CSS343/502 - Coding Standards

The purpose of these standards is to promote program clarity, readability, and comprehension so as to produce programs that are easier to read, test, debug, modify, and maintain. C++ is used as the example language for this standard.
Any violation of these standards not approved, **in advance**, by the instructor will be subject to penalties at the discretion of the instructor. In general, exceptions will be approved only where justified by unique circumstances. Your text may not follow these standards, but we have chosen to adopt a more universal set of standards reflected in industry and academe.

General Goals

A program should be well structured and that structure should be evident both by the language of the code and by the physical appearance of the program.
Strive for effective simplicity and clarity rather than clever programming tricks which may not be understood by other programmers and which may cause difficulties later.
Use an object-oriented approach when dealing with complex problems, i.e, identify the objects and operations needed to resolve the problem. Then construct and test these objects and their operations.

Spelling and Capitalization

Because C++ is case sensitive, it is easier to program and remember variable names if some standard conventions are followed.
1. Reserved words are all in lower case as in int in the following: int classSize;
2. Declare constants and variables at the beginning of a file or a function definition.
3. Constants are all in uppercase. **PI = 3.1416;**

B. Variables, functions and file names must be descriptive of their use. You may use lower case, combination of lower and upper case or other style you prefer **but need to keep it consistently across all your files**. Following shows a style commonly used but you don't have to follow it if you don't like.
int rate, hourlyWage;
FindMax (void);
digitime.h
digitime.cpp

Punctuation And White Space

Do not use a preceding or a trailing underscore in identifier names - except for those that will not be used in actual code such as a name to be used in an #ifndef compiler directive. The reason for this standard is that it is difficult to distinguish _Edit or Edit_ from Edit.
Put each **{** and **}** on separate lines.
Each **{** indicating the beginning of a block and its corresponding end of block brace **} are**

recommended to be aligned and commented as to what block is started and ended.

Use space liberally, yet sensibly, to make your code readable.
Typically there are spaces surrounding operators such as =, +, etc.
Limit your code and comments to 80 characters per line.
Do NOT allow code or comments to wrap when printed on standard 8.5 by 11 inch paper in portrait mode.

Documentation And Style

Each FILE (class declarations, class definition, and class usage) should contain opening documentation as: A bordered block of comments should be placed at the beginning of each file to explain it, such as what that unit does, how it works, any special algorithms it uses, etc. The block of comments at the beginning of a program unit should clearly identify its author, date of initial creation and date it was last modified. This is illustrated below. The block should stand out visually to improve the readability of the program unit.

```
// ------------------------------------------------ file name ------------------------------------------------
// Programmer Name Course Section Number
// Creation Date
// Date of Last Modification
// -----------------------------------------------------------------------------------------------------------
// Purpose - a brief statement of the program's function
// -----------------------------------------------------------------------------------------------------------
// Notes on specifications, special algorithms, and assumptions.
// -----------------------------------------------------------------------------------------------------------
```

Functions:

Each function or method should perform a single well-defined operation. If your function or method accomplishes two tasks, break it in two functions. This improves the modularity of the code.
Comments should be placed at the beginning of each function definition to explain it, such as what that function does, how it works, any special algorithms it uses, etc. For example:

```
// ----------------------------------ReadyToQuit-------------------------------------------
// Description:
// ----------------------------------------------------------------------------------------
int ReadyToQuit (void)
{
      ...
} // end of ReadyToQuit
```

Indent the declarations and statements appropriately within each function.

Public vs. private

-- Methods should be made public only when necessary (that is, they are part of the interface with other modules).
-- Data members should almost never be made public.
-- In most cases, do not circumvent information hiding by using operations that return internal, implementation-dependent data (e.g., pointers or references to data members).

Global variables and constants

-- Do not use global variables anywhere in your code. While these are necessary in rare cases, you will not need to use them for any programs in this class.
-- Global constants are acceptable. Global constants are defined in the .h file of the class they go with.
-- Very few numbers appear in the code body. Use constant identifiers!

Consts

-- Use const member functions when possible.
-- When passing by reference use const when possible.

Comments

1. Comments should be used to explain key program segments and segments whose purpose or design is not obvious to a reader of your program. Remember that what is obvious to you the day you write a program may not be as obvious to you a week or two later. Furthermore, what is obvious to a programmer may not be clear at all to a reader or maintainer of the program. When in doubt, it is better to provide more documentation than required rather than less than is needed.
2. When the header file of a less commonly used class library is inserted in a program using #include, a comment should be used to explain its purpose. For example:
#include <iomanip.h> // provides output formatting functions
3. Block comments should be used throughout the body of a program to describe the purpose of logical sections of code. These should describe program flow in functional terms rather than detailing specific statements.
4. Meaningful identifier names should be used. All variables and constants should be explained preferably by using self-explanatory names.

wages = hoursWorked * hourlyRate;

is manifestly more self-explanatory than:

w=h*r;

Avoid cute identifiers, programs are used to accomplish tasks and not to make editorial statements. The following may be amusing, but it is inappropriate in a real world program.

baconBroughtHome = slaveLabor * lessThanImWorth;

All diagnostic and/or prompting messages issued (at execution time) by the program must be self-explanatory. For example, because there are so many date formats, the prompt:

Please enter the date.

does not help the user know what format to use for the date. A better prompt would be

Please enter the date in the form dd/mm/yy :

Label and format all program output so as to be easily understandable to the user.

5. It is the quality of the comments, **not the quantity**, that is important. Make every comment count. Make sure comments and code agree. Incorrect or misleading or useless comments, including misspelled words and incorrect usage, detract from program readability. Thus program documentation must be appropriately updated as the program code is modified.

6. Programs should be formatted to enhance readability. Among the things you should do are the following. Avoid the use of multiple statements on a line. Use at most one statement per line. Recommend to use one or more spaces between the items in a statement to make it more readable. For example:

total = (total + newAmount) + 0.5;

is clearer and easier to read than

total=(total+newAmount)+0.5;

Use white space to enhance readability. Insert a blank line between sections of a program and wherever appropriate in a sequence of statements to set off blocks of statements. Adhere rigorously to alignment and indentation guidelines to emphasize the relationship between the various parts of a program.

For example; if statements require several lines, indent the second and each subsequent line to show that they are part of the previous statement.

```
if (count <= 100)
{
    count++;
    total = a + b + c;
}
```

Recommend to use parentheses to indicate precedence of operations in all arithmetic and logical expressions with more than one operator.

courseGrade=quiz1*0.30+quiz2*0.30+quiz3*0.40/17;

is harder to understand, and not equivalent to

courseGrade = ((quiz1 * 0.30) + (quiz2 * 0.30) + (quiz3 * 0.40)) / 17;