Computer Systems and “C” Programming

Maxwell James Dunne
Summer 2016
About: Me

- Undergraduate at UCSC in Computer Engineering and Electrical Engineering.
- Masters from UCSC
- Worked on a mountain lion tracking collar in Graduate School
- Renovated and built new hardware for CMPE118 based off the Ubo32
CMPE 13/L - Computer Systems and C Programming - Spring 2013

Welcome to CMPE 13/L — First Class is Tuesday 02-Apr-2013 @ 2PM.

Background

Computer Systems and C Programming is a class intended to bring you up to speed on programming small and large programs in C. Originally written in 1978, C remains the most popular programming language, and the most used one in terms of numbers of computer programs written in it. There are no prerequisites, but you are expected to be at least somewhat familiar with programming and computers. In this class, we are going to approach C from an embedded paradigm, and all of your programming assignments are going to be on a 16-bit embedded micro, the Microchip PIC16 (or dsPIC33). You will learn how to program in C, how to write modular code, and some of the tips and tricks when dealing with an embedded micro. This is a programming class and you will be writing lots of code. Expect to spend at least 15-20 hours outside of class playing with the code to get things to work.

Acknowledgements

I would like to acknowledge Cyrus Bazeghi, for all his help with the course material, organization, and lecture material. Steve Summit, who taught an Introductory and Intermediate Programming class at the Experimental College at the University of Washington in Seattle, WA, has generously allowed us to use his notes on K&K and other supplementary materials. Microchip Corp. has generously provided slides and software through their academic partner program, and some of the file system slides come from Henry Cheng at UC Davis.

Index of class resources

- General Class Information — class and section times, instructor and TA information.
- Lecture Video — Video files of the lectures, and download information for the right codec.
- Handouts — quizzes, quiz solutions, other helpful handouts.
- Pizzas — announcements, general discussion, and help.

Handouts

- General Course Information
- Course Syllabus
- Handouts
- Extended course description

CMPE-013 Website
# CMPE-013/L Syllabus

## UNIVERSITY OF CALIFORNIA, SANTA CRUZ
BOARD OF STUDIES IN COMPUTER ENGINEERING

CMPE-13/L: COMPUTER SYSTEMS AND C PROGRAMMING
SPRING 2013 LECTURE/LAB CALENDAR

"Notes to accompany K&R," by Steve Summit available on the class website and also available at: [http://www.eskimo.com/~scs/class/knotes/top.html](http://www.eskimo.com/~scs/class/knotes/top.html) [Notes]

<table>
<thead>
<tr>
<th>Lecture</th>
<th>Reading (to be completed before lecture)</th>
<th>Out</th>
<th>Due</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Intro to C Programming</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Hello World!, History, Tools, C Runtime, Heap/Stack</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Comments and Variables</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Literals and Constants</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Functions, Modules, Scope</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Operators and Expressions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Precedence and Associativity</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Scope and Arrays</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Strings and MalloC/Free</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Pointers</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Pointers and Functions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Pointers and Functions (cont) and Structs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>Pointers and Functions (cont) and Structs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>Event Driven Programming</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>State Machines, Modular C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>Unions and Bitfields</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>Exercises and the Environment</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Notes: [http://www.eskimo.com/~scs/class/knotes/top.html](http://www.eskimo.com/~scs/class/knotes/top.html) [Notes]
CMPE-013/L Piazza

University of California, Santa Cruz - Spring 2013
CMPE 13/L: Computer Systems and C Programming

Description

Computer Systems and C Programming is a class intended to bring you up to speed on programming small and large programs in C. Originally written in 1978, C remains the most popular programming language, and the most used one in terms of numbers of computer programs written in it.

There are no prerequisites, but you are expected to be at least somewhat familiar with programming and computers (CMPE12 is recommended).

In this class, we are going to approach C from an embedded paradigm, and all of your programming assignments are going to be on a 16-bit embedded micro, the MicroChip PIC24 (or dsPIC33).

You will learn how to program in C, how to write modular code, and some of the tips and tricks when dealing with an embedded micro.

This is a programming class and you will be writing lots of code. Expect to spend at least 15-20 hours outside of class playing with the code to get things to work.

General Information

Lecture Information:
7:30-8:45 PM
CLASS: Physical Sciences 114
LAB: Jack Baskin Engineering, 109
Ming Ong Lab, 108

Textbooks:

It is also available (both new and used) on Amazon:
http://www.amazon.com/Programming-Language-2nd-Brian-Kernighan/dp/0131103628/ref=sr_1_1?ie=UTF8&sr=8-1&keywords=k%26r+books

And it looks like quite a bit less on AbeBooks:
http://www.abebooks.com/products/srid/978011363627

If you are going to order the book, do so early so you aren’t waiting for it to arrive.

