# The Elements of Programming Style

The following rules of programming style are excerpted from the book "The Elements of Programming Style" by Kernighan and Plauger, published by McGraw Hill. Here is quote from the book: "To paraphrase an observation in *The Elements of Style* by Strunk and White, the rules of programming style, like those of English, are sometimes broken, even by the best writers. When a rule is broken, however, you will usually find in the program some compensating merit, attained at the cost of the violation. Unless you are certain of doing as well, you will probably do best to follow the rules."

1. *Write clearly– don't be too clever.*
2. *Say what you mean, simply and directly.*
3. *Use library functions whenever feasible.*
4. *Avoid too many temporary variables.*
5. *Write clearly – don't sacrifice clarity for "efficiency."*
6. *Let the machine do the dirty work.*
7. *Replace repetitive expressions by calls to common functions.*
8. *Parenthesize to avoid ambiguity.*
9. *Choose variable names that won't be confused.*
10. *Avoid unnecessary branches.*
11. *If a logical expression is hard to understand, try transforming it.*
12. *Choose a data representation that makes the program simple.*
13. *Write first in easy-to-understand pseudo language; then translate into whatever language you have to use.*
14. *Modularize. Use procedures and functions.*
15. *Avoid* `goto`*s completely if you can keep the program readable.*

16. *Don't patch bad code – rewrite it.*
17. *Write and test a big program in small pieces.*
18. *Use recursive procedures for recursively-defined data structures.*

19. *Test input for plausibility and validity.*
20. *Make sure input doesn't violate the limits of the program.*
21. *Terminate input by end-of-file marker, not by count.*
22. *Identify bad input; recover if possible.*
23. *Make input easy to prepare and output self-explanatory.*
24. *Use uniform input formats.*
25. *Make input easy to proofread.*
26. *Use self-identifying input. Allow defaults. Echo both on output.*

27. *Make sure all variable are initialized before use.*
28. *Don't stop at one bug.*
29. *Use debugging compilers.*
30. *watch out for off-by-one errors.*
31. *Take care to branch the right way on equality.*
32. *Be careful if a loop exits to the same place from the middle and the bottom.*
33. *Make sure your code does "nothing" gracefully.*
34. *Test programs at their boundary values.*
35. *Check some answers by hand.*

36. *10.0 times 0.1 is hardly ever 1.0.*
37. *7/8 is zero while 7.0/8.0 is not zero.*
38. *Don't compare floating point numbers solely for equality.*

39. *Make it right before you make it faster.*
40. *Make it fail-safe before you make it faster.*
41. *Make it clear before you make it faster.*
42. *Don't sacrifice clarity for small gains in "efficiency."*
43. *Let your compiler do the simple optimizations.*
44. *Don't strain to re-use code; reorganize instead.*
45. *Make sure special cases are truly special.*
46. *Keep it simple to make it faster.*
47. *Don't diddle code to make it faster — find a better algorithm.*
48. *Instrument your programs. Measure before making "efficiency" changes.*

49. *Make sure comments and code agree.*
50. *Don't just echo the code with comments — make every comment count.*
51. *Don't comment bad code — rewrite it.*
52. *Use variable names that mean something.*
53. *Use statement labels that mean something.*
54. *Format a program to help the reader understand it.*
55. *Document your data layouts.*
56. *Don't over-comment.*