Acknowledgments

- Class notes partially based on
- 211 classes taught at Rutgers in prior years
- Material from textbook site
- Lots of material available on the web (via google search, wikipedia)

Text

- Books:
  - Computer Systems by Randall Bryant and David O’Hallaron … Required text
  - Computer Organization and Design by David Patterson and John Hennessy (4th edition)

Prerequisites: 198:112

What this really means:
- You know at least one programming language C.
- You know something about how to write, run, and test programs.
- Elementary knowledge of math and algorithms
**C resources**

- *The C Programming Language*
  by Brian W. Kernighan & Dennis M. Ritchie
- C tutorial
  - [http://www.le.ac.uk/cc/tutorials/c/](http://www.le.ac.uk/cc/tutorials/c/)
- Intro to C
  - [http://cprog.tomsweb.net/cintro.html](http://cprog.tomsweb.net/cintro.html)

**Grading**

Grading:
- 2 Mid-terms - 30%
- Final Exam – 35%
- Projects – 35% (PA1: 5, PA2: 10, PA3: 10, PA4: 10)

Written Homeworks

Final Exam is cumulative.

No make-up exams except for university sanctioned reasons. Must inform professor before the exam. Don’t assuming informing implies being allowed to take a make-up (see above statement). Make-up will be held the next day, early morning.

Projects are not created equally.

Later projects are harder and hence worth more.

---

**Main Components**

- CPU
  - Executes Instructions
- Memory
  - Stores Programs and data
- BUS
  - Transfers data
  - CPU facing
  - Front Side Bus (FSB)
  - I/O Bus
- Storage
  - Permanent
- I/O devices
  - Input
    - Keypad, Mouse, Touch
  - Output
    - Printer, Screen
  - Both (input and output)
    - USB, WiFi, Touch screen, DISK

---

**Computer Architecture**

620 Mhz  L1 cache 32 KB
Ipad A4 Processor

Max. CPU clock rate 1 GHz (iPad)
Instruction set ARM v7 32 bit RISC
Core 1
  L1 cache 64 KB
  L2 cache 640 KB

Multiple cores in a single chip
1.2 to 1.4 GHz dual core
1.4 to 1.6 GHz quad core

New trends
1.2 to 1.4 GHz dual core
1.4 to 1.6 GHz quad core
Multiple cores in a single chip

Computer Architecture
- Architecture: art or practice of designing
- What’s Inside?
- How is it put together?
- How does it work?
**Von Neumann Architecture**
- Model of a computer that used stores programs
  - Both Data and Program stored in memory
  - Allows the computer to be “Re-programmed”
- CPU is central to the computer

**Instruction Set Architecture**
- Application Software
  - Compiler
  - OS

**Von Neumann in Practice**
- Diagram showing components of a computer system

**Prog Language to Hardware**
- High level Program
  - Assembly level Program
  - Machine level Code

Computer Architecture: ISA, memory, I/O, Power
Specialized HW: GPU, co-processor

**Compiler Hardware**
- Hardware and software
- ISA: Interface between Software & Hardware

**Instruction Set Architecture**
- Application Software
  - Compiler
  - OS

**Von Neumann in Practice**
- Diagram showing components of a computer system

**Prog Language to Hardware**
- High level Program
  - Assembly level Program
  - Machine level Code

Computer Architecture: ISA, memory, I/O, Power
Specialized HW: GPU, co-processor
What will you get out of the course?

- Basic Elements of Computer Systems
  - CPU, Memory, I/O
- Workings of various sub systems
- Design issues
  - Performance vs cost tradeoffs
- Interaction of Software (programs) and the underlying hardware (on which programs execute)

Understanding Processor/CPU performance

- What does a processor do?
  - PC is Program Counter

Measuring Processor Speed

- How long does it take to execute a program
  - quicker the better
- How long does it take to go from point A to Point B
  - Need to Know
  - Speed, Distance (or)
  - Start time, End time
- Speed
  - Constant, Average, Variable
  - RPM, Circumference, distance

