Coding Style Guideline

KEYS:

- Maintainability
- Readability
- Modularity
- Reusability
- No unnecessary global
- Clear function definition
- For your level, no function should last for more than 2 pages long
- Consistent style

Variables:

- Long names – use camel style or embed the words with "_".
- Never use a single letter other than just a simple loop count for a loop with just a few lines.
- Even loop count which carry any rather significant meaning to the whole function / program, you should use a meaningful name.
- Variable names should be descriptive of the value stored in them. Local variables whose purpose is self-evident
- Global should never be just one character. Always start a global with an uppercase.

Good Documentation

- Use ***enum*** or names who serve the purpose of "self-documentation". Your variable names and function calls should generally make it clear what you are doing.
- Comments should not describe what the code self-evidently does. For example: i=i+1; // increase i by one. This comment offers no information at all.
- File header: Each file should contain a comment describing the purpose of the file and how it fits in to the larger project.
- Function header: Each function should be prefaced with a comment describing the purpose of the function (in a sentence or two), the function's arguments and return value, any error cases that are relevant to the caller, any pertinent side effects, and any assumptions that the function makes.
- Use macro or global const to represent error code. Do NOT hardcode numeric error like -1, -2, 0, etc. inside the function.
- Explain error code; such as return -1. What dose "-1" mean? Do this along side where they were declared, such as inside your header file.
- Embed blocks of code with comment (especially when particularly long). Block usually entails the meaning of a single sub-component / movements / logic.
- Should ALWAYS comment Tricky bits of code (such as pointer arithmetic, or complex Boolean expression, etc.)

Good Use of Whitespace

- Conscientious in indentation, spaces; AND be consistent.
- Stay with 80 characters / line.

Good Variable Names

Variable names should be descriptive of the value stored in them, such as isPrime, anagramCt. A single letter must be limited to local variables whose purpose is self-evident (e.g. loop counters or array indices), and self-contained inside that loop inside. Parameters can be one (well-chosen) word. NEVER use a single letter for global variables. Imagine what it is like to try to find all instances.

Magic Numbers

Magic numbers are numbers in your code that have more meaning than simply their own values. For example, if you are reading data into a buffer by doing "fgets(stdin, buf, 256)", 256 is a "magic number" because it represents the length of your buffer.

You should use #define to clarify the meaning of magic numbers. In the above example, doing one of the followings:

- sizeof(buf);
- "#define BUFLEN 256" and then using the "BUFLEN" constant in both the declaration of "buf" and the call to "fgets".

No "Dead Code"

"Dead code" is code that is not run when your program runs, either under normal or exceptional circumstances. Your submission should have no "dead code" in it. How about those "printf" statements you used for debugging purposes but since commented. The alternative is to use ***conditional compilation***.

Modularity of Code

You should strive to make your code modular. Examples:

a) If blocks of code are repeatedly being used, they can be extracted out into a function.
b) Long functions that perform several tasks should be split into sub-functions when practical.
c) Code that performs different functions should be separated into different modules.

Failure Conditions/Error Checking

When writing a program, we usually only consider the success case. It is equally, if not more, important to consider the failure cases. A typical one is the error from accessing a file, or from malloc.

Proper Memory and File Handling

If you allocate memory (malloc, calloc), you should free it after use. Your program should not have memory leaks. If you use open a file, you should close it after use. Closing a file is very important, especially with output files. The reason is that output is often buffered.

Consistency

Example - where the curly braces go for control blocks, such as "if" , while, etc.

Trouble Shooting Tips for robotics projects

- Coming Soon!