CMPE-013/L

Computer Systems and “C” Programming

Maxwell James Dunne
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 Uno32
Class Resources

- CANVAS
- Piazza
- Auxiliary Website
  - Storage for lecture videos and slides.

9i + 1a6
# CMPE-013/L Syllabus

## Lecture Topics

<table>
<thead>
<tr>
<th>Lecture</th>
<th>Reading</th>
<th>Out</th>
<th>Due</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Hello World, History, Tools, C Runtime, Heap/Stack</td>
<td>Lab 0: Compiling, Running, and Debugging Due: 10-Apr-13 @ 11:55 PM</td>
<td>Quiz 0 (Intake Quiz)</td>
</tr>
<tr>
<td>2</td>
<td>Chapter 1, all (K&amp;R): Chapter 1, all</td>
<td>Lab 1: Simple Calculator Due: 17-Apr-13 @ 11:55 PM</td>
<td>Lab 0</td>
</tr>
<tr>
<td>3</td>
<td>Literals and Constants</td>
<td>Lab 2: LED Bounce Due: 24-Apr-13 @ 11:55 PM</td>
<td>Lab 1</td>
</tr>
<tr>
<td>4</td>
<td>Functions, Modules, Scope</td>
<td>Lab 3: Matrix Math Library Due: 03-May-13 @ 11:55 PM</td>
<td>Lab 2</td>
</tr>
<tr>
<td>6</td>
<td>Operators and Expressions</td>
<td>Lab 4: RPM Calculator Due: 08-May-13 @ 11:55 PM</td>
<td>Lab 3</td>
</tr>
<tr>
<td>8</td>
<td>Scope and Arrays</td>
<td>Lab 5: Linked List Sort Due: 15-May-13 @ 11:55 PM</td>
<td>Lab 4</td>
</tr>
<tr>
<td>9</td>
<td>Pointers, Pointers and Functions</td>
<td>Lab 6: LCD Driver Due: 22-May-13 @ 11:55 PM</td>
<td>Lab 5</td>
</tr>
<tr>
<td>10</td>
<td>Pointers, Pointers and Functions (cont.) and Structs</td>
<td>Lab 7: BattleBoats Due: 29-May-13 @ 11:55 PM</td>
<td>Lab 6</td>
</tr>
<tr>
<td>12</td>
<td>Handouts: CKO Chapter 5 UNIX Crash Course</td>
<td>Lab 8: Morse Code Decoder Due: 06-Jun-13 @ 11:55 PM</td>
<td>Lab 7</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>Handouts: Chapter 7, all (K&amp;R): Chapter 7, all</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

---

**CMPE-013/L: “C” Programming**

Maxwell James Dunne
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 - 9:45 AM
CLASS: Physical Sciences 114
LAB: Jack Basson 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&qid=1364667071&sr=1-1&keywords=k%26r

And it looks like quite a bit less on Abe Books:
http://www.abebooks.com/products/isbn/9780131103627

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

View on Piazza

X & R availability on Amazon and Abe Books
3/29/13 9:37 PM


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&pid=1364667071&sr=1-1&keywords=k%26r

And it looks like quite a bit less on Abe Books:
http://www.abebooks.com/products/isbn/9780131103627

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

--G

#pin
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

  http://www.soe.ucsc.edu/classes/cmpe013/Spring11/Gonick/
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 after delay, 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
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 ReGrades

- Re-grading of lab assignments will only be done if we have a clerical error (i.e.: we added points wrong) or we somehow missed your work. Note that this does not include you submitting the wrong commit ID in the Google Form for your lab. We will allow a single exception to resubmitting the correct commit ID per student once per quarter.
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 μ32 (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
Responsibility

• See CANVAS
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 Newmann Computer Architecture

Main memory → Registers → Arithmetic logic unit → Control unit → CPU → External memory

Input devices → CPU → Output devices

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
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>00010001000010000000000000000</td>
<td>ld  de,0x0008</td>
</tr>
<tr>
<td>0X1EA4</td>
<td>11000100000000000000000000000</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>11010100000000000000000000000</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

- Because of many shortcuts available, C is not always friendly to new users – hence the need for comments!
What we will cover in 13/L (1.3)