CPU Clock

- Every action is driven by a clock in the CPU
- Clock time = 1/
  - Frequency
- Mhz clock = $10^{-6}$
  - seconds
- Ghz clock = $10^{-9}$
  - seconds
- From CPU speed, you know time for 1 clock cycle
Time for a program

- CPU executes various instructions
- A Program has several Instructions
  - How many?
    - Depends on program, compiler
- Each Instruction can take several CPU cycles
  - How many?
    - Depends on the Instruction Set Architecture (ISA)
    - ISA – Learn in this course
- Each cycle has a fixed time based on CPU, BUS speed
  - What is the clock time, memory speed etc?
    - Depends on the hardware, organization
    - Computer Architecture – Learn in this course

CPU Performance Equation

\[ \text{CPU time} = \frac{\text{# of Instructions} \times \text{Cycles Per Instruction} \times \text{Clock cycle time}}{\text{Clock Cycles}} \]

How Good is the Compiler?

Defined by the Architecture

Quantifying Computer components

- CPU Speed
  - Mhz or Ghz, CPU Speed, MIPS, MFLOPS...
  - 1.33 Gmhz ... Intel Atom processor
- Bus Speed
  - Front Side Bus (FSB) ... 533 Mhz Intel Atom
  - Number of channels, Number of data paths
- Memory capacity, memory speed
  - Gigabytes, Mhz x DataRate
  - 166 MHz DDR memory, Quad pump
- Disk capacity, Disk Bandwidth
  - GB, TB, MB/sec
- Power Consumption
  - Watts, mWatts,
  - Battery life time (standby vs active) Watt-Hr

Class of CPU

<table>
<thead>
<tr>
<th></th>
<th>Server</th>
<th>Desktop</th>
<th>Embedded</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cost of System</td>
<td>5K to 1 M</td>
<td>700$ to 5K</td>
<td>100 to 700$</td>
</tr>
<tr>
<td>Cost of CPU</td>
<td>50$ to 1K</td>
<td>70 $ to 200$</td>
<td>1 to $100</td>
</tr>
<tr>
<td>Performance metrics</td>
<td>Throughput, Availability</td>
<td>Response time, price Graphics</td>
<td>Power, Battery life Graphics</td>
</tr>
</tbody>
</table>
Intel Processors

Direct Media Interface - DMI

- Different bus for RAM, I/O and other components
- i3, i5, i7 processors

Core i7 3.5 Ghz
Core i5 3.1 Ghz
Core i3 2-2.5 Ghz

Other Systems

- iPhone
  - 620 Mhz ARM chip
  - SIMD, high performance integer CPU (8-stage pipeline, 675 Dhrystone, 2.1 MIPS)
  - 16 K/16 K cache
  - 0.45 mW/MHz power draw (with cache)
- Wii
  - CPU: PowerPC-based "Broadway" processor, 720 Mhz
  - GPU: ATI "Hollywood" GPU, 243 MHz
- iPad
  - 1GHz Apple A4
  - Upto 10 hours of battery life

Laptop Ratings

IBM ThinkPad T42 (Pentium M Processor 735 1.7GHz, 512MB RAM)

Intel® Core™ 2 Duo P8600 (2.4GHz/1066Mhz FSB/3MB cache), 4G memory, 100 G disk
Moore’s law

- Gordon Moore was an Intel Engineer
- An observation about improvements in hardware
- No of transistors on a chip double every 18 months
- Exponential growth seen in other hardware
  - Memory capacity
    - 2x every 2 years
  - Processors Speed
    - 2x every 18 months
  - Disk capacity
    - 2x every year

Clock speed

Processor Performance
Figure 1.4: Growth in processor performance since the mid-1970s. The chart plots performance relative to the mid-1970s as measured by the SPECint92 benchmark (see Section 1.3.4). Since mid-1970s, processor performance growth was largely technology driven and averaged about 30% per year. The increase in growth to about 50% since then is attributable to more advanced and less costly chip implementation technologies. By 2001, the growth rate is lower due to the increased complexity of chip design. Source: Intel. 