--G
#pin
View on Piazza
CMPE-013/L Extended Course Description

CMPE13/L: Computing Systems and C Programming

19. Computer Systems and C Programming, 3 Introduction to the C programming language as a means for controlling embedded and general computing systems. Continuing the exploration begun in course 12, students move to higher levels of abstraction in the control of complex computer systems. Prerequisites: Courses 12 and 12L. Concurrent enrollment in course 13L required. R. Hughey.

13L Computer Systems and C Programming Laboratory Laboratory sequence in C programming for embedded and general computing systems. Two two-hour laboratories per week. Concurrent enrollment in course 13 required. 3 credits. R. Hughey.

Explanation of prerequisites:

This course follows the “bottom up” approach to teaching programming. The (CMPE15-CMPE12-CMPE11) sequence provides students with the knowledge of how a computer or embedded system is designed and programmed at the assembly and higher-level language levels. Course 12 focuses on the machine level, while Course 13 takes student to a higher level language and extensive training in structured programing habits and tools.

The knowledge of computer memory systems and low-level programming developed in CMPE12 lays the groundwork for understanding C pointers, the classic stumbling block of new programmers who start with C. The course also builds on the exception and interrupt handling from CMPE12, a key concept for following course CMPE12/L.

In addition to cementing students’ understanding of computing systems through the continued hardware/software discussion begin in CMPE12/L, this course provides a C-based alternative to the Java course CMPE11A. C is the fundamental software tool for the computer engineer, much as digital hardware tool.

Students completing CMPE13/CMPE18 will also be able to move directly into CMPE11B, as with transfer students who worked in C on their CMPE11A-articulated course.

Required Skills to Pass the Course

1. Tools for programming, including IDE, compilers, debuggers, and linkers.
2. Knowledge of software design principles, including:
   a. Top-down and bottom-up design
   b. Documentation, inline and external
   c. Coding style
   d. The use of local variables
3. C programming fundamentals
   a. Data types, declarations, assignments
   b. Functions
   c. Conditional and loop structures
   d. Static and dynamic memory allocation and manipulation
   e. I/O library and string functions
4. Interfacing of C with embedded systems
   a. Using C to control hardware
   b. Developing mixed assembly language and C programs.
5. Ability to use C to solve programming problems.
6. Ability to manipulate dynamic data structures such as linked lists, multi-dimensional arrays, and structures.
7. Understanding of computer architecture
CMPE-013/L Books


- **[Notes]**: “Notes to accompany K&R,” by Steve Summit available on the class website and at: http://www.eskimo.com/~scs/cclass/krnotes/top.html

CMPE-013/L Optional Books


CMPE-013/L Optional Books


- **[Zyante]**: “Programming in C,” interactive C book online by Frank Vahid and Smita Bakshi (not free) at: [https://zybooks.zyante.com/#/zybook/7VU9pYQ5ue/tableofcontents](https://zybooks.zyante.com/#/zybook/7VU9pYQ5ue/tableofcontents)
CMPE-013/L Attendance

• Highly Recommended
  – Material builds up quickly
  – Videos available, but not the same as being there

• In class quizzes, at the beginning of class at least once a week
  – No excuses accepted post-quiz.

• At least one lab section per week
  – Easier to get through the labs with help
  – TA will be available then
Spring is 20% Academic Honesty

- Cheating is presenting someone else's work as your own
- All code turned in will be run against a code-checker
- Anyone caught cheating will immediately fail the class and the lab, and be reported to their college
- Copying each other's code is *never* acceptable.

- Don't do it—not worth it.
CMPE-013/L Grading

- Lecture and Lab are one and the same: CMPE-013 and CMPE-013L will get the same grade, same evaluation.

- In class quizzes (once per week): 30%
- Programming assignments (one per week): 70%

- No midterm, no final
CMPE-013/L Lab Work

- All programs we are using can be loaded onto your own laptop for use at any time—they are all free.
- We’re using microchip’s MPLABX IDE and XC32 compiler
- Running on a microcontroller development board by Digilent, the Uno32 or $\mu$32 (essentially the same board)
- You can buy the this hardware directly from Microchip if you want to after the class is over
Lab Kit

Uno32

Basic I/O Shield

PICkit3
Reading Assignment for Today

- [Gonick]: parts 1-3, review your notes from CE-12/L
  - There will be a quiz on this on Thursday
- Check out the Webpage and Piazza Forum
Responsibility

• See Handout
What is a Computer?

