**Assignment**

**March 10, 2019**

1. Implement a bit vector class whose public interface is provided in the **BitVector.h** header file.
   1. Use any appropriate private fields and methods to support the implementation of the public methods.
   2. The public and private member functions of the BitVector class should be implemented in a separate C++ source file named **BitVector.cpp**. You need to follow standard coding guidelines (similar to our C style guide) and make your implementation as robust as possible and guard against invalid inputs.
   3. The test suite for the BitVector class implementation is provided in the file **BitVectorTest.cpp**. You can compile and link this test suite against your BitVector implementation using the g++ compiler on Linux:  
       g++ -g -o BitVectorTest BitVector.cpp BitVectorTest.cpp  
      If this step succeeds, you are then ready to execute the test suite by running the generated **BitVectorTest** executable. If any of the test cases fail, the program will terminate with an assertion failure indicating the function and line number of the failing test case. You will then need to debug your implementation, preferably using the GNU debugger gdb. Repeat the fix-compile-link-execute cycle until your program passes all the test cases and you get the success message “ALL OK”.
   4. Make sure the BitVectorTest program completes within 10 seconds of running time. Once you have debugged your implementation and fixed all bugs, you can then compile your program with optimization enabled:  
       g++ -g -O3 -o BitVectorTest BitVector.cpp BitVectorTest.cpp  
      This should greatly increase your program speed but make sure it completes in the stipulated time. If it is taking much longer than that, then it’s time to optimize some of your time-consuming routines.  
      **NOTE**: Do NOT change anything in the test suite file BitVectorTest.cpp unless absolutely necessary and that too, only during the program debugging phase.
2. Use the BitVector class to solve the problem of determining whether a string has all unique characters (Solution 4 in the slides). You should not make any assumptions about what type of characters can occur in the input string; you can only assume that the character set is ASCII.

**Deliverables:**

1. **BitVector.h**: with the private section of the class filled in as appropriate.
2. **BitVector.cpp**: implementation of the public and private member functions of the BitVector class.
3. **unique\_chars.cpp**: program to solve the unique chars problem using the BitVector class.