Bucknell University
Electrical and Computer Engineering Department
Series on Technical Communication

Program Source Code

2019-11-07

Introduction

You might not think of computer source code as a form of communication, but it almost always is used in that way. As a student you want to communicate to your instructor that you have successfully completed your programming assignment. When working as part of a team, your teammates need to be able to understand clearly how your work interfaces with other parts of the overall design. Perhaps most importantly, you need to communicate with your future self--in a few days, weeks, or months you may want to modify or reuse your old code, and it's surprising how quickly you forget why you used a particular constant or conditional test.

Many of the suggestions provided here are matters of style rather than function, and your instructor or employer may have distinctly different ideas about what constitutes good style. When such conflicts arise your first priority should be to use a style that is consistent with the expectations of your team. In the absence of formal requirements you can define your own style, but be sure to use it consistently throughout your work.

Comments

In Python, the major comment blocks are defined as **docstrings**. The general format and usage of docstrings is defined by the Python Software Foundation [PEP8][PEP257].

The comment block at the top of each file should indicate the file name (helpful if it is printed), authors, and a general description of the function or purpose of the code [JSF, rule 133]. This block should include the license boilerplate that describes your rights as the author of the code [GC++][GPython]. In the real world you would also document revisions here.

Comments for variables and constants may be in a common comment block that precedes all of the declarations, or each identifier may be defined with a single line comment that is just above the declaration or an in-line comment after the declaration.

Never use trivial comments [JSF, rule 131] such as

```
      void foobar(); // foobar has no parameters and returns nothing
      i++;           // increment i
      foo = 17;      // initialize foo to 17
```

or

```
      spam = 0                    # Set spam to 0
      eggs_and_spam.append(beans) # append beans to eggs_and_spam
```

Assume that the reader understands the syntax of your programming language. Your comments should describe your **intent**...what are you trying to do?

In Python, the docstring for a function, class, or method is a string literal that must be the very first statement in the object being described [PEP257]. In other words, the docstring comes **after** the line that declares the function, class, or method. In C and C++, comments should be placed **before** the block of code that is being described [Barr, rule 2.2.b][GC++]. Short comments within the code should be above or on the same line as the documented statement(s) [GPython]

Comments should never be nested, particularly C block comments (i.e. /* ... */) or Python's triple double quotes (i.e. """ … """).

For executable code, a separate comment block can properly document no more than about half a page of code. If in-line comments are used then you should expect to add a comment for every 3 to 6 lines of code.

Never "comment out" executable statements [JSF, rule 127][Barr, rule 2.1.c].

Comments used when declaring variables and constants should do more than restate the obvious. Describe the **meaning** of the data. For Boolean variables, describe what it means when the variable is true.

- Worst:
```
const int barfoo = 127;
int foobar;
```
- Bad:
```
// barfoo is 127
const int32_t barfoo = 127;
int foobar; // foobar is stored here
```
- Better:
```
// Access-door tamper switch address
const *int DOOR_SWITCH = 0x1234;
int InputData; // Audio input, right channel
```
- Best:
```
// Access-door tamper switch address, 0=closed, 1=open
const *int32_t DOOR_SWITCH_ADDR = 0x1234;
// 16-bit right chan audio in, signed, left justified
int16_t RtAudioInput;
```

Identifiers

Naming rules for identifiers are often specified so that the characteristics of the object bound to the identifier can be inferred from the name of the identifier itself. For example, some organizations will require that an identifier for a new datatype ends with "_t", or that the names of constants always begin with "k" [GC++]. Because such rules are not universally standardized and are often just a matter of style you may use whatever naming scheme you like in this course. However, you should always strive for clarity and consistency in your coursework.

Names of variables and constants should be noun phrases that indicate the meaning of the data (`AudioIsMuted`, `audio_is_muted`, `bottlesOnTheWall`, `MEANING_OF_LIFE`). Boolean variable names should suggest the question that they answer, or their meaning when true.

Names of functions should be verb phrases that indicate the function performed (`InitUART`, `ejectPassenger`) or the meaning of a return value (`button_is_pressed`, `BufferIsEmpty`).

Formatting

Indentation should be consistent. In Python code, use four spaces (**not** a tab) for each level of indentation [PEP8][GPython]. For C, the amount of indentation for each level should be at least two [GC++][JSF] but not more than four [Barr] spaces. Spaces are typically preferred over tabs.

In general you should not let lines of C code exceed 80 characters [Barr, 1.2.a][GC++], Python should be no more than 79 characters per line [PEP8]. The references provide the rationale for this rule and also list acceptable exceptions. The references also provide guidance on how to split a line that is, by necessity, too long.

In C, blocks of code, even trivial or empty blocks, must be surrounded by curly braces ('{' and '}') after statements such as `if`, `else`, `while`, or `switch`.

Delete unnecessary variable declarations, function prototypes, symbol definitions and similar cruft. Delete trailing whitespace at the end all lines[PEP8]

In C you can use a single blank line to separate blocks of code [Barr], but you shouldn't have multiple blank lines [GC++]. Python allows two blank lines before top-level definitions but not more than a single blank line elsewhere [GPython][PEP8].

Statements and Expressions

Don't rely on implicit typecasting. Change the datatype of a value explicitly if necessary.

Provide a single entry point and a single exit point for functions and conditional blocks of statements. In C, avoid statements such as `goto` or `continue`, and use `break` only with `case`.

References

[PEP8] PEP 8 -- Style Guide for Python Code
[PEP257] PEP 257 -- Python Docstring Conventions
[GPython] Google Python Style Guide
[GC++] Google C++ Style Guide
[Barr] Embedded C Coding Standard, The Barr Group
[JSF] Joint Strike Fighter C++ Coding Standards, 2005, Lockheed Martin Corporation

About this Guide

This guide is part of a series that has been established to provide a repository of information on technical communication for the students and faculty of the Bucknell University Electrical & Computer Engineering Department. Its primary goal is to foster consistent standards applied to the preparation of reports, presentations, and other forms of communication within the ECE curriculum. In effect, the guides in this series constitute official department policy on technical communication.

Although it is important to adhere to standards for graded class work, you should strive to maintain high standards as well in ungraded work and in day-to-day communications with your professors, other students, and professional contacts. You should view every instance of communication as an opportunity to practice your skills so that eventually they become second nature. Fair or not, your colleagues will form opinions of your professional competence based partly on how well you express yourself.