3110 Open Lab Requirements

Software:

Assignments for this class will be developed using Visual C++ in the Microsoft Visual Studio 2008 environment. This software is installed on all departmental lab machines and on the terminal server "shemp.cs.mtsu.edu." Because of the Computer Science Department's agreement with Microsoft, you may check out Visual Studio installation disks from the Department and install it on your personal computer for academic use.

Turn In:

Create a **zipped** folder containing all source files and project files except the debug folder. The folder should be named **YourLastNameLab#.zip** where YourLastName is replaced by your last name and # is replaced by the lab number of the project. For example, if I was turning in lab 1, I would place my project in HankinsLab1.zip. Your project will be rebuilt and your source files will be printed (.h, .cpp). I will use this printout for grading style requirements. **This printout should look nice: I will use a courier font of size 10 and portrait printout.** You should have **NO wraparound** problems when this type of printout is used.

You should make sure your project works on our lab machines if you are using a different version of Microsoft Visual Studio. I will rebuild your project and run it on my machine. If I have any trouble compiling or executing your project, I will notify you by e-mail.

You will turn your project in to the appropriate dropbox using Desire2Learn (http://elearn.mtsu.edu). Your user name and password is the same as for Pipeline.

Due Date:

- Each project is due at midnight (actually 11:59 pm) of the due date.
- The maximum grade that you can earn on a project will decrease by 10% per class day. Weekend counts as 1 day and holidays will be excluded.
- No assignments will be accepted after graded assignments have been returned.
- Late labs will not be accepted after three class days.

Academic Honesty:

- All lab assignments must be done on an individual basis – **COLLABORATION** on lab projects is **NOT PERMITTED** unless a team project is assigned.
- The penalty for unauthorized collaboration will range from a grade of zero for an individual assignment to a failing grade for the course. There should be no discussion of laboratory assignments with other students. The following situations are not allowed and will be treated as cheating
    - Showing to or acquiring from other students any materials related to assignments such as source code and documentation.
    - Helping or seeking help from other students to debug programs. However, you may get help from a lab assistant or instructor.
- All source code must be **original.** Only when special permission is given should you use code from another source. When permission is given, you should reference the source of the code in your documentation.

- If you use the computers in the lab, never save a copy of your source code or project on one of these computers. If someone finds that code and copies it, I have no way of knowing whether you collaborated or not. It is best to use a thumb drive or some other media on which to do your work if you are using lab computers.

Correctness (at least 60% of the total lab grade)

- Programs, which produce syntax errors, will **not** be graded and will automatically receive a grade of 0.
- Programs that do not link will **NOT** be graded and will automatically receive a grade of 0.
- The lab must meet written specifications and be correct. This is the most important aspect of any computer program. It must do what it is required to do and the computation must be correct. Therefore you should carefully read the specifications. Make sure your program meets these specifications and produces correct results.
- Partial credit will be given for a partially working program. If your program is not fully functional, then **in your limitations (see documentation)**, you must describe what parts of the program are not working. Your output should in some way demonstrate that you have the portion working that you are sure works.
- Check your answers for correctness.

Documentation (up to 20% of the lab grade)

- Documentation should appear at the beginning of each source code file (.cpp or .h)
  - Your application program (the file containing the main method) should include a heading with the following information:

    FILE: **the name of the file containing the code**
    PROGRAMMER: **Your name**
    CLASS: **CSCI 3110**
    DUE DATE: **date in the form Wednesday, 1/14/2010**
    INSTRUCTOR: **Dr. Hankins**

    The heading should also include the following **separate** sections in **complete correct sentences**

    a short description of the program's purpose
    a description of user input or file to be used
    a description of expected results or output file
    a description of how to use the program
    limitations (explain what part of the program is or is not working)

  - Each .h and .cpp file should contain a header with

    FILE: **file name**
    PROGRAMMER: **programmer's name**

    And a section written in complete correct sentences that describes what is contained in the file (i.e. a description of the class that is defined or implemented in the file)

  - Each .h file should contain brief documentation for each data field and method.
  - Begin each function prototype **and** definition with comments identifying the purpose of the function. Put function prototypes before *main*, and put function definitions after *main*.

- Use identifiers in function prototypes (not just types) -- in general a function prototype will be identical to the first line(s) of the corresponding definition except for the extra semicolon in the prototype.
- All local variables must be briefly described.
- Each class member should be briefly described.
- Comments should appear prior to each loop statement. This comment identifies the purpose of the loop statement.
- Comments should appear at the beginning of each branch of every if-else statement. This comment specifies what is true when control flow comes to this branch.
- It is not required or recommended to comment EACH statement.

Style (up to 20%)

- Use a blank line between groups of lines or between items as needed to enhance readability.
- Every class should include a .h file and a .cpp file.
- Each function or method should have a single well-defined purpose.
- Use identifiers in function prototypes (not just types) -- in general a function prototype will be identical to the first line(s) of the corresponding definition except for the extra semicolon in the prototype.
- As a rule of thumb, no function should be longer than one page of code. It should be easy to determine the logic of your program by reading the main method.
- Properly use reference/value arguments (page 37-38 in our text).
- Use const functions appropriately.
- Make data members in a class private and provide accessors & mutators for each data member that a client should be able to access.
- Use valued functions appropriately (only use when a single item needs to be created and returned and no I/O is needed).
- Use proper indentation (see page 42-44) and be consistent with your indentation.
- Blank lines should be used for readability.
- Use braces { } around conditional and loop statements when appropriate. Put each brace on a line by itself, and align it with the keyword of the structure (i.e, under the *i* of *if*, the *w* of *while*). (Exception: for a do-while loop, the while and Boolean condition go on the same line as the closing brace.)
- Give identifiers meaningful names. Identifiers include variable names, function names, and class names. Programmers are supposed to get a feeling of identifier meanings just from the names. Select names according the naming convention listed on page 41 of our text.
- Avoid global variables – variables defined outside of any braces. Global variables make it hard to maintain code. However, you can define global constants.
- Global variables should not be used in functions. In other words, variables in functions should either be declared locally or passed as parameters

- When input is accepted from the command line, a prompting print should clearly specify what is to be entered and if necessary how the item will be entered. For example if the user were expected to enter a date, they would need to know the form of the date to be entered (like 01/16/2009).
- Each line (including statements and comments) should be an appropriate length. If a line is too long, please break it up.
- Output must be neatly formatted and appropriately labeled.
- The client file (the file containing the main() function) should have the following basic outline:
- Heading (see documentation above)
- #include statements
- Function prototypes
- main() function
- Other functions should follow the main() function. Make sure to include appropriate headings for each function.
- Other items may be mentioned as the semester progresses.

Efficiency (up to 20%)

Care should be exercised to select efficient algorithms, which do not have excessive space and time requirements.