Intro to Programming 2017

Presentation: http://bit.ly/2wdXWJR

COMP ARCH BACKGROUND

How does a computer function?
How does a programming language interact with a computer?
What are you telling the hardware to do?

Binary, Bits

- Binary is base 2, decimal is base 10
- Numbers are abstract, there are many ways to display the same number
- “Base N” means there are N digits in total
- Binary uses 0s and 1s only instead of 0-9
- Using only 0 and 1 makes it easier to store information
- Byte = 8 Bits
- Counting example - count to 7 with binary
    - 000, 001, 010, 011, 100, 101, 110, 111
- Getting decimal from binary equation
    - Binary abc = decimal 4a + 2b + 1c
- Other common bases: Octal (8) and Hex(16)
- Used for human convenience

Logic Gates

- Logic gates are at the core of most parts of the computer
    - memory storage, math
- AND, OR, NOT, NAND, NOR truth tables
- Think of this like programming with hardware
- Logical arithmetic examples (A and B represent true/false variables)
    - NAND + NOT = AND
    - ~(A or B) = (~A and ~B) ~(A and B) = (~A or ~B)
- Logic gates are built from transistors
- Storing a bit of digital information - HI or LOW
    - One bit of memory uses transistors and logic gates to store a charge

    - Many different ways to store a bit, lots of different types of memory/storage
    - Show example of a memory latch built from NAND gates, explain how charge is retained

von Neumann Architecture

- Read Only Memory ROM
    - Nonvolatile - long term storage, retains bits when power lost
    - Usually referred to as “storage”
    - Software, instructions, programs, files
    - “Flash ROM” type of EEPROM (Electrically Erasable Programmable ROM)
        - Each bit uses 2 MOFSET transistors
        - Only erased in large blocks, EEPROM can erase individual bits
        - More compact memory storage (fewer transistors), means faster write and slower erase than EEPROM
    - TI LaunchPad TM4C123 Flash ROM 256 kibibytes
- Random Access Memory RAM
    - Volatile - short term storage, doesn’t retain bits when power lost
    - Usually referred to as “memory”
    - Registers, variables
    - “Static Ram” SRAM type of RAM that uses 6 MOSFET transistors to store a bit
    - TI LaunchPad TM4C123 RAM 32 kibibytes
- Control Unit - controls the execution of the code
    - Instruction Register - current instruction being executed
    - Program Counter - points to next instruction
    - Uses the System Clock
- Processor - CPU
    - Contains lots of logic gates, some temp memory
    - Processor knows what to do by reading current instruction in binary, more on instructions and functions in the in programming section
    - ALU - Arithmetic Logic Unit, performs some arithmetic and logic functions
        - Logic gates for instructions like ADD, SUBTRACT, AND, OR, NOT, etc.
    - Size (number of bits) of instructions is called the Word Length
        - terms like “x32”, 32 bit processor”, “64 bit architecture”, etc
- Input and Output (I/O)

    - There are many forms that I/O can take
    - Keyboards, mouses, displays, network ports, USB, microcontroller I/O ports
    - Microcontroller I/O - more on this in Programming 2 Tech Talk
    - Digital I/O pins are either HI (~3.3v) or LOW (~0v), use to interface with integrated circuits and sensors
    - Analog I/O pins can read and write a range of voltages
- BUS and System Clock
    - The bus transports information between the components using gates
    - Von Neumann architecture uses one BUS for everything, one at a time
- Other architectures: harvard architecture uses multiple BUSes

Computers vs microcontrollers

- Similar architecture
- Microcontrollers typically have a lot less RAM, ROM, and a smaller processor
- Computers typically run Operating Systems (Linux, Windows, Mac OS), Microcontrollers typically run compiled firmware
- Firmware is usually a binary file compiled from source code and begins to run when the microcontroller powers on
- Operating Systems allow you to operate the computer, give instructions, run binary files, and more on your own through a CLI (Command Line Interface, Shell/ Terminal), some will add a GUI (Graphical User Interface) to make it easier
- Linux is a very popular Operating System, based on Unix, more on this later
- Raspberry Pi - Basically like a microcontroller with a Linux operating system, most people classify it as a small computer
- The key part of microntroller programming is memory restrictions. Not a big problem for this project but it matters for your classes and/ or real world work.

