2) Beginnings

These notes are about why I do things the way I do them and why I use certain tools. Reading code teaches me how other folks solve a problem, but the real learning comes when I write and edit my own code. I use a plain editor, instead of a full blown integrated development environment (IDE), because of impatience. When I imagine some code I want to write it down, compile it, and see how it works. I can have my simple editor open, ready to write/edit/run code, with only a few clicks . If I call up Microsoft Visual Studio I have to wait for over a minute before I can even start to write code. The IDE from Atmel is even slower, taking almost three minutes. So I have learned to use smaller, lighter tools. I am more productive because I no longer fight with the tools. They do EXACTLY what I want them to do and little more. I can write my ideas quickly, then set up makefiles and project folders afterward. I have templates to start most projects, all I need to do is edit them a little. I change the makefile to reflect the current project and that's about it; then I am ready to add my new code. Learning from tested procedures sets you up to succeed.

Normally, I wake up with ideas and want to write them down. If my code editor won't wake up fast enough I'll use a pad and pencil. Mostly I want to capture the pseudo code in my head onto paper or onto the screen. Once I flesh out my ideas I put comment markers // in front of each line. I use the pseudo code as comments while building the app. I include a lot of notes in the comment lines to keep track of changes I am thinking about or have made.

Over time I have taken to using lighter tools and creating more templates to use. The templates allow me to create working code much faster. I tend to work top-down so I can compile and run the code from almost the very start. Any function which I haven't written yet exists as a stub which does nothing and returns nothing; it is just a place holder. As I fill in those stubs I get more functionality with each compile. I also keep an extensive library of most of the code I have written. I copy chunks from files in that library for any given project. Over the years these functions have become more generalized and easier to implement.

Using a makefile may seem old school, but it gives me better control over the compile and link processes. It requires very few computer resources. You can direct where each portion of code or data will go while using them for microcontroller programming. One portion will burn to flash while another goes into EEPROM; one area is for the program, the other is for any data it may require. You can also write to the EEPROM from your application. Using a makefile gives me knowledge about the processor, the memory structure, and the libraries necessary to link everything together.

I used notepad++ for years until I finally got frustrated with the way it auto-formatted things. Even turning that off was not enough. So I moved to the PSPad editor because I can use Whitesmiths' formatting with a press of a button. I change it to whatever other format is desired with another button press. Eclipse allows this too but did not in the past.

You should get a decent code editor. They provide you with highlighting and formatting. I am using PSPad now because it allows the formatting style I like. Eclipse is another editor, but it is enclosed in an IDE which takes up more space. PSPad and Eclipse are free. They work for all the languages I use and many, many others. In the long run this saves a lot of keystrokes, plus it has a great search and replace engine. Reading code from your editor of choice is the most common way to examine code. Seeing it improperly formatted just gives me shivers!

I thought my formatting style was all my own, developed over years of writing a variety of languages, on many computers. What really fixed it into my head and fingers was Pascal. The brackets and braces just felt right. The indentation style felt right too. White space is cheap and really helps in comprehending files as you read them. But, what I thought was my own had been claimed. Whitesmiths style came up when I did a search. I also is found it in the Eclipse IDE. With a few keystrokes Eclipse will transform terribly formatted code into perfectly formatted code. It is just like magic :)

Read here: https://en.wikipedia.org/wiki/Indentation_style#Whitesmiths_style for examples. Indenting the body of looping structures, keeping the braces around it, helps me see modules and how they structure the program's flow. One thing I learned from Whitesmiths was its switch command formatting. Since I don't use the switch command very often a formatting preference had not crystallized for me.

Many editors won't let you format code properly, or make it so much of a pain to change that you have to modify your methods to conform. Once again, I think this is a matter of freedom. If I have to fight with an editor over how to format a file I won't use it very long.

Since I write code under both Windows and Linux operating systems, I need two different editors. On Windows I use PSPad, as I have mentioned. Under Linux, I am still trying a number of them to see which is the best fit. Eclipse works well once you get through all of the IDE hoops. gedit was working fine, then it wasn't, then it was. Flaky software I don't like. I do use nano if I don't care about formatting. Sublime Text2 is good, I've been testing it. Currently I am testing Geany. It doesn't seem to mind my formatting but I've only been using it for editing, not for writing new code. So it needs more testing.

My indents are standardized to 3 spaces. I have seen examples using 2, 4, or 5 spaces. I find 2 spaces to be too few to notice in a multiple loop structure. I find 4 and 5 to be wasteful of space under the same context. 3 spaces feels just right. Makefiles are notorious for needing a true tab rather than using spaces. Normally I use three spaces which allows me to work from a generic text editor or a full blown IDE on the same file. Using a tab can mean your indentations become 8 spaces deep on some editors or 4 on others or 5 on yet others. By translating a tab into 3 spaces or simply typing 3 spaces you get uniformity across editors.

Poorly formatted code is hard to read and even more difficult to debug. Properly formatted code, with plenty of whitespace and many comments, is your best way to be able to understand the code you are writing right now, in six months, or in six years. Formatting and comments are your link to sanity.

I like PSPad as an editor. It also has the lightest project apparatus I've found yet. I have decided to include PSPad project files (*.ppr) for every lesson. If you have installed PSPad on your Windows box you can double click on the *.ppr file and PSPad will open with all of the files of the application open and ready to edit. Easy navigation between each file in the editor makes life easier. Copy and paste from one to another is simple. Creating a new project is very simple too. Either open all of the necessary files in the editor and hit new project from files, or you can create a new project and fill it with new files one by one. It does not demand you do things in any given way. Freedom. It is also very lightweight, fast, and free.

You will be using a terminal window (Command Prompt) a lot. To create a new one on a Linux box just type ctrl-alt-T. On a Windows box go to Start | All Programs | Accessories | Command Prompt. On my Ubuntu box I have it available near the top of the Task Bar. On my Windows box I dragged it from my Start menu onto the toolbar close to the lower left. I use it so often I make sure it is easily available with a mouse click. On my Ubuntu box I have three terminal windows open automatically on start up. They are that important to me. I like the control and the freedom. (Noticing a theme yet? :)

By learning, and using, generic tools I can test my code against a number of compilers. I can edit the code anywhere I go because I am not tied to any IDE. I can take a flash drive with my work and edit any file anywhere I go. In fact, I can take an entire Ubuntu OS on my flash drive with my work and have the environment ready to go from any computer I can boot. But normally I just need an editor and a place to sit; or even just a pad of paper and a pencil. Ideas are in my head ready to coalesce on paper or on a screen.

The application on the computer screen is a map of the problem space, my computer hardware, and my ability to craft software. I see computers as tools, like saws or hammers, with which I can build what I imagine.

www.pspad.com

https://www.pspad.com/en/download.htm

https://en.wikipedia.org/wiki/Indentation_style#Whitesmiths_style