Figure 4.1: DRAM capacity. 

Figure 5.1: Evolution of memory granularity. 

Figure 6.1: Memory Price. 

Retail $50 to $100/GB
Memory Price

Disk Drive Growth

Power Ratings

- Atom Processor 1.6 Ghz, 4 W .. netbook
- AMD Athlon 1.4 Ghz, 64 W .. desktop

Pentium
- 75 MHz 6.0 W 9.4
- 90 MHz 9.0 W 10
- 100 MHz 10.1 W 10.1
- 133 MHz 11.0 W 11.5
- 150 MHz 11.6 W 12.3
- 166 MHz 14.5 W 11.4
- 200 MHz 15.5 W 12.3

30$ to 50 $ per 100 G , 1 TB for 100 $ to
Battery Lifetimes

- Netbook 5 to 6 hours
- Laptop 3 to 6 hours
- iPad 8 to 10 hours
- Smart Phones
- 7 to 8 hours, Standby 24 hours

Battery Life being highlighted

Source: www.hp.com

Battery Technology

Future

- Advanced Architectures
- Multi-core (more than 1 CPU on a chip)
- Performance Accelerators GPU
- Graphic chips (Xbox, Wii, nintendo)
- Probability Processing
- Embedded Computing
  - Processors in vending machines, washer dryers, cars
- Cloud Computing
  - Computing as a utility
- Low Energy Design
  - Green is IN
Course Goals

Architecture course have been traditionally taught in two ways: top-down or bottom-up. We’re going in from the middle. **Programmer-Centric Approach.**

**Goals:**
- Show that by knowing more about the underlying system, one can be more effective as a programmer.
- Write programs that are more reliable and efficient.
- Understand how programs interact with the underlying hardware.
- Learn the ins and outs of a computer’s architecture.
- Learn how to do low-level programming.

Course expectations...

What to expect from the course:
- Will cover key issues and concepts in class.
- Recitations will provide review and teach you the tools you need.
- 4 Programming Projects (Don’t freak out… yet)
- 2 mid-terms and a final (maybe) practice homework and quizzes

So what do I expect of you:
- Come to class
- Read the book (Listening to me is not good enough)
- Work through the problems in the book (not really homework… but it helps)
- Do the projects
- Ask questions (IMPORTANT)

Outline of Topics

Topics: Chapter Numbers from text
- Intro Chapter 1
- C programming
- Information Representation Chapter 2
- Assembly Language Programming (x86) Chapter 3
- Digital Logic Chapter 4.2
- Processor Architecture Chapter 4.3
- Pipelining Chapters 4.4, 4.5
- Memory Hierarchy Chapter 6
- Virtual Memory Chapter 10
- I/O and System buses Chapter 11

Project Information

Projects Summary:
- There will be 4 projects (not as bad as it sounds).
- Most projects will require some degree of programming.
- The high-level programming language of choice is C. No Java.
- Projects are very “Do-able” if you don’t wait until the last minute to start them. If you do… well that’s your fault.
- That being said, I will NOT accept any late projects. No excuses.
- Projects posted on class web-page.

Cheating (the sad reality):
- I know it happens.
- I don’t like it, I won’t tolerate it, and I will be looking for it.
- If you get caught, both parties will be punished.
198:211
Computer Architecture

Week 2/Part 1
Fall 2010

- Topics:
  - Comparison of Java and C
  - C Programming Language Review

Intro to C
- TAs in the recitations will go over C in detail
- Other details: cereal machines, accounts etc
- Compiling, debugging tools (GCC)
- Learn C by programming
  - Don’t wait until Programming assignments are due
  - Start by coding, testing small C programs
- Remember you already know JAVA
  - Learning another language is easy

Why C after Java!!
- It is good to be bilingual or multilingual!
- More job opportunities!!!
- Java is high level Programming language
- C is both high level and low level
- Better understanding of low-level mechanisms
- Better Understand Language-architecture interface… Objective of this course
- Learn C/C++, JAVA, and Python
- Memory-management

Java verse C

Java Program
javac ...

