Princeton University
COS 217: Introduction to Programming Systems
Rules of Programming Style

These rules of programming style are from *The Practice of Programming* (Kernighan and Pike). Italicized rules pertain to the *process* of programming; non-italicized rules pertain to the *products* of programming, that is, computer programs. The parenthesized number at the end of each rule indicates the page that describes the rule.

Style: Names

1. Use descriptive names for globals, short names for locals (3).
2. Be consistent (4).
3. Use active names for functions (4).
4. Be accurate (4).

Style: Expressions and Statements

5. Indent to show structure (6).
6. Use the natural form for expressions (6).
7. Parenthesize to resolve ambiguity (6).
8. Break up complex expressions (7).
9. Be clear (7).
10. Be careful with side effects (8).

Style: Consistency and Idioms

11. Use a consistent indentation and brace style (10).
12. Use idioms for consistency (11).
13. Use else-ifs for multi-way decisions (14).

Style: Function Macros

14. Avoid function macros (17).
15. Parenthesize the macro body and arguments (18).

Style: Magic Numbers

16. Give names to magic numbers (19).
17. Define numbers as constants, not macros (20).
18. Use character constants, not integers (21).
19. Use the language to calculate the size of an object (22).

Style: Comments

20. Don't belabor the obvious (23).
21. Comment functions and global data (24).
22. Don't comment bad code, rewrite it (25).
23. Don't contradict the code (25).
24. Clarify, don't confuse (26).

Interfaces

25. Hide implementation details (104).
26. Choose a small orthogonal set of primitives (105).
27. Don't reach behind the user's back (105).
28. Do the same thing the same way everywhere (105).
29. Free a resource in the same layer that allocated it (107).
30. Detect errors at a low level, handle them at a high level (111).
31. Use exceptions only for exceptional situations (112).

Debugging

32. Look for familiar patterns (120).
33. Examine the most recent change (120).
34. Don't make the same mistake twice (121).
35. Debug it now, not later (121).
36. Get a stack trace (122).
37. Read before typing (122).
38. Explain your code to someone else (123).
39. Make the bug reproducible (123).
40. Divide and conquer (124).
41. Study the numerology of failures (124).
42. Display output to localize your search (124).
43. Write self-checking code (125).
44. Write a log file (125).
45. Draw a picture (126).
46. Use tools (127).
47. Keep records (127).

Testing

48. Test code at its boundaries (140).
49. Test pre- and post-conditions (141).
50. Use assertions (142).
51. Check error returns (143).
52. Program defensively (142).
53. Test incrementally (145).
54. Test simple parts first (145).
55. Know what output to expect (146).
56. Verify conservation properties (147).
57. Compare independent implementations (148).
58. Measure test coverage (148).
59. Automate regression testing (149).
60. Create self-contained tests (150).

Performance

Portability