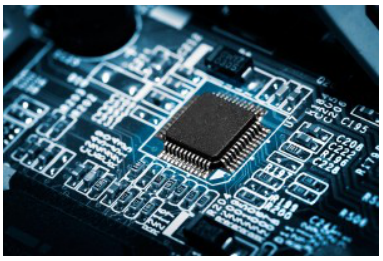
Embedded C Tutorial : A Beginner's Guide

JUNE 17, 2014 BY RICHA



You're surrounded by hundreds of electronic devices today that would have looked out of place even a few decades ago, from the TV and the DVD player to the microwave and digital camera. Most of us think these to be ordinary devices that perform simple functions. But have you ever wondered how they work? As simple as these devices seem on the surface, they each have complex microprocessors (or microcontrollers) running inside their body. These microprocessors perform operations when requested. Like when you press the button on your digital camera to take a photo, the microprocessor will perform the functions necessary to capture the image and store it. The microprocessor's functions are controlled, guided and overseen by the embedded system software. Just like your computer is controlled by the Operating System (like Windows), your camera is controlled by the embedded software. The embedded software and embedded hardware form an embedded system.

Embedded C is the most popular embedded software language in the world. Most embedded software is written in Embedded C. Embedded C is very similar to C– if you know C, you won't have a problem learning Embedded C. **This course can help you learn about the microprocessor environment**. You'll learn how to write embedded software programs, in low level assembly language. Embedded C takes it a step further and lets you write C like programs, suitable for the microprocessor environment. **You can take this introductory course on C to learn more about high level programming.**

In this basic Embedded C tutorial, we're going to give you some info on the important embedded programming concepts. We'll also demonstrate how Embedded C is different from ordinary C.

Basic Concepts of Embedded C and Embedded Programming

Embedded C, even if it's similar to C, and embedded languages in general requires a different kind of thought process to use. Embedded systems, like cameras or TV boxes, are simple computers that are designed to perform a single specific task. They are also designed to be efficient and cheap when performing their task. For example, they aren't supposed to use a lot of power to operate and they are supposed to be as cheap as possible. As an embedded system programmer, you will have simple hardware to work with. You will have very little RAM, ROM and very little processing power and stack space. Your goal is to write programs that are able to leverage this limited processing power for maximum effect. As an ordinary C programmer, you don't have as many constraints.

The reason why most embedded systems use Embedded C as a programming language is because Embedded C lies somewhere between being a high level language and a low level language. Embedded C, unlike low level assembly languages, is portable. It can run on a wide variety of processors, regardless of their architecture. Unlike high level languages, Embedded C requires less resources to run and isn't as complex. Some experts estimate that C is 20% more efficient than a modern language like C++. Another advantage of Embedded C is that it is comparatively easy to debug.

Embedded C Compilers

There are a variety of different compilers on the market, manufactured by different companies, that use Embedded C. One of the more popular ones is the Keil compiler. Because of this, Embedded C is also sometimes known as Keil C.

Embedded C has several keywords that are not present in C (**learn more about the concept of keywords in this course**). These keywords are associated with operations needed by microprocessors. You will need to be familiar with all of them to be able to write Embedded C programs. In this tutorial, we'll show you some of the common ones, like data or idata, bdata and using:

idata/data: The data keyword will store a declared variable in the internal (i) memory of your microprocessor. Take a look at the example below:

```
unsigned char data a;
```

Here, the unsigned char declaration is like a normal C declaration. We just added the data keyword, which tells the microcontroller to store the unsigned char a in the internal data memory.

bdata: The bdata keyword lets you store a declared variable in the bit addressable memory. Take a look at this example:

```
unsigned char bdata a;
```

This is similar to the data declaration we showed you above. You have to access bdata variables in a different way, however.

Using: This keyword lets you execute a function by letting it access a register bank. There are three possible values: 1, 2 and 3.

```
void yourfunction () using 1
{
//statement
}
```

Here, your function called yourfunction will using the register bank at 1. Register banks are a part of the bank-switching technique used by microprocessors. This is an advanced concept, which we cover in more detail in our course.

Embedded C vs. Regular C

While we have discussed the main differences between Embedded C and Regular C, there is another major difference that drastically affects the structure of an Embedded C program and sets it apart from an ordinary C program. When you write a regular C program, you access it from within your operating system software, run it and then, when you're done, you exit back into your operating system (**learn more about writing your own C program with this course**). With an Embedded C program, you have no operating system to fall back on! Your program will, for all intents and purposes, act like the operating system for the embedded device.

Obviously, your program can never stop running, as this will cause the device it's supposed to be operating to crash. Therefore, every Embedded C program must have a structured loop that keeps it running constantly. You can use a simple for loop or a while loop to do that. A normal Embedded C program will follow this format, for example:

```
void main ()
{
//initialize
while (condition) {
//keep doing this
}
}
```

As you know, every C program starts with a main declaration. It's the same with Embedded C. The only difference is that an infinite loop will have to be included, and it will contain the most important parts of the code.

Now that you're familiar with the basics of embedded system programming and Embedded C go ahead and try it out for yourself!

Filed Under: For Students, Technology