PROGRAMMING BACKGROUND

Shells and CLI's (Command Line Interfaces)

- Shell - a shell is used to access an operating system through either a GUI or a CLI
- Windows Shell - cmd.exe, shares some linux commands but not a Unix-Like system
- Unix-Like Systems (Like Linux, Mac OS) - Bash shell most common
    - More Unix shell examples: Tcsh, Ksh, Zsh
    - If you want to be an engineer, you will encounter Linux and the Bash Shell at some point
    - The commands for most Unix shells are about the same, more specifics below
- SSH - allows you to remotely connect to one Shell's CLI from a different Shell's CLI

Scripting Language vs Programming Language

- Most languages do not fall exactly into one category or another
- There are ways to use a language that is commonly used as a programming language as a scripting language and vice versa
- Programing language
    - C, C++, Java
    - Needs to be compiled into an executable
- Scripting language
    - Python, JavaScript, Perl
    - Doesn't need to be compiled
    - Usually offers a way to run in a CLI environment line by line
- Python is interesting because you can use it in the python CLI or save a .py file and then run it. When .py file runs, it compiles line by line. You can also compile it into a .pyc or .pyo file, so it can act like a programming language too

Machine code instructions, assembly, registers, C

- Machine code instructions - binary with the same size as processor word length
    - First few numbers (usually 4) are the instruction type, the rest are arguments

    - An exe file is code that was compiled into binary machine code instructions
- Assembly language is machine code that is simplified with words, directly translates back to binary
- Registers are used to temporarily store variables
- 0001 011 001 000 010 Machine Code in Binary
  ADD R3, R1, R2 Assembly
  r3 = r1 + r2; C
- This is what a C compiler is doing - goes from C to Assembly to Binary
- Great image at https://www.tutorialspoint.com/compiler_design/compiler_design_quick_guide.htm
- C compiler puts variables that are currently being used into registers, if there are not enough registers it also uses the stack. If you use Assembly then you have fine control of the hardware registers and stack, if you use C it gets abstracted, which is good for productivity

Programming Environments

- Github Atom
    - Packages: language-c, gpp-compiler
- https://ideone.com/
- Eclipse
- Kiel
- Visual Studios
- Any text editor - vim, nano

Version Control - Git, Github, SVN

PROGRAMMING SPECIFICS

Overview

- Learn programming and robotics by reading a lot (documentation, classes, tutorials, textbooks) and practicing a lot
- Go over C syntax examples (next page) in an interractive way, tell them to pull up a programming environment and follow along (probably ideone.com)
- Once basic C is covered, move on to Linux/ Atom in Ubuntu VM, compile and run a program there
- Move on to C challenges, can use any environment
- End - C challenge
- Links to tutorials

Unix Shell commands

Navigation

- ls
- cd foldername
- cd ..
- mkdir foldername
- rm -r foldername
- rm filename

Text editors (vim, nano, emacs)

- vim filename
- :w = save
- :q = quit
- Combine commands
- :wq = save and quit
- v = highlight characters
- d = cut
- y = copy
- p = paste

C compile and run with gcc compiler

- gcc -o myProgram myProgram.c
- gcc: name of compiler CLI tool
- -o myProgram: output to myProgram.exe
- ./myProgram
- If it gives a “permission denied” type error when you run it
    - chmod +x myProgram

C Concepts and Examples

Read these notes next to the Code Snippets

Hello World

- Program structure, main, #include, semicolons, comments
- int main() {}
- Main function runs first
- #include statements link external code libraries
- Semicolons end a line of code
- //comments look like this
- printf(“”); //prints a string to the console

Variables, Assignment, Data Types

- Variables, data types, formatting print statements
- Declaring vs initializing
    - Declaring: int x;
    - Initializing: x = 0;
- throw an int into hello world
- Assignment operator =
    - Stores the value on the right into the variable on the left
    - RAM
- There is no String in c - just a group of chars
- Data types have a range of possible values
    - Overflow: when you go outside of that range
- int - integer numbers, range depends on hardware
- double - decimal numbers
- char - one letter, 8 bits, ASCII
- For true false use int 1 and 0
    - Helpful tip: #define TRUE 1
- Adding variables to strings (“number: %d”, num)
    - %d int, long
    - %f double, float
    - %c char
- Escape characters: adding a new line \n

If Statements

- if (condition) {}
  else if (condition) {}
  else {}
- True or false values
- Compare values < > <= >=
- Equality == !=
- if ((2 > 1) == 1) {//condition is true, block will execute}

