C for Embedded Systems

INSTRUCTOR: AARON BARANOFF

How to contact me...

E-mail: aaron_baranoff@hotmail.com

I do have another email addresses aebaranoff@gmail.com and aaron@mataitech.com as well as other work ones the hotmail is the one I have been using for teaching and gets forwarded to my cell phone.

I generally answer emails within 24 hours but on a rare occasion I too get swamped and it could take up to 72 hours. If I don't respond to your email within 48 hours please resend.

Sent me an e-mail so I get each of your e-mails also so you get each others emails.

E-mail and talk to each other you and the people you meet in this business are each other best source of knowledge and networking.

Books and References

- A C Reference Manual (fifth edition)
  by Samuel P. Harbison III & Guy L. Steele Jr.
  Copyright 2002, Prentice Hall
- C Traps and Pitfalls
  by Andrew Koenig
  Copyright 1989, Addison Wesley

Week 2

- Focus
- Prerequisites
- Grades
- Goals
- Outline
- Introduction
- What is Software?
- Why C not C++
- Scope
- Volatiles
- Discussion – Embedded Systems All Around Us

This is Strictly a Practical Class

Give a man a fish and you feed him for a day. Teach him how to fish and you feed him for a lifetime. Lao Tzu

Give a man a fish and you feed him for a day. Teach him how to fish and you feed him for a lifetime. If teach about the science or theory of fishing prior to learning to fish he will starve before he will learn.

Aaron Baranoff's addition to Lao Tzu's Quote

Prerequisites

- Comfortable C programmer
- Willingness to learn
- Willingness to ask questions. Don't be shy.

Grades

- This is a Continuing Education class.
- The are no grades.
- No grades are intending to have you try new things without the downside risks of not knowing the answer first.
- Show up (login), learn and participate and you get a CE.

Class Goals

- Learn the portion of C that you may not have learned before that relates to embedded systems software
- Build in the labs routines that you can use to learn this new knowledge and that can be a foundation for further developments in embedded systems.
- Learn how to avoid common pitfalls by trying these new skills out and asking questions.

Be able to apply what was learned and network with the rest of the class.

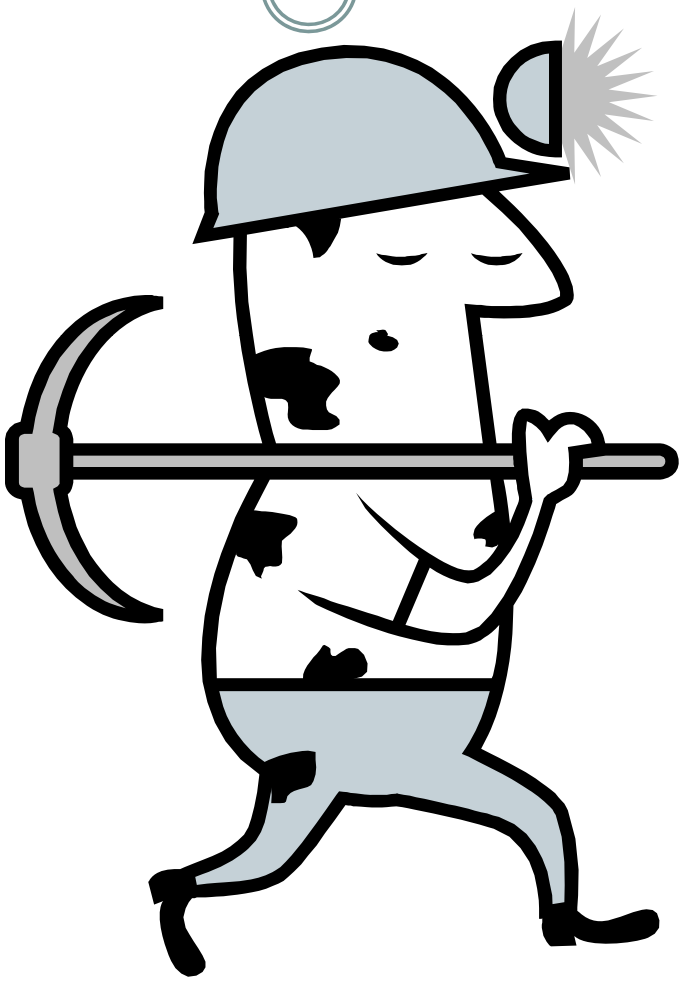
Course Outline

- Introductions
- What is software?
- Scope
- Volatiles
- Registers
- Statics
- Bit manipulation
- Pointer handling
- Arrays and pointers
- Function pointers

Introduction

- My name is Aaron Baranoff I have been writing software commercially since I was 16 years old most of it embedded systems.
- I have spend most of my career working in “bit mines”. Working close to hardware and most of it at chip companies.
- I work full time for L-3 Communications in Anahiem.
- I also own my own small business MataiTech LLC
- I have been teaching with UCI Extension since 2003
- Just like you when I started off nobody taught me what was safe and unsafe when talking to hardware.

Welcome to the "Bit Mines”



“bit mines” was coined by an unknown engineers

What is software?

- Software is simply the sequence of code required to get the proper bits in and out of the proper bytes in the right sequence with the correct timing and present them to the user of the code in a way they can manipulate the process without violating the hardware rules and without violating common sense.
- The user of the software can either be another piece of software or user by way of some hardware.
- This all needs to be done in a way that is both portable, readable and maintainable.