• Computer
  – A computer is a machine that manipulates data based on a list of instructions called program.
  – A computer consists of hardware and software.

• Computer Hardware
  – Computer hardware is the physical part of a computer.
  – A typical computer consists of central processing unit (CPU), main memory and external memory, and input and output devices.
  – A CPU consists of control unit (CU), arithmetic and logic unit (ALU), and registers.
Samples of Computer Hardware

A single board computer and a tiny computer.
The von Neumann Computer Architecture

Both programs and data are stored in the same memory
The Harvard Computer Architecture

Programs and data are stored in different memory
Hardware Trends

Moore’s Law (1965):

The number of transistors that can be inexpensively placed on an integrated circuit increases exponentially, doubling approximately every two years.

Based on Moore’s law, every two years, the following approximately double:

- CPU speed at which computers execute their programs.
- The amount of main memory.
- The amount of secondary memory.
Computer Programming Languages

Three types of programming languages

1. Machine code or machine languages
   A sequence of 0’s and 1’s giving machine specific instructions
   Example: 00011001

2. Assembly language
   Using meaningful symbols to represent machine code.
   Example: add hl,de

Assembler: Assembly code → machine code
Disassembler: machine code → assembly code
Computer Programming Languages

3. High-level languages
Similar to everyday English and use mathematical notations (processed by compilers or interpreters)
Example of a C statement:

\[ a = a + 8; \]
Programming Languages

Many differences including:

1. Abstraction type
2. Compiled vs. interpreted
3. Memory management
4. Type system
The C Programming Language

- Procedural
- **Compiled**
- Manual memory management
- **Statically typed**
- Small
- Low overhead

Python

C

Maxwell James Dunne – Summer 2016
Comparison of High-Level Language with Machine Code and Assembly Code

The memory addresses, machine code, and assembly code corresponding to a C statement \( a = a + 8 \) for the Rabbit 3000 8-bit microprocessor.

<table>
<thead>
<tr>
<th>Memory address</th>
<th>Machine code</th>
<th>Assembly code</th>
</tr>
</thead>
<tbody>
<tr>
<td>0X1EA1</td>
<td>0001000100001000000000000000</td>
<td>ld de,0x0008</td>
</tr>
<tr>
<td>0X1EA4</td>
<td>1100010000000000000000000000</td>
<td>ld hl,(sp+0)</td>
</tr>
<tr>
<td>0X1EA6</td>
<td>000111001</td>
<td>add hl,de</td>
</tr>
<tr>
<td>0X1EA7</td>
<td>11010100000000000000</td>
<td>ld (sp+0),hl</td>
</tr>
</tbody>
</table>
10 Reasons to Learn C


1. C is one of foundations for modern information technology and computer science.
2. C is the most commonly used programming languages in industry.
3. C is a standardized programming language with international standards.
4. Writing computer programs is essential to solving complex science and engineering problems.
5. Computer programming is becoming a necessary skill for many professions.
6. Computer programming can develop student’s critical thinking capabilities.
7. C is one of the most commonly used programming languages in colleges and universities.
8. C is the language of choice for programming embedded and mechatronic systems with hardware interface.
9. C excels as a model programming language.
10. Once you have learned C, you can pick up other languages without much difficulty by yourself because all other modern languages borrowed heavily from C.
Structured Programming in C

- A disciplined approach to writing programs in C.
- Clear, easy to test and debug, and easy to modify.
History of C

• C
  – Invented by Ritchie based on B, a simplified version of BCPL.
  – Used to develop Unix operating system and Unix commands
  – Most system software such as OS are written in C or C++
  – Replacement for assembly language for hardware interface.
  – By late 1970's C had evolved to “K & R C”

• C Standards
  – 1st C standard created in 1989 by ANSI, ratified by ISO in 1990. It is called C89. Some call it C90.
  – 2nd C standard was ratified in 1999, called C99.
Just the Facts

• C was developed in 1972 in order to write the UNIX operating system
• C is more "low level" than other high level languages (good for MCU programming)
• C is supported by compilers for a wide variety of MCU architectures
• C can do almost anything assembly language can do
• C is usually easier and faster for writing code than assembly language
Busting the Myths (1.2)

The truth shall set you free...

- C is not as portable between architectures or compilers as everyone claims
  - ANSI language features **are** portable
  - Processor specific libraries are **not** portable
  - Processor specific code (peripherals, I/O, interrupts, special features) are **not** portable

- C is **not** as efficient as assembly
  - A *good* assembly programmer can *usually* do better than the compiler, no matter what the optimization level – C will use more memory
Busting the Myths (2.2)

The truth shall set you free...

