Course Overview

Computer Systems Organization (Spring 2015)  
CSCI-UA 201, Section 3  

**Instructor:** Joanna Klukowska  
**Teaching Assistants:** Paige Connelly & Carlos Guzman

Slides adapted from  
Andrew Case, Jinyang Li, Mohamed Zahran, Randy Bryant and Dave O’Hallaron
Not that kind of organization
This class adds to your CV...

- C programming
- UNIX / Linux
- X86 assembly
- Low level debugging
- Reverse engineering
What this class is about

- Those details that set hackers apart from novice programmers
  - How your program runs on the hardware
  - Why it fails
  - Why it is slow
- Modern computer systems are covered in layers of abstraction
An x64 processor is screaming along at billions of cycles per second to run the XNU kernel, which is frantically working through all the POSIX-specified abstraction to create the Darwin system underlying OS X, which in turn is straining itself to run Firefox and its Gecko renderer, which creates a Flash object which renders dozens of video frames every second because I wanted to see a cat jump into a box and fall over.

I am a God.

Many layers of abstraction
Course Theme: Abstraction Is Good But Don’t Forget Reality

- Most CS classes emphasize abstraction (abstract data types, asymptotic analysis, ...)
- This class peeks “under-the-hood” into specific details of implementation
- Goal:
  - Make you more effective programmers
    - Debug problems
    - Tune performance
  - Prepare you for later “systems” classes in CS
    - Compilers
    - Operating Systems
    - Networks
    - Computer Architecture
    - Distributed Systems
Reality #1:
int != integer, float/double != real number

```
2147483647 + 1 = 2147483648
1,000,000 + 0.00001 = 1,000,000.00001
Right?
```
Reality #1: int != integer

// Java program:

public class Reality1 {
    public static void main ( String [] args ) {
        int x = 2147483647;
        System.out.printf( "x       = %d \n" +
                          "x + 1 = %d \n", x , x+1);
    }
}

OUTPUT:

x       = 2147483647
x + 1 = -2147483648
Reality #1: int != integer

// C program:

#include <stdio.h>

int main ( ) {
    int x = 2147483647;
    printf ( "x = %d \n" "x + 1 = %d \n", x , x+1);
}

OUTPUT:

x = 2147483647
x + 1 = -2147483648
Reality #1: int != integer

If $x$ is an integer, is $x^2 \geq 0$ always true?

YES

What changes if $x$ is an int?

If $x, y$ and $z$ are integers, is $(x + y) + z = x + (y + z)$ always true?

YES

What changes if they are of type int?

Homework (DNHI): Answer the questions above. For the one(s) that are no longer true for variables of type int, create an example of code in Java that shows the problem.

Challenge: Create similar example(s) in C.
Reality #1: float/double != real number

// Java program:

public class Reality1 {
    public static void main ( String [] args ) {
        float f1 = 1000000f;
        float f2 = 0.00001f;
        System.out.printf ( " f1 = %20.10f\n" +
                        " f2 = %20.10f\n" +
                        "f1 + f2 = %20.10f\n",
                        f1, f2, f1+f2 );
    }
}

OUTPUT:

    f1 = 1000000.000000000000
    f2 = 0.0000100000
    f1 + f2 = 1000000.0000000000
Reality #1: float/double != real number

// C program:
#include <stdio.h>

int main ( ) {
    float f1 = 1000000;
    float f2 = 0.00001;
    printf ( " f1 = %20.10f
" 
    " f2 = %20.10f
"
    "f1 + f2 = %20.10f
",
    f1, f2, f1+f2 );
}

OUTPUT:
    f1 = 1000000.0000000000
    f2 = 0.0000100000
    f1 + f2 = 1000000.0000000000
Reality #1: float/double != real number

If $x$ is a real number, is $x^2 \geq 0$ always true?

**YES**

What changes if $x$ is a float/double?

If $x$, $y$ and $z$ are real numbers, is $(x + y) + z = x + (y + z)$ always true?

**YES**

What changes if they are of type float/double?

**Homework (DNHI):** Answer the questions above. For the one(s) that are no longer true for variables of type float/double, create an example of code in Java that shows the problem.

**Challenge:** Create similar example(s) in C.
Reality #2:
You need to know assembly

• Chances are, you'll never write programs in assembly (compilers are much better at it than people).
• But: understanding assembly helps one understand machine-level execution
  – Debugging
  – Performance tuning
  – Writing system software (e.g. compilers, OS)
  – Reverse engineering software
    • Creating / fighting malware
      – x86 assembly is the language of choice!
Reality #3: Memory Matters

- Memory is not unbounded
  - It must be allocated and managed
- Memory referencing bugs especially wicked
- Memory performance is not uniform
  - Cache and virtual memory effects can greatly affect performance
#include <stdio.h>

int main ( ) {
    int d = 3;
    printf("d = %d\n", d);
    int a[1];
    int i;
    for (i = 0; i < 5; i ++ ) {
        a[i] = 214748364;
    }
    printf("d = %d\n", d);
}
Reality #3: Memory Matters

Copy the values from one 2D array to another. The sizes of both arrays are 2048x2048.

```c
void copyij(int src[2048][2048],
            int dst[2048][2048])
{
    int i,j;
    for (i = 0; i < 2048; i++)
        for (j = 0; j < 2048; j++)
            dst[i][j] = src[i][j];
}
```

```c
void copyji(int src[2048][2048],
            int dst[2048][2048])
{
    int i,j;
    for (j = 0; j < 2048; j++)
        for (i = 0; i < 2048; i++)
            dst[i][j] = src[i][j];
}
```

About 7 times faster on Intel Core i7 3930K.

WHY?
Reality #4: There is more to performance than asymptotic analysis (the Big O)

• (But do not tell your teachers in Data Structures and Algorithms courses that I said that!)

• Constant factors matter too!

• Optimization has to happen at multiple levels: algorithm, data representation, details of implementation.

• Optimizing implementation requires understanding of the underlying system.
Reality #5: Computer do more than just execute programs

• Data moves in and out: I/O systems are critical to program reliability and performance

• Computers communicate with one another via networks
Course Perspective

- Most Systems Courses are Builder-Centric (building things)
  - Computer Architecture
    - Designing a pipelined processor
  - Operating Systems
    - Implement large portions of operating system
  - Compilers
    - Write compiler for simple language
  - Networking
    - Implement and simulate network protocols
Course Perspective

- This course is **programmer-centric**
  - Understanding of underlying systems (to make us more effective programmers)
  - Bring out the hidden hacker in everyone
  - Dissecting the frog
Administrative Matters
Course Website

http://cs.nyu.edu/~joannakl/cs201.03_s15/
“Lab” Machines

• All work for this course should be done on a provided lab machine.

• The “lab” machine is a virtual Ubuntu Linux machine that you should download from the course website.

• You need to install VirtualBox from https://www.virtualbox.org/ in order to be able to use/run your “lab” machine.