Embedded Systems

- How many CPUs/Or Microcontrollers are they in you daily lives.
- Most are embedded. Most are not PCs or workstations of any sort.
- How many processors does your PC have?
- Let have everyone post up all the items in their daily lives that have a Microprocessor.
- Interesting note, most don't have Windows or Linux. 99% have processors that will be around long after you PC is gone.
- For every PC there are 10s (possibly 100s) of times more non-PC CPUs or Microcontroller sold.

Good new everyone of them need software

Common Sense

- Software Can NOT violate common sense.
- Would you build the roof of the building before you build the foundation.
- Why do people thing software should be designed from the top (the roof) down (the foundation).
- Software's interface to the world is by way of the hardware. No hardware/No software. Not all hardware requires software however all software requires hardware.
- So we need to learn to build a solid foundation to the ground (hardware).
- Most software engineers in the world need embedded software so they don't need to learn about hardware.
- Most software engineers would prefer to think of the software without understanding hardware. That is why they need people who can interface to the hardware.

Common Sense is not as common as it should be.

Why C not C++ or something else

- C is the primary language for device driver writing operating systems and the like.
- Why because it was designed for those purposes.
- C++ and other languages are designed to protect you from yourself. C gives you the power to do almost anything but with the rope to hang yourself and do it efficiently.
- Even Microsoft and the Linux agree on this. If you write a device driver in C++ Microsoft will not support you. Virtually all Linux drivers are in C and those that start as C++ end up being converted to C.
- C++ does things behind your back.
- That is why we are talking C not C++ or other languages.

Why C (con't)

- OK then why was I told you buy a C++ compiler…
- Because C can be compiled with any C++ compiler not vice versa.
- C++ compilers are pickier and that is a good thing as long as I have the ability to do anything I need to and my hardware needs me to do.

Before you learn to Run you need to learn or relearn the stuff you thought you knew.

- Most of you learned C from a book or in school. You learned syntax but, not when to use and when not use different parts of the language.
- C has a lot a features some of which if you want to write good portable and clean code you should NOT use.
- As I stated C gives you the rope to hang yourself.
- So this is where we start showing you what to use and what not to use in the language.

Scope

- Globals
- Locals
- Static

Globals

- Functions and Variables can be global in scope.
- That means function can call or modify as long as they know that they exist.

Globals Examples


```
/* These functions and variables are available to anyone who wants them */
int myglobalvar, myglobalvar2;
/* this is called a prototype – this allows a function find one that is not
   defined yet or is defined in another file. */
extern void anotherglobalfunc(int param1);
void myglobalfunc(int param1)
{
   anotherglobalfunc(param1);
   myglobalvar++; myglobalvar2++;
   /* useful stuff here */
}
void mysecondglobalfunc(int param1)
{
    myglobalfunc(param1);
    myglobalvar++; myglobalvar2++;
   /* other useful stuff here */
}
void anotherglobalfunc(int param1)
{
    myglobalvar++;
   /* useful stuff here */
}
```

Locals

- Variables can be local in scope.
- That means a given variable is only available to the function in which the variable was created.
- Each time the function is called, it is a new instance and it has no memory of what it was before.

Locals Examples


```
int myglobalvar=0, myglobalvar2=1;

void myglobalfunc(int param1)
{
   int localvar=2;

   myglobalvar++; myglobalvar2++;
   localvar++;
   /* useful stuff here */
}
void mysecondglobalfunc(int param1)
{
   int localvar=3; /* no problem with the same name as the function
                      above, they have no relation */
   int localvar2=4;

   myglobalfunc(param1);
   myglobalvar++; myglobalvar2++;
   localvar2++;
   /* other useful stuff here */
}
```

Statics

- Variables and Functions can be static in scope.
- That means function is only available to the file in which it was created.
- A static variable declared outside a function is the same a global but only within a given file.
- A static variable declared inside a function is the same as global but only within a given function.

Static Keyword

- Two purposes for static: limiting the scope, specifying persistence
- Functions or variables can be static. A function or variable (declared outside of a function) is static then it is only visible inside that c file.
- If a static variable is declare inside a function it acts like a global variable for that function. Its value is remembered from one call to the next time that function is called.

```
static int staticfunction(char a) /* only available inside this file */
{
   if (a == 'b')
      return 1;
   else
      return 0;
}
static int var; /* only available inside this file */
int  globalfunction(int temp)
{
   static int staticvar=0; /* only available inside this function */

   temp = staticvar++;

   return temp;
}
```

The ‘volatile’ keyword

- Examples without volatile

```
void testNoVolatile(void)
{
    unsigned int *hw_addr =
             (unsigned int *)0x12345678;

     *hw_addr = 1; /* this may be ignored */
     *hw_addr = 2; /* this may be ignored */
     *hw_addr = 3;
}

void delay(unsigned int times)
{
    unsigned int i;
    for (i=0; i< times; i++);
}

void testing(void)
{
    unsigned int *hw_addr2 = (unsigned int *)0x87654321;

    *hw_addr2 = 0x1000; /* this may be ignored */
    delay(1000000); /* hopeful – delay that may get optimized out */
    *hw_addr2 = 3;
}
```

The ‘volatile’ keyword (cont)

- Example with volatile

```
void testVolatile(void)
{
   volatile unsigned int *hw_addr = (unsigned int *)0x12345678;

    *hw_addr = 1; /* this will NOT be ignored */
    *hw_addr = 2; /* this will NOT be ignored */
    *hw_addr = 3;
}

void fixeddelay(unsigned int times)
{
   volatile unsigned int i;
   for (i=0; i< times; i++);

}
```