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

Naming conventions for 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

- 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.
- 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.
- 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. nothing but just 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 arbitrary than what it is meant to represent.

For example,

Bad	Better
fgets(stdin, buf, 256)	fgets(stdin, buf, sizeof(buf)-1) Or #define siz 256 char buf[siz]; cgets(stdin, buf, siz);
For (i=0; i<10;i++) Arr[i] = i*10;	Int nElements = sizeof(arr)/sizeof(arr[0]); for (i=0; i<nElements;i++) Arr[i] = i*10;

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:

- Always boundary check for array ;
- Always check the status 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

- Use sound
- Use display
- Use pause

Preliminary Programming Style Checklist

Before your program can be considered complete, you must ensure that the following requirements are met. Please ask the instructor if there is anything you are unsure about.

1. Code is properly commented – Meaningful comments only. Do not state the obvious. Examples:
    i. meaningful : //follow the line until you see silver.
    ii. Useless: //start a program, or // sensor value higher 50, or //stop motors.
2) No "X's" remaining in the compiler errors window – pay attention to proper syntax, and fix all warnings. Even if the warnings are for unreferenced functions or variables, try your best to minimize them.
3) Code is neat and properly formatted, using appropriate curly braces and indentations – Each curly brace should have its own line. Take advantage of the "fix formatting" button on the toolbar!
4) Header files should only contain variable and sensor initializations. It is a good idea to create a header file if you do not already have one, especially if you are working as a team.
5) All variables should either be initialized in a header file or at the beginning of each function (Note: even task main()). Do not initialize variables randomly throughout your code.
6) Variables should have meaningful names that reflect their functionalities. Try to limit a single letter only to represent something like a local loop count within "a small scope". For large scope, even the loop count variable should be more intuitive.
7) Always create a variable or macro for a repetitiously used constant.
8) Minimize global variables.
9) Prefix the global with 1st letter to be in upper case. If name is long, use camel style, e.g. EncoderPerDegee.
10) Must use a proper variable name for your sensors and motor ports. Do not directly reference it as S1, motorA, etc., inside functions.