- “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
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
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
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_PORTx_PINx 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 || !RCpins >= 0x0000 || !RCpins <= 0x02FF) {
    return ERROR;
  }

  RCState = INIT;
  // Go through input and set each RC pin direction and force to low
  for (i = 0; i < RC_PINCOUNT; i++) {
    curPin = ((RCpins & 0x01) << i);
    if (RCpinMap[numRCPins] == i) {
      RCpinMap[numRCPins] = i;
      numRCPins++;
      RC_Uptime[i] = SERVO_CENTER;
    }
    RC_TRISCLR[i] = rcBitsMap[i]; // Sets pin direction to output
    RC_LATCLR[i] = rcBitsMap[i]; // Forces pin to low state
    dbprintf("\nEnabling pin: %x", curPin);
  }
  return SUCCESS;
}

typedef char C;
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(a3w)A a3w;
#define DO(n,x) { i=0; _n=(n); for (i<_n++){};
  I "ma(n)({R(I++)mailoc(n)\{n\}r,d,n)} I \"d\"s(I DO(r,d) I \"d\"s(I =i; DO(r,d) i =i; R =i);
  A ga(t,r,d)i \"d\"{}\{A =n\}ma(\{5\}tr(r,d); i=i+1,t=t, R =R;
  V1(iota)\{ I n=\(w->p\); A =ga(0,1,8,8); DO(n->p[i]=
  V2(plus)\{ I r=\(w->r\)^d=\(w->d\); netr(r,d); A =ga(0,r,
  DO(n->p[i]=\{a->p[i]+w->p[i]\}; R =i);
  V2(from)\{ I r=\(w->r\)^d=\(w->d\); n=\(tr(r,d);
  A =ga(w->t,r,d); mv(z->p,\(w->p\(\{n\}a->p\),n); R =i;
  V1(box)\{ A =ga(1,0,0); t=\(z->p(I)\); w Ri;
  V2(cat)\{ I an=\(tr(r,a->d),\)netr(w->r,a->d), n=
  A =ga(w->t,1,8,8); mv(z->p,a->p,an); mv(z->p=an,\n  V2(find)\{
  V2(plus)\{ I r=\(w->r\)^d=\(w->d\); n=\(tr(r,a->p),\)w=\(r->w\);
  A =ga(w->t,r,a->p); mv(z->p,w->p,an); mv(z=\(n\)w=\(n\)w);
  if(n=\(n\)w=\(n\)w) mnv(z->p,\(w->p\(\{n\}a->p\),n); R =i;
  V1(sha)\{ A =ga(0,1,8,w->r); mv(z->p,w->d,a->w->r); R =i;
  V1(id)\{ R wj; V1(size)\{ A =ga(0,0,0); t=\(z->p=\(w->\r
  pi(i)\{P("\",i)\}\}nl\(\{"\n\}pr(w)A wj; I r=\(w->w\)^d=\(w->d\); n=\(tr(r,d); DO(r,p,i(d[
  if(w=\(d->\)DO(n,P("\r
  wj; pr(w->p[i]) else DO(n,p(i[

C ut[]]="{++};",A("um[]"=0,p,plus,from,find,0,r,s,c,at),
("um[]")=0,p,plus,from,find,0,r,s,c,at),
}CMPE-013/L: “C” Programming

Maxwell James Dunne
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 -> Compiler -> C Header File

Assembly Source File
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
Funamentals of C
A Simple C Program

Example

Preprocessor Directives

Header File

#include <stdio.h>

#define PI 3.14159

Constant Declaration (Text Substitution Macro)

Function

int main(void)
{
    float radius, area;

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

Variable Declarations

Comment

Maxwell James Dunne
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?
CMPE-013/L

git

Maxwell James Dunne
Unix/Linux

- Operating System developed by ATT/Bell Labs in 1970’s
- Several variants (BSD, OSX, Solaris, Linux) developed over the years.
- Written in C (and some assembly)
- Unix was designed to be portable, multi-tasking and multi-user in a time-sharing configuration
Basic CLI

- **Command Line Interface:** a means of interacting with a computer where the user issues commands to the program in the form of successive lines of text (command lines).
- Most UNIX commands are using a CLI.
- Commands often have parameters or arguments added to them (e.g.: `ls -a`).
- Contrasts to a GUI (Graphical User Interface).
Basic FileSystems

