Enumeration type
- Array of struct
    - Operations with array of struct
        - Define struct type, declare an array of the struct type
        - Read data into array of struct
        - Linear Search within the array
        - Sort data stored in the array
        - Print data
- C++ class
    - Private vs public data and methods
    - constructor and destructors
    - Accessor and mutator methods
    - constant data of a class
    - const method
    - static data of a class
        - access static data of a class
    - static method of a class
        - invoking static method of a class
    - Header file, implementation file, client program (main program)
    - Using class in the main program
        - Invoking method of a class in the client program
    - Array of class objects
        - Access methods of individual object in the array
- Conditional compilation (#ifndef, #define, #endif)
- Overloaded operators:
    - overloaded operator as an independent client function,
    - overloaded operator in a class
    - overload ==, <=, >=, !=, =, <<, >> operators
    - overloaded [], ++, = (assignment operator),
    - copy constructor
    - friend function (>> (extraction operator) and << (insertion operator))
- Pointer and dynamic memory allocation
    - Declare pointer
    - * dereference and & address operators
    - Assign pointer to point to memory location
    - Define pointer to point to struct type data
        - Access struct members through pointer
    - Use new operator to allocate memory space
    - Use delete operator to deallocate memory space
    - Recognize and avoid memory leak

- List Class:
    - array based implementation
    - pointer based implementation
    - common methods: insert, delete, retrieve, length
    - overloaded operators, copy constructor
- Templated function
    - Declare, define and call templated function
    - Pass function as a parameter to a templated function
- Templated class
    - How to define a templated class
    - How to use a templated class in the client program
    - Templated struct node definition in templated pointer based listclass implementation
- Standard Template Library
    - vector, deque, and list
        - Declare objects of these STL types
        - Use the methods defined in these STL types
        - Declare and use iterators of these STL types
    - stack and queue
        - Usage (declare, usage with the methods, as well as the use of iterator)
        - Applications of stack and queue in various problems

Example Test Questions for Practice:

1. Short questions:
    a. What is the difference between having a function return type being "const dataType &" and it being "dataType&"?

    b. When is it necessary to make a function or a class a friend of another class?

    c. List all who can access:
    the private member of a class
    protected member of a class
    public member of a class

    d. What is difference between static data and a regular data of a class?

2.
    a. The "queueClass" shown below is not complete. Add member functions and friend functions to the class interface at appropriate places as directed by the comments given in the program.
    b. Show the implementation of the overloaded != operator
    c. Show the implementation of the overloaded << operator

    For questions (b) and (c), assuming the queue is implemented using a singular, circular linked list with backPtr pointing at the last node in the list

class queueClass

```
{
public:
        queueClass();
        queueClass (const queueClass& q);
        ~queueClass();

        int length() const;    // returns the number of items in queue

        void Enqueue(const dataType & dataItem);
        // dataItem is added at the end of the queue

        void Dequeue();
        // item at the front of the queue is removed

        void Front(dataType & frontData) const;
        // item at the front of the queue is returned
```

// Write a prototype for an overloaded != operators here

```
        // two queues are not equal if the number of items in the queues are different
        // (the class does not have data "size" that keeps track of the # of items in the queue)
```

// **Write a prototype for the overloaded assignment operator here:**

// Write the prototype for the overloaded input and output operators here

```
private:
        struct nodeTye{
                dataType    item;
                nodeType    *next;
        };
        nodeType* head;
        nodeType* backPtr;
};
```

3. Convert the queueClass given in problem 2 into a class template:
   a. show the header file for the class template
   b. Show the implementation of the **overloaded assignment operator** for the queueClass class template.
   c. Show C++ code that creates two objects: floatQ and charQ, of queueClass, such that floatQ represents a queue of float values, and charQ represents a queue of character values.

4. Short answer questions
   - Show the statement(s) that dynamically creates an object of the *Sensor* class and stores it in a variable you declare. You may assume the class exists and has a default constructor.
   - Using the object you dynamically allocated above, call the *calibrate* member function of the *Sensor* class (which takes no arguments).
   - Given the class Candidate which represents someone running for a political office, overload the > operator based on the class data member votesReceived. Provide the complete function including its type, function name (including the scope resolution operator), parameter list, and function body.

- Assume you have two objects (c1 and c2) of the Candidate class described in the previous question. Using these objects, give an expression that makes use of the operator you overloaded above.

5. Multiple Choice Questions:
   - A non-default destructor is needed _______.
     1. always (for correct operation)
     2. optionally
     3. only if a developer wants to be more explicit than the default
     4. when there will be more than one instance of a class
     5. when class data members are dynamically allocated

   - Consider the following code:
     ```
     int x = 15;   // located in memory at 0x1236
     int* intPtr;  // located in memory at 0x1240
     intPtr = &x;
     cout << *intPtr;
     ```

     What is its output?
     1. an error
     2. 15
     3. a random number
     4. 0x1236
     5. 0x1240
   - The following function prototype is for the ____________.
     ```
     UberObject & operator=(UberObject & rhs);
     ```
     1. copy assignment
     2. default constructor
     3. move constructor
     4. move assignment
     5. copy constructor

6. True/False Questions:

   - The role of constructors is to initialize objects and data members.
     1. True
     2. False

   - A non-static member function may only refer to non-static data members of the class.
     1. True
     2. False

   - All type parameters in the template prefix must appear in the body of a function template.
     1. True
     2. False