Programming Rules

NOTE. There is a distinction between rules and guidelines. A rule is a *requirement*; it must be followed. A guideline is a *suggestion*; it is strongly encouraged but does not have to be followed. The following are rules; whenever an assignment is given, these rules are implicitly part of it and must be followed.

1. You must make sure that your program is free of all errors when it is compiled, linked, and executed on any of the department's "Lab G" computers, prior to submitting it. These are the machines in Lab 1000G, which are named `cslab1`, `cslab2`, and so on. All of these machines have identical architectures and software, so if it runs correctly on one, it will run correctly on any other[1]. In general, a program can run correctly on one machine but not another, for one reason or another. This requirement stipulates that it must run correctly on the lab machines.

2. You must submit all of the source code and nothing else, unless the assignment states otherwise. Do not submit an executable, data files, or output files.

3. Programs will not be accepted after their deadlines.

4. *The program must be your work, and your work alone.* You are not free to share solutions or parts thereof with anyone else unless this has been explicitly stated by the instructor. If you do not understand what it does or why it works because someone else's hand is in it, this will be discovered one way or another. You are forewarned that it is reasonable for you to explain to your instructor how your program works if asked to do so. If you cannot explain it, then it is not "yours". Attributing someone else's work as your own is *plagiarism*, and it is a violation of Hunter College policy. We will file an official complaint against any student who we believe has committed plagiarism.

5. *Every program must be professionally documented.* Every distinct source code file must contain a preamble with the file's title, author, brief purpose and description, date of creation, and a revision history. The description must be a few sentences long at the minimum. A revision history is a list of brief sentences describing revisions to the file, with the date and author (you) of the revision. This is an example of a suitable preamble:

```
/*****************************************************************************
Title :       draw_stars.c
Author :      Stewart Weiss
Created on :  April 2, 2010
Description : Draws stars of any size in a window, by dragging the mouse to
              define the bounding rectangle of the star
Purpose :     Demonstrates drawing with the backing-pixmap method, in which the
              application maintains a separate, hidden pixmap that gets drawn
              to the drawable only in the expose-event handler. Introduces the
              rubber-banding technique as well.
Usage :       draw_stars
              Press the left mouse button and drag to draw a 5-pointed star
Build with :  gcc -o drawing_demo_03 draw_stars.c \
                 `pkg-config --cflags --libs gtk+-2.0`
Modifications: April 29, 2010
               Improved efficiency of the algorithm a bit. See the code.
*****************************************************************************/
```

[1] Unless you are so clever that you have figured out how to make it behave differently depending on which host it is running, in which case you might have "shot yourself in the foot."

All function prototypes in your program, whether members of a class or not, must have a prologue containing a description of each parameter and the return value, if it has one, as well as appropriate pre- and post-conditions. These prologues *must not be in the implementation files of classes, but in the class interfaces.* All non-trivial algorithms must be documented in plain English in a multi-line comment block. All non-trivial declarations must have adjoining, brief comments. *Documentation is usually worth 10% of the grade.*

6. Every program must follow commonly accepted stylistic guidelines regarding the use of blank lines, white space, indentation, and naming of program entities such as variables, classes, functions, and constants. Your program must be consistent in its use of typographical format for distinguishing types, variables, functions, and constants. For example, you might decide that all type names should begin with an uppercase letter and all variables begin with lowercase letters, or that all variables use underscores to separate the words in the name, as in `number_of_scores` and `first_author`, or that they use changes in case, as in `numberOfScores` and `firstAuthor`. *Style is usually worth 10% of the grade.*

7. Programs should avoid error-prone syntax as much as possible. For example, it is better to write the condition

   ```
   if ( 0 == N )
   ```

   than the condition

   ```
   if ( N == 0 )
   ```

   because of the very common mistake of writing `( N = 0 )` instead. Similarly, one should always use braces with compound statements such as if's and while's, even if they are not necessary, as the following example demonstrates:

   ```
   count = N;
   while ( 0 < count ) {
       a[count--]++;
   }
   ```

   because if you make a habit of doing this, you will not be in the situation where you inadvertently write this:

   ```
   count = N;
   while ( 0 < count )
       cout << a[count] << endl;
       count--;
   ```

   and waste time trying to figure out why your program is stuck in an infinite loop.

8. *Every program must be correct to receive full credit.* "Correct" means that for every possible input, it produces output that is consistent with the specification. If the program produces correct results for some, but not all, inputs, it is not correct. Since there may be infinitely many possible inputs, you cannot possibly establish your program's correctness by running it on all inputs. You must use a combination of sampling (i.e., testing) and logical analysis to convince yourself of its correctness. *Correctness is usually 50% to 70% of the grade.* A very common mistake I have found is for students to hand in programs that do not even run correctly on the input file I gave out. In other words, students have failed to check the outputs of their programs before submitting them. This is either laziness or hubris.

9. The output of any program should be readable and understandable by ordinary human beings who lack mind-reading capability and who have not read the assignment or the program. For example,

```
The file "playlist1" contains 6 songs that won Grammies in 2010.
```

is more understandable than the output

```
6 songs
```

or this

```
playlist1: 6
```

10. Every program must satisfy specified performance requirements if these are stated. This means that it uses an amount of storage and running time within specified or reasonable limits. *Performance may be worth between 10% and 30% of the grade, depending on the assignment.*

11. There are no rigid rules. All rules can be broken, but it is best to check with the instructor before taking a chance.