**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 Ciesi, Jiaying Li, Mohamed Zahir, Randy Bryant and Dave O'Hallaron

---

**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

---

**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

---

**Not that kind of organization**

---

**Many layers of abstraction**
Reality #1: int != integer, float/double != real number

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

OUTPUT:
```
x     = 2147483647
x + 1 = -2147483648
```

C program:
```c
#include <stdio.h>

int main ( ) {
    int x = 2147483647;
    printf ( "x     = %d 
" +
            "x + 1 = %d 
",
        x , x+1);
    return 0;
}
```

OUTPUT:
```
x     = 2147483647
x + 1 = -2147483648
```

Reality #1: int != integer

Java program:
```java
// Java program:
public class Reality1 {
    public static void main ( String [] args ) {
        float f1 = 1000000f;
        float f2 = 0.00001f;
        System.out.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
```

C program:
```c
#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 );
    return 0;
}
```

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

If you are a real number, is \( x^2 \geq 0 \) always true?

**YES**

What changes if you are a float/double?

**Reality #3: Memory Matters**

```c
#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("a[i] = %d\n", a[i]);
    }
    printf("b = %d\n", d);
}
```

**OUTPUT** (one possibility):

```
d = 3
a[0] = 214748364
b = 3
```

### Homework (DNH):
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
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

Administrative Matters

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 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.