CODING STYLE

COMMENTS
Good places to put comments are:
- a broad overview at the beginning of a module
- data structure definitions
- global variable definition
- at the beginning of a function
- tricky steps within a function
- before each block of statements
- anything out of the ordinary
The opening / of all comments should be indented to the same level as the code to which it applies
If you put a comment on the same line as code, set it off from the code with a few tabs.
Don't continue such a comment across multiple lines.
The size of the comment should be proportional to the size of the code that it refers to.
Comments should describe what the code is doing, not how it’s being done.

TYPES AND DECLARATIONS
Explicitly define types as often as possible.
Think about the order of the fields.
Try to keep related fields grouped.
Within groups of related fields, pick some uniform scheme for organizing them, for example alphabetically
or by frequency of use.
When all other considerations are equal, place larger fields first, as C's alignment rules may then permit the
compiler to save space by not introducing "holes" in the structure layout.

NAMING CONVENTIONS
#define designations should be in ALL CAPS
Names should be meaningful
Names should be chosen to make sense when your program is read
Variables should be noun clauses.
Use some consistent scheme for naming related variables.
Boolean variables should be named for the meaning of their "true" value.
Procedures (functions called for their side-effects) should be named for what they do, not how they do it.
Function names should reflect what they return.

INDENTATION AND LAYOUT
Each {} statement block set of statements should be indented
Avoid unnecessary curly braces
Use spaces around keywords.
Use spaces around binary operators, except . and ->, for they are morally equivalent to array subscripts,
and the "punctuation" operator ','.
Don't use spaces around unary operators, except sizeof and casts.
Don't parenthesize things unnecessarily.
If an expression gets too long to fit in a line, break it next to a binary operator. Put the operator at the
beginning of the next line to emphasize that it is continued from the previous line. Don't add additional
indenting to the continued line.
**EXPRESSIONS AND STATEMENTS**
Feel free to use a FOR loop where some of the parts are empty.
If a FOR statement gets too long to fit in a line, turn it into a while.
If your loop control is that complicated, it probably isn't what FOR is for (pun intended).
Always put braces following IF statements. If you do this, you never have to remember to put them in when you want more than one statement to be executed, and you make the body of the if statement more visually clear.

**IN GENERAL**
Create your code so that it is modular; in understandable blocks of code.
Functions should be short and sweet.
Include occasional sanity checks in your code... “and you thought this couldn’t happen” kind of scenarios.
Use goto sparingly. Two harmless places to use it are to break out of a multilevel loop, or to jump to common function exit code. (Often these are the same places.)
Avoid sloppiness. Decide what your style is and follow it precisely.
The standard library is your friend. There's no excuse for writing code which already exists there.
Fancy algorithms are buggier than simple ones, and they're much harder to implement. Use simple algorithms as well as simple data structures.
Data dominates. If you've chosen the right data structures and organized things well, the algorithms will almost always be self-evident. Data structures, not algorithms, are central to programming.
All those random extra spaces make me wonder if the programmer was even paying attention!