Programming Rules

NOTE. There is a distinction between a rule and a guideline. 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 a programming assignment is given, these rules are implicitly part of it and must be followed. Failure to follow them will result in a loss of points on the assignment, the exact number of which is dependent on the assignment.

1. You must make sure that your program is free of all errors when it is executed on any of the department's `cslabXX` computers, prior to submitting it. These are the machines in the walk-in lab, 1001B, which are named `cslab1`, `cslab2`, and so on. All of these machines have identical architectures and software, so if a program 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 these lab machines specifically.

2. *Every program must be correct to receive full credit.* "Correct" means that for every possible input allowed by the assignment, it produces output that is consistent with the assignment specification. If the program produces correct results for some, but not all, inputs, it is not correct. Since there may be an unbounded number of 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 worth anywhere from 50% to 70% of the grade.* A very common mistake is for a student to hand in a program that does not even run correctly on the input files suggested by the instructor. In other words, the student failed to check the outputs of the program before submitting it. This is either laziness or hubris. It is also very common to make a "small" last-minute change to a program and fail to re-test the program on all inputs (because after all it was such a tiny change), only to learn later that the change "broke" the program completely. Test the exact version that you submit!

3. You must submit all of the source code and *absolutely nothing else*, unless the assignment states otherwise. Do not submit any data files or output files unless told to do so. This will result in a loss of points.

4. For full credit, an assignment must be submitted in the manner described in the assignment by the deadline. Whether or not it is accepted after the deadline, and if so, how much it will be penalized for lateness, is stated in the syllabus is the determining rule.

5. *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 your instructor might ask you to explain how your program works and that you should be able to do so without advance preparation. If you cannot explain it, then it is not "yours". Representing 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.

6. *Every program must be professionally documented*:

   (a) Every distinct source code file, whether a `bash` script or a Perl program, 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

[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."

sentences describing revisions to the file, with the date and author (you) of the revision. This is an example of a suitable preamble:

```
#!/usr/bin/env  perl
# nested_if.pl
# Usage                 : nested_if.pl
# Written by            : Stewart Weiss
# Created on            : October 18, 2010
# Description           : Demonstrates how to "nest" if-statements
#*****************************************************************************
```

(b) 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, but the specific project's rubric might state otherwise.*

7. Every Perl program must turn on all warnings and use the strict pragma:

```
$^W = 1;     # turn on warnings
use strict;  # behave!
```

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

```
if ( 0 == $number )
```

than the condition

```
if ( $number == 0 )
```

because of the very common mistake of writing "`if ( $number = 0 )`" instead. Similarly, one should always do this with loop conditions:

```
until ( 0 == $count ) {
    $count = $count - 1;
}
```

9. 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, functions, and constants. Your program must be consistent in its use of typographical format for distinguishing variables, functions, and constants. For example, you might decide that all variable names 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`, which is called *camelCase*. *Style is usually worth 10% of the grade.*

10. 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, unless specified otherwise. For example, the output

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

is more understandable than the output

```
6 songs
```

or this

```
playlist1: 6
```

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