Byte Code (.class)

Java Virtual Machine

Hardware and Operating System

C Program
gcc ...

Compiled Code

java ...
### Java vs C

<table>
<thead>
<tr>
<th>Java</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>object-oriented</td>
<td>function-oriented</td>
</tr>
<tr>
<td>strongly-typed</td>
<td>Flexible (cast)</td>
</tr>
<tr>
<td>No pointers</td>
<td>pointers</td>
</tr>
<tr>
<td>Automatic memory mgmt</td>
<td>Left to programmer</td>
</tr>
<tr>
<td>Strings as type</td>
<td>Only char arrays</td>
</tr>
<tr>
<td>layered I/O model</td>
<td>byte-stream I/O</td>
</tr>
</tbody>
</table>

### Java vs C

```java
public class tvshow
{
    public static void main (String args [])
    {
        System.out.println("Jersey Shore");
    }
}
```

```c
#include <stdio.h>
int main(int argc, char *argv[])
{
    printf("Jersey Shore\n");
    /* \n is linefeed, \t tab */
    System.out.println("Jersey Shore");
}
```

### Data types

```java
main() { int a, b, c, sum;
    a = 1; b = 2; c = 3;
    sum = a + b + c;
    printf("sum is %d", sum);
 }
```

```java
main() { int a, b;
    float c, sum;
    a = 1; b = 2; c = 3.5;
    sum = a + b + c;
    printf("sum is %f", sum);
 }
```

### Numeric data types

- **char**
  - Individual characters (Range 127 to -128)
- **int**
  - Integers
  - Short (-32768 to 32767)
  - Long -2,147,483,648 to 2,147,483,647
- **float**
  - Real numbers 3.4 E +/- 38  (32 bits long)
  - double
  - Real numbers with double precision 3.4 E +/- 308  (64 bits long)
- **Modifiers**
  - Short (16 bit), long (32bit)
  - Signed, unsigned
  - for integers and whole numbers respectively
### Arithmetic Operators

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Operation</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>multiply</td>
<td>$x \times y$</td>
</tr>
<tr>
<td>/</td>
<td>divide</td>
<td>$x \div y$</td>
</tr>
<tr>
<td>%</td>
<td>modulo</td>
<td>$x \mod y$</td>
</tr>
<tr>
<td>+</td>
<td>addition</td>
<td>$x + y$</td>
</tr>
<tr>
<td>-</td>
<td>subtraction</td>
<td>$x - y$</td>
</tr>
</tbody>
</table>

- All associate left to right.
- $* / \%$ have higher precedence than $+ -$.

### Special Operators: ++ and --

- Changes value of variable before (or after) its value is used in an expression.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Operation</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>++</td>
<td>postincrement</td>
<td>$x++$</td>
</tr>
<tr>
<td>--</td>
<td>postdecrement</td>
<td>$x--$</td>
</tr>
<tr>
<td>++</td>
<td>preincrement</td>
<td>$++x$</td>
</tr>
<tr>
<td>&lt;=</td>
<td>predecrement</td>
<td>$--x$</td>
</tr>
</tbody>
</table>

- Pre: Increment/decrement variable before using its value.
- Post: Increment/decrement variable after using its value.

### Examples

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    char weight[4];
    int w;
    w = 140;
    printf("Here is what you weigh now: %i\n", w);
    w--;
    printf("w--: %i\n", w);
    w++;
    printf("++w: %i\n", w);
    printf("post DECR %i \n", --w);
    printf("post INCR %i \n", w++);
    printf("value of w %i \n", w);
    return(0);
}
```