- How files are stored and organized by the operating system
- All modern OS use a hierarchical file system
  - Starts at a root directory
  - Other directories underneath the root directory
  - Hardware simply looks like another directory (e.g.: /dev).
- Files have several properties that can be specified (e.g: file type, permissions, visibility)
Basic Version Control

• Need to keep track of changes in a collection of documents
• Keep the ability to “roll back” to previous working code
• Coordinate with multiple programmers working on the same code.
• Allows for branch to try something that can then be merged back into development.
Git Theory

- Same theory as all version control systems in that it keeps tracks of changes.
- Except Git differs in a major way.
  - Instead of checking in changes to a server, you ‘commit’ them locally. Only when you need the changes uploaded to the server do you ‘push’.
Git Basics

- Files are only tracked for changes if added to the repository. Files inside the directory are not tracked by default.
- At any point, a ‘commit’ can be made. This will take a snapshot of all tracked files.
- Once ‘commits’ have been made, they are ‘pushed’ back to the server as a separate operation.
Git Practice

- All commands prefaced with `git`
  - clone
  - status
  - add
  - commit
  - push
  - pull
  - log
Git Commands (clone)

- Copies Repository for the first time. Only used when you do not have a copy of the repository

```bash
$ git clone git@soe.ucsc.edu:/classes/cmpe013/spring16/MaxSampleStudent.git
Cloning into 'MaxSampleStudent'...
remote: Counting objects: 6681, done.
remote: Compressing objects: 100% (6395/6395), done.
remote: Total 6681 (delta 4526), reused 400 (delta 237)
Receiving objects: 100% (6681/6681), 76.70 MiB | 1.98 MiB/s, done.
Resolving deltas: 100% (4526/4526), done.
Checking connectivity... done.
```
Git Commands (status)

- Give status of the repository. Showing state of files within the repo

```
$ git status
On branch master
Initial commit
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    README.txt
nothing added to commit but untracked files present (use "git add" to track)
```

```
$ git status
On branch master
Initial commit
Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file:  README.txt
```
Git Commands (add)

- Either adds a new file to the repository or marks a file for commit. This command has no output. Call ‘status’ again to verify operation.
Git Commands (commit)

- Creates a snapshot of all files in the repository at the current time. At any point you can come back to this point so commit early & often.

  $ git commit -m "committing an empty readme for a test"

- Adding the --a automatically adds all tracked files

  $ git commit -am "committing an empty readme for a test"
Git Commands (push/pull)

- Synchronizes the local repository with the remote server. Push should be used frequently to ensure server copy is updated.
- Once pushed to the server, the files are safe from any and all catastrophes that happen to your system.
- **WARNING:** pull will overwrite your local files. Use with discretion.
- We are not sympathetic to your data loss.
Git Commands (log)

- Gives a history of commits for the repo. Call with argument ‘–n 1’ if only last ID is desired.
Git Usage in Labs

- Work out of your repository.
- If working on personal computer, entire tool chain is available on Windows/Mac/Linux.
- Do **NOT** copy files to your repository to commit them.
Git Workflow

1. Clone the repository or pull if the repository if work has been done in a different location.

2. Work on the lab as normal.
   a) Make commits when milestones are reached with useful commit messages. The messages are important, be descriptive.
   b) MPLABX can make commits directly from the IDE. There is no reason not to commit early and often.

3. Push back to the server when session is done or when you want to make sure you have a backup (you don’t need to push every commit, but if you do, then the most you will ever lose is from the latest commit).

Repeat as necessary.
Commit through MPLABX
Lab Submission

1. Finish working on your lab and commit for the last time.
2. Go to gitlab and grab the commit ID you wish to submit.
3. Submit this string to the associated Google Form.
4. Push your files. Without a push the commit ID submitted is useless.
5. The time you submitted the form counts as your turn in time. You must still push the repository.
Lab Submission Verification

1. Move to or create a different directory. **DO NOT DO THIS IN YOUR WORKING REPO.**
2. Clone your repository to this new directory.
3. Checkout your specific commit using the command below
   - `git checkout CommitID` (the long hexadecimal string)
4. Your repository is now exactly as it will be when graded; check and make sure all files are there in the proper state.
5. Delete the newly created repo once you are sure the files are there.