16.070: Introduction to Computers and Programming

Style Guide

Above all, remember: clever code is unreadable code.
Save your cleverness for algorithms and document them well.

Key:
- ➢ **YOU MUST** follow this rule
- • We recommend you follow this rule

Syntax Specific Guidelines

IF

- ➢ IFs with ELSE IF clauses should always include an ELSE block even if it's only an error message.
- ➢ IFs with AND conditions should only use fully elaborated Boolean expressions, and not rely on ordering for implicit conditions.
- ➢ IFs with ELSE IF clauses should form an exhaustive set of alternatives
- • IFs that serve only to guard a block of statements (i.e. test only one condition) and have no alternatives (ELSE IF blocks) do not necessarily require an ELSE block
- • Nested IFs contain implicit AND conditions, making them hard to maintain. Be very careful when you use them.
- • ELSE statements should be followed by a comment that explains the conditional to which it is attached.
- • Combine multiple IF/ELSE IF alternatives that execute the same code. Duplicated code requires more effort to maintain.
- • It is preferred that all if statements, even those with only one line, use braces.

BOOLEAN EXPRESSIONS

- ➢ Short-circuiting should not be used, because the order in which multiple expressions in a single **if** are evaluated is not always defined. Use nested Ifs to avoid runtime errors. Comment these cases to warn other programmers that a potential problem exists.
- ➢ Conditional expressions should not have side effects.
- • Use parenthesis wherever possible. Avoid relying on evaluation order.

ARRAYS

- ➢ Use constants to define your array bounds. It makes writing loop code more scalable.

FOR

- ➢ When traversing arrays, use attributes of the array definition to define the bounds of iteration.
- • Don't use for loops for tasks other than iterating. For loops are designed for iteration and imply the presence of iteration. If not iterating, use a while loop.

WHILE

- Use flags and/or sentinels to exit loops when necessary. Do not use multiple returns. A return should come at the end of a function. Do not use **break** or **continue**.
- Try to express the conditional as a positive rather than simply using a logical NOT

SWITCH / CASE

- Include a default condition even if you think it will never be reached during normal operation.

General Guidelines

NAMING

- Try to avoid generic names like X, Y, I, etc. Exceptions to this rule include FOR loop variables, which often use I, J, K, etc. to clearly identify nesting.
- Name constants using all capitals and variables using mostly lowercase.
- Global variables should be named in some way to indicate their special scope, e.g. having a 'g' as the first character of the variable's name.
- Pointer variable names should begin with *p* or with *p_* .
- Try to avoid uncommon abbreviations. Elaborate abbreviations fully in comments at declaration.
- Descriptive but concise variable names help more to make code readable than any other facet of the programming, including comments.
- Function names should be as descriptive of their purpose as possible.

FUNCTION DECLARATIONS

- Banner comments should include:
  - The function name and a description of its function. Be sure to include any caveats or limitations.
  - A list of its dependencies
  - All inputs and outputs
  - very long argument list should be split over several lines
  - use ANSI style argument declarations, not K & R
- Clearly demarcate where each function begins and ends. A short comment after the ending '}' is useful.

WHITESPACE

- Make opening braces the last character on their line or else on a line alone.
- Put the closing braces of functions on their own lines.
- Place a space between comma separated elements.
- Insert a blank line between functionally different blocks of code, and before any control flow construct.
- Don’t separate comments that describe a piece of code from the code by a blank line.
- Very long lines of code should be broken over several lines

COMMENTS

- When programming in C (as you will be in 16.070), **always** use c-style comments.

- Banner comments at the top of the file are important. They should include:
  - The author's name and contact information (i.e. e-mail address)
  - History of recent modification with dates
  - A list of any other modules and/or external data that the file depends upon
  - (Not required but often helpful) A list of any modules that depend on the file
  - A description of the file's functionality.

- Comment closing braces with a reference to their opening braces when they are not within a few lines of each other.

- Comment control flow structures. Especially explain exit condition for sentinel loops

- More comments are better. Your comments should explain the general flow of your code, however you need not explain every line, especially when the function of the line is obvious.

- When possible, comment expected ranges/states next to variable declarations.

INDENTATION

- Always indent at the beginning of each new block or control structure. This applies to one-line blocks as well.

- Ensure that your indentations always line up.

- A minimum of two spaces and maximum of five should be used for your indent.

GENERAL DON'TS

- Don't use goto.

- Don't use ++ or -- unless it's on a line by itself or in a `for` statement.

- Don't use the comma operator (if you know don't know what that is the don't worry)

- Don't use the ternary conditional operator (<boolexp>?<exp>:<exp>)

- Use pointers only when necessary.

ADDITIONAL STYLE GUIDES AND RESOURCES

- The Indian Hill Style Guide (a quasi-standard):
  - http://www.cs.umd.edu/users/cml/cstyle/indhill-cstyle.html

- The Ten Commandments for C Programmers
  - http://www.cs.umd.edu/users/cml/cstyle/ten-commandments.html

- C. M. Lott's index of C and C++ Style Guides
  - http://www.cs.umd.edu/users/cml/cstyle/

- Your problem set solutions provide many examples of well written code.