```c
#include<stdio.h>

main(
{
    int i = 3, j = 4, k;
    k = i++ + --j;
    printf("i = %d, j = %d, k = %d", i, j, k);
}
```

### Relational Operators

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Operation</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;</td>
<td>greater than</td>
<td>$x &gt; y$</td>
</tr>
<tr>
<td>&gt;=</td>
<td>greater than or equal</td>
<td>$x \geq y$</td>
</tr>
<tr>
<td>&lt;</td>
<td>less than</td>
<td>$x &lt; y$</td>
</tr>
<tr>
<td>&lt;=</td>
<td>less than or equal</td>
<td>$x \leq y$</td>
</tr>
<tr>
<td>==</td>
<td>equal</td>
<td>$x = y$</td>
</tr>
<tr>
<td>!=</td>
<td>not equal</td>
<td>$x \neq y$</td>
</tr>
</tbody>
</table>

- Result is 1 (TRUE) or 0 (FALSE).
- Note: Don't confuse equality ($=$) with assignment ($\leftarrow$).
Logic Operators

- **Symbol**  | **Operation** | **Usage**
- ! | logical NOT | !x
- && | logical AND | x && y
- || | logical OR | x || y

- Treats entire variable (or value) as TRUE (non-zero) or FALSE (zero).
- Result is 1 (TRUE) or 0 (FALSE).

Bit operators

- In C, there are operators that work on bits of a word
  - & logical AND
  - | inclusive OR
  - ~ NOT
- Example
  - x = 8  y = 7
  - x & y
  - x | y
  - ! x
  - E.g., 72 & 184 = 8 ; 72 | 184 = 248 ;

Variable Declarations

- Variables are used as names for data items.
- Each variable has a type, which tells the compiler how the data is to be interpreted (and how much space it needs, etc.).
- int counter;
- int startPoint;
- Float pi=3.14;
- int is a predefined integer type in C.

#include <stdio.h>

int main()
{
    int pints=1;
    float price = 1.45;
    printf("You want %d pint.
",pints);
    printf("That be $ %f, please.\n", price);
    return(0);
}

Control Structures

- Same control structures as Java. Same syntax
- Conditional
  - if
  - if-else
  - switch
- Iteration
  - while
  - for
  - do-while
- also has the break and continue expressions.
Control Structures

- Same control structures as Java. Same syntax
- Conditional
  - if
  - if-else
  - switch
- Iteration
  - while
  - for
  - do-while
  - also has the break and continue expressions.

Sequencing and grouping

- Statement_1 ; statement_2; statement _n;
- executes each of the statements in turn
- a semicolon after every statement
- not required after a {...} block

The if statement

- Same as Java
  - if (condition,)
    {statements,}
  - else if (condition _2)
    {statements _2}
  - else if (condition _n)
    {statements _n}
  - else {statements _n}
- evaluates statements until find one with non-zero result
- executes corresponding statements

Examples from: http://www.java2s.com/Tutorial/C/

The switch statement

- Allows choice based on a single value
  - switch(expression) {
    - case const1: statements1; break;
    - case const2: statements2; break;
    - default: statementsn;
  }
- Effect: evaluates integer expression
- looks for case with matching value
- executes corresponding statements (or defaults)
The switch statement

```c
int fork;
switch(fork) {
    case 1:
        printf("take left");
        break;
    case 2:
        printf("take right");
        break;
    case 3:
        printf("make U turn");
        break;
    default:
        printf("go straight");
    }
```

Repetition

- C has several control structures for repetition

<table>
<thead>
<tr>
<th>Statement</th>
<th>repeats an action...</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>while(c) {}</code></td>
<td>zero or more times, while c ≠ 0</td>
</tr>
<tr>
<td><code>do (...) while(c)</code></td>
<td>one or more times, while condition is ≠ 0</td>
</tr>
<tr>
<td><code>for (start; cond; update)</code></td>
<td>zero or more times, with initialization and update</td>
</tr>
</tbody>
</table>

while loop and for loop

```c
#include<stdio.h>
main(){
    int i = 0;
    while (1) {
        i = i + 1;
        printf("the value of i is %d\n",i);
        if (i>5) {
            break;
        }
    }
}
```

The break statement

- `break` allows early exit from one loop level

```c
#include<stdio.h>
main(){
    int i = 0;
    while (1) {
        i = i + 1;
        printf("the value of i is %d, i\n",i);
        if (i>5) {
            break;
        }
    }
}
```
The continue statement
- continue skips to next iteration, ignoring rest of loop body