Functions, Average 2 numbers function, math, int/double behavior

- Function - code block, return type, curly brackets and scope, calling function, parameters, void function
- Function calls are replaced by the variable they return
- Void function: no return
- Difference between int and double
- Math: + - * /
- Show loss of precision with double/ int - it removes everything after the decimal

Arrays, Strings

- Arrays are variables that can hold more than one value
- Each value is identified by an index. Index starts from 0
- Arrays have data types, like all variables
- int numbers[size]; //create
- int number = numbers[3] //access
- Initial value is 0
- Strings are just arrays of characters
- char name[] = “Cole Thompson”;
  - Empty [] tell compiler to calculate array length
  - “” tells compiler that it is an array of characters
  - The last bit in the array is the end of line character (0)
  - The length of the array is (number of characters + 1)
- String library - a set of standard functions that use char[]
  - strlen() example

Exploring For Loops

- For loops for(i=0; i<10; i++) {}
- Index, end condition, step

- First loop: index at initial condition
- Last loop: index one step before end condition is true
- 1) execute code block
- 2) step index
- 3) check end condition
    - If false, go to step 1

Pointers, Memory Structure, String Example

- *char greeting = “hello world”;
- A word is stored byte by btye in memory
- In this example, the variable called **greeting** is a pointer. It’s value is the memory address of the first byte of the String
- char ch1 = *greeting;
- This is called derefrencing a pointer. ch1 now holds the character ‘h’
- When you call a function

Pointers, functions

- When you pass a variable to a function, you only give it the value of the variable.
- The function will copy this value to a new variable
- Any changes made in the function will only be made to this new variable
- How do change an existing variable in another function? Pointers!
- Give the function a pointer to the variable, it will make a copy of the pointer (a copy of the memory address), and then you can access the same spot of memory

C Examples - Code Snippets

NOTE: be careful with COPY/PASTE, quotation marks will often copy wrong

//Helo World

```
#include <stdio.h>
int main() {
   printf("Hello, World!");
   return 0;
}
```

//Variables, Assignment, Data Types

```
#include <stdio.h>
int main() {
   int age = 20;
   printf("My name is Cole.\nI am %d years old.\n", age);

   age = age + 1;     //go into this. Why no int keyword? Why does “age + 1” not work?
   printf("Now I am %d years old.\n", age);

   char ch1 = ‘a’;
   char ch2 = ch1 + 1;
   printf("The first letter of the alphabet is %c and the second is %c.\n", ch1, ch2);
   return 0;
}
```

//If Statements

```
#include <stdio.h>
int main() {
        if ( (2 > 1) == 1 ) {
                printf("true");
        }
        else {
                printf("false");
        }
}
```

//Average 2 numbers function, math, int/double behavior

```
#include <stdio.h>
double avg(double a, double b) {
   return (a + b) / 2;
}
int main() {
   double a = 10;
   double b = 20;
   double myAvg = avg(a, b);
   printf("The average of %f and %f is %f.", a, b, myAvg);
   return 0;
}
```

//Arrays and Strings example

```
#include <stdio.h>
int main() {
   int numbers[5];
   printf("Number at index 2 before setting: %d\n", numbers[2]);
   numbers[2] = 893419;
   printf("Number at index 2 after setting: %d\n", numbers[2]);

   char name[] = "Cole Thompson";
   int nameLen = strlen(name);
   printf("Length of the string \"%s\": %d\n", name, nameLen);
   return 0;
}
```

//Exploring Loops

```
#include <stdio.h>
int factorial(int n) {
        int i;
        int product = 1;
        for (i = n; i >0; i--) {
                product *= i;
        }
        return product;
}
int main() {
        int i;
        for (i = 0; i < 10; i++) {
```

```
            printf("%d ", i);
        }
        printf("\n");
        int f = 10;
        printf("factorial of %d is %d\n", f, factorial(f));
        return 0;
}
```

```
//Pointers
#include <stdio.h>
void stringExample() {
	char * greeting = "hello world!";
	char ch0 = *greeting;			//dereference the pointer
	printf("%s\n", greeting);
	printf("%c\n", ch0);
}

void passVariable(int var) {		//original var not changed
	var += 1;
}

void passPointer(int * varPt) {		//original var changed
	*varPt += 1;				//dereference the pointer
}

void intExample() {
	int myVar = 5;
	passVariable(myVar);
	printf("pass variable: %d\n", myVar);
	passPointer(&myVar);			//& gets the pointer of the variable
	printf("pass pointer: %d\n", myVar);
}

int main() {
	stringExample();
	intExample();
	return 0;
}
```