- There is **NO SUCH THING** as self documenting code – despite what many C proponents will tell you
  - C makes it possible to write very confusing code – just search the net for obfuscated C code contests...
    (www.ioccc.org)
  - Not every line needs to be commented, but most **blocks** of code should be

```
    i += 1; // increment
```

- Because of many shortcuts available, C is not always friendly to new users – hence the need for comments!

Maxwell James Dunne – Summer 2016
What we will cover in 13/L (1.3)

- Git
- "C" programming
  - Using C in an Embedded Environment
  - Comments
  - Variables, Identifiers and Data Types
  - Literal Constants
  - Symbolic Constants
  - printf() Library Function
  - Operators
  - Expressions and Statements
  - Making Decisions
  - Loops
  - Functions
  - Multi-File Projects & Storage Class Specifiers

Maxwell James Dunne – Summer 2016
What we will cover in 13/L (2.3)

- “C” programming (con’t)
  - Arrays
  - Data Pointers
  - Function Pointers
  - Structures
  - Unions
  - Bit Fields
  - Enumerations
  - Macros with #define

- Advanced Techniques
  - State Machines
  - Recursion
  - Interrupts
  - Program decomposition
  - Abstraction
  - Scope
  - Static / Dynamic Memory allocation
What we will cover in 13/L (3.3)

- Embedded “C” on a microcontroller
  - Specific issues with uControllers
  - Peripheral usage
  - Reading documentation

- Testing and Debugging
  - Commenting
  - Test harnesses
  - Incremental development
  - Issues with embedded debugging

16k RAM
Key things we’ll enforce (1.2)

- **Well documented code**
  - Self-documenting code does not exist!
  - Good variable names and comments are req’d

- **Clean and clear style**
  - More important to adhere to established guidelines than use the “right” style
  - We’ll use (modified) K&R style guide

- **Modularity and decomposition**
  - Code is segmented by functionality
  - Proper use of `.h` and `.c` files
  - Good use of functions for clean implementation
Key things we’ll enforce (2.2)

- State Machines / Event Driven Programming
  - Best way to design reactive systems
  - Makes debugging much easier

- Incremental build and test
  - Every bit of code has a unit test
  - Unit test is designed along with code block
  - Use of pseudo-code and comments
  - End to end code checks

Maxwell James Dunne – Summer 2016
Concept of Ugly/Beautiful

- Beautiful Dog
- Ugly Dog
Concept of Ugly/Beautiful

- Beautiful Car

- Ugly Car
Concept of Ugly/Beautiful

- Beautiful Building
- Ugly Building
Concept of Ugly/Beautiful

- Beautiful Man
- Ugly Man
Concept of Ugly/Beautiful

- Beautiful Woman
- Ugly Woman
Concept of Ugly/Beautiful

Function: RC_Init

Parameters

RCpins: an unsigned short with a 1 in each position to set the pin as an RC servo pin, should be a bitwise OR of the #define'd RC_PORTxxx pins.

Returns

char: SUCCESS or ERROR

Description

Initializes the RC_Servo subsystem, sets each pin as a digital output, and sets the uptime for each pin at 1.5msec, with a period of 20msec.

Notes: Uses TIMER4 with a rollover.

Author: Gabriel Hugh Elkaim, 2011.12.15 16:42

char RC_Init(unsigned short int RCpins) {
    char i, curPin;
    unsigned short int CurrentTime;
    dbprintf("\ninitializing the RC Servo Module.");
    // Check if inputs are in range, and if already initialized
    if ((RCpins == 0x0000) || (RCpins > 0x02FF) || (RCstate != off)) {
        return ERROR;
    }
    RCstate = init;
    // Go through input and set each RC pin direction and force to low
    for (i = 0; i < RCPINCOUNT; i++) {
        curPin = (1 << i);
        if (RCpins & curPin) {
            RCpinMap[numRCPins] = i;
            numRCPins++;
            RC_upTime[i] = SERVOCENTER;
        } else if (RCstate == 0x0101) {
            RC_TRISCLR[i] = rcBitsMap[i]; // Sets pin direction to output
        } else if (RCstate == 0x0102) {
            RC_LATCLR[i] = rcBitsMap[i]; // Forces pin to low state
        }
        dbprintf("\nEnabling pin: %02X", curPin);
    }
}