```c
#include<stdio.h>
main(){
  int i;
  for(i = -10; i < 11; i++){
    if (i<0) {
      continue;
    }
    printf(" the value of i is %d\n", i);
  }
}
```

Functions in C
- Functions in C are similar to methods in Java (minus the associated objects).
- Function are pass-by-value.
- Using functions has three aspects:
  1. Writing the function declaration.
  2. Calling the function.
  3. Writing the function body.
- Writing the function declaration is not always needed but generally preferred.

Functions in C
- Declaration (also called prototype)
  ```c
  int Factorial(int n);
  ```
- Function call -- used in expression
  ```c
  a = x + Factorial(f + g);
  ```
  1. evaluate arguments
  2. execute function
  3. use return value in expression

Function Definition
- State type, name, types of arguments
- must match function declaration
- give name to each argument (doesn't have to match declaration)
- int Factorial(int n)
  ```c
  {
    int i;
    int result = 1;
    for (i = 1; i <= n; i++)
      result *= i;
    return result;
  }
  ```
  gives control back to calling function and returns value
Input and Output

- Variety of I/O functions in C Standard Library.
- Must include `<stdio.h>` to use them.

```c
printf("%d\n", counter);
```
- String contains characters to print and formatting directions for variables.
- This call says to print the variable `counter` as a decimal integer, followed by a linefeed (`\n`).

```c
scanf("%d", &startPoint);
```
- String contains formatting directions for looking at input.
- This call says to read a decimal integer and assign it to the variable `startPoint`. (Don’t worry about the `&` yet.)

Output Examples

- This code:
  ```c
  printf("%d is a prime number.\n", 43);
  printf("43 plus 59 in decimal is %d.\n", 43+59);
  printf("43 plus 59 in hex is %x.\n", 43+59);
  printf("43 plus 59 as a character is %c.\n", 43+59);
  ```
- produces this output:
  - 43 is a prime number.
  - 43 + 59 in decimal is 102.
  - 43 + 59 in hex is 66.
  - 43 + 59 as a character is f.

More About Output

- Can print arbitrary expressions, not just variables.
- `printf("%d\n", startPoint - counter);`
  - Print multiple expressions with a single statement.
- `printf("%d %d\n", counter, startPoint - counter);`
- Different formatting options:
  - `%d` decimal integer
  - `%x` hexadecimal integer
  - `%c` ASCII character
  - `%f` floating-point number

Output Examples (continued)

- Formatting instructions can contain additional information:
  - `Min Field Width`
  - `Precision`
  - Float: Num of digits to the right of decimal point
  - Int: Min Number of digits to be printed
  - String: Number of chars from string to print

```c
int iv = 12345;
printf("%2.3d\n", (iv));
printf("%10d\n", (iv));
printf("%10.5f\n", (3.1456123));
printf("%10.2f\n", (3.1456123));
printf("%.2f\n", (3.1456123));
```
Examples of Input

- Many of the same formatting characters are available for user input.
  - `scanf("%c", &nextChar);`
    - reads a single character and stores it in `nextChar`
  - `scanf("%f", &radius);`
    - reads a floating point number and stores it in `radius`
  - `scanf("%d %d", &length, &width);`
    - reads two decimal integers (separated by whitespace), stores the first one in `length` and the second in `width`
- Must use ampersand (`&`) for variables being modified. (Explained later when we talk about pointers)
- Exactly how this matching is done will be covered later.

Comments

- Begins with `//` and ends with `//`
- Can span multiple lines.
- Cannot have a comment within a comment.
- Comments are not recognized within a string.
  - Example: "my/"don't print this/?string"
    - would be printed as: my"don't print this"/string
- As before, use comments to help reader, not to confuse or to restate the obvious

main Function

- Every C program must have a function called `main()`.
- This is the code that is executed when the program is run.
- The code for the function lives within brackets:
  - `main()
    {`  
      `/* code goes here */`
      `}`