typedef char C;
typedef long L;
typedef struct a(I t,r,d[3],p[2]){}A;
#define P printf
#define R return
#define V1(f) A f(w)A w;
#define V2(f) A f(a,y)A a,y;
#define DO(n,x) { i =0; n = (n); for(i < n++ +i){(x)}
#define ma(n) (R(1*calloc(n+1)); m(d,i) = (d,i); do tr(r,d) = (d,i); i = i + 1; do (d,i) = (d,i); R zj}
#define ga(t,r,d) I t;{A z=(A)ma((5+tr(r,d))}; z=t,t,z=z;
V1(iota){I n=/w->p[A = ga(0,1,8); DO(n,->p[i]=
V2(plus){I r=w->r,*d=w->d, n=t(r->d); A = ga(0,r,DO(n,->p[i]=a->p[i]+w->p[i]); R zj}}
V2(from){I r=w->r->,d=w->d+1, n=t(r->d); A = (w->t,r,d); R zj}}
V1(box){A z=(w->t,r,d); m(z,p,w->p(n++;a->p(n)); R zj}
V2(cat){I an=tr(a->r,a->d),w=m(r=w->r,w->d),n=a;
A = (w->t,1,8); m(z,p,a->p,an); R zj}
V2(find())
V2(ish){I r=a->r->,a->d, n=t(r,a->p),w=m(r->w->p(n++;a->p.n));
if(n=an) m(z,p,p->p,an,n); R zj}}
V1(sha){A z=ga(0,1,8); m(z,p,w->p,d,w->r)R zj}
V1(id){R wj V1(size){A z=ga(0,0,0); t->p=n=r+2?
pi(i) {P("%d",(i));} I (P("%n");)}
pr(w) A wj I r=w->r, d=w->d, n=t(r->d), DO(r,p) d[if(w->t)DO(n,=P("%");)pr(w->p[i])else DO(n,p)]
C ut[]="+\$f", A("%d[]")={0,p,plus,from,find,0,r,sh,ca},
("%w[]")={0,id,size,iota,box,sha,0};
Concept of Ugly/Beautiful

- Code can be ugly or beautiful
- We will strive to write beautiful code
- While you might not recognize it, yet, you will by the end of this quarter.
Software Architecture

- Often orthogonal to Project Management
- Can be the success or failure of a project
Flow Chart and Pseudo-Code

- Pseudo-code is plain English that explains in coarse steps what the code should do:
  - Not syntax specific
  - Forms the basis for your top-level comment
  - Hides details of programming language
Style Examples

- Adherence to a specific style
  - Variables in camelCase with leading lower case
  - Functions in CamelCase with leading Upper case
  - `#define` (literal constants) are in UPPERCASE
    - Exception: macros can look like functions if they act like one
  - Variable and Function names are descriptive
    - Eg: `backingUpState = TRUE;`
  - Correct use of white space and indentation
  - Correct placement of braces `{}`
Development Tools Data Flow

C Source File ➔ Preprocessor ➔ C Compiler ➔ C Header File

Assembly Source File ➔ Compiler ➔
Compilation

- **Source Code Analysis**
  - “front end”
  - parses programs to identify its pieces
  - variables, expressions, statements, functions, etc.
  - depends on language (not on target machine)

- **Code Generation**
  - “back end”
  - generates machine code from analyzed source
  - may optimize machine code to make it run more efficiently
  - dependent on target architecture

- **Symbol Table**
  - map between symbolic names and items
  - like assembler, but more kinds of information
Fundamentals of C

A Simple C Program

Example

Preprocessor Directives

`#include <stdio.h>`

`#define PI 3.14159`

Header File

Constant Declaration (Text Substitution Macro)

Function

```c
int main(void)
{
    float radius, area;

    //Calculate area of circle
    radius = 12.0;
    area = PI * radius * radius;
    printf("Area = %f", area);
}
```

Variable Declarations

Comment
C Runtime Environment (1.2)

- C Runtime is the “backend” of C:
  - Allocates space for stack
  - Initialize stack pointer
  - Allocates space for heap
  - Copies values from Flash/ROM to variables in RAM that were declared with initial values
  - Clear uninitialized RAM
  - Disable all interrupts
  - Call main() function (where your code starts)
C Runtime Environment (2.2)

- Runtime environment setup code is automatically linked into your application by XC32 compiler
- Code comes from:
  - XC32: `crt0.s / crt0.o` (crt = C RunTime)
- User modifiable if absolutely necessary
Stack/Heap

- Stack grows from bottom “up”
- Heap grows from top “down”
Stack/Heap
Questions?