CS 61C: Great Ideas in Computer Architecture

Lecture 2: Numbers & C Language

Krste Asanović & Randy Katz

http://inst.eecs.berkeley.edu/~cs61c
Agenda

• Numbers wrap-up
• This is not on the exam!
• Break
• C Primer
• Administrivia, Break
• C Type declarations
• And in Conclusion, ...
Recap: Binary Number Conversion

**Binary → Decimal**

1001010₂ = ?₁₀

<table>
<thead>
<tr>
<th>Binary Digit</th>
<th>Decimal Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0 x 2⁰ = 0</td>
</tr>
<tr>
<td>1</td>
<td>1 x 2¹ = 2</td>
</tr>
<tr>
<td>0</td>
<td>0 x 2² = 0</td>
</tr>
<tr>
<td>1</td>
<td>1 x 2³ = 8</td>
</tr>
<tr>
<td>0</td>
<td>0 x 2⁴ = 0</td>
</tr>
<tr>
<td>0</td>
<td>0 x 2⁵ = 0</td>
</tr>
<tr>
<td>1</td>
<td>1 x 2⁶ = 64</td>
</tr>
</tbody>
</table>

Σ = 74₁₀

**Decimal → Binary**

74₁₀ = ?₂

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Binary (odd?)</th>
</tr>
</thead>
<tbody>
<tr>
<td>74</td>
<td>0</td>
</tr>
<tr>
<td>/2 = 37</td>
<td>1</td>
</tr>
<tr>
<td>/2 = 18</td>
<td>0</td>
</tr>
<tr>
<td>/2 = 9</td>
<td>1</td>
</tr>
<tr>
<td>/2 = 4</td>
<td>0</td>
</tr>
<tr>
<td>/2 = 2</td>
<td>0</td>
</tr>
<tr>
<td>/2 = 1</td>
<td>1</td>
</tr>
</tbody>
</table>

Collect → 1001010₂
Signed Integer Representation

How to represent -5?
Sign & magnitude (8-bit example):

<table>
<thead>
<tr>
<th>sign</th>
<th>7-bit magnitude (0 ... 127)</th>
</tr>
</thead>
</table>

Rules for addition, a + b:

- If \((a>0 \text{ and } b>0)\): add \(b\) to \(a\)
- If \((a>0 \text{ and } b<0)\): subtract \(b\) from \(a\)
- ...
- +0, -0 → are they equal? If so, comparator must handle special case! (If not equal???)

Cumbersome

- "Complicated" hardware: reduced speed / increased power
- Is there a better way?
Going backwards by going forwards

• How to add -1 to a four-bit number?

\[
\begin{align*}
0111_{\text{two}} & = 7_{\text{ten}} \\
????_{\text{two}} & + = -1_{\text{ten}} \\
0110_{\text{two}} & = 6_{\text{ten}}
\end{align*}
\]
Going backwards by going forwards

- Add $10000_{two}$ and throw away fifth bit, result unchanged
  - Result wraps around to same value

\[
\begin{align*}
0111_{two} & = 7_{ten} \\
10000_{two} + 10111_{two} & = 16_{ten} + 7_{ten} + 16_{ten} \\
0111_{two} & = 7_{ten}
\end{align*}
\]
Going backwards by going forwards

- Add ($10000_{two} - 1_{two} = 01111_{two}$) and throw away fifth bit
- One less than original value

\[
\begin{align*}
  0111_{two} & = 7_{ten} \\
  01111_{two} & + 10110_{two} = 16 - 1_{ten} \\
  10111_{two} & = 16 + 6_{ten} \\
  0110_{two} & = 6_{ten}
\end{align*}
\]
### 4-bit Example

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>0111</td>
</tr>
<tr>
<td>+ -3</td>
<td>+ 1101</td>
</tr>
<tr>
<td>4</td>
<td>1 0100</td>
</tr>
<tr>
<td>4</td>
<td>0100</td>
</tr>
<tr>
<td>+ 16</td>
<td>+ 1000</td>
</tr>
<tr>
<td>4 + 16</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
</tbody>
</table>

**Map negative → positive numbers**
- Example for N=4-bit: $-3 \rightarrow 2^4 - 3 = 13$
- “Two’s complement”
- No special rules for adding positive and negative numbers

-8 -7 ... -1 0 1 ... 7
0 1 ... 7 8 9 ... 15

$+ 2^4 = 16$

Extra bit is ignored as doesn’t fit in 4 bits.
Two’s Complement (8-bit example)

<table>
<thead>
<tr>
<th>As Signed Decimal</th>
<th>As Unsigned Decimal</th>
<th>Binary Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>-128</td>
<td>128</td>
<td>1000000000</td>
</tr>
<tr>
<td>-127</td>
<td>129</td>
<td>1000000001</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>-2</td>
<td>254</td>
<td>11111110</td>
</tr>
<tr>
<td>-1</td>
<td>255</td>
<td>11111111</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>00000000</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>00000001</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>127</td>
<td>127</td>
<td>01111111</td>
</tr>
</tbody>
</table>

Most-significant bit (MSB) equals sign when interpreted as a signed number.
Signed versus Unsigned Binary Numbers

• Both are stored as a collection of bits
• The same set of bits can be used to represent a signed number or an unsigned number (or a string, or a picture, or a....)
• It’s up to the operation interpreting the bits

\[
\begin{align*}
1101_{two} & = +13_{ten} \text{ (as an unsigned number)} \\
1101_{two} & = -3_{ten} \text{ (as a signed number)} \\
1101_{two} & = \text{orange (as a color)} \\
1101_{two} & = \text{cat (as a type of animal)} \\
1101_{two} & = \text{whatever you want it to mean...}
\end{align*}
\]
Unary Negation (Two’s Complement)
4-bit Example (-8_{ten} ... +7_{ten})

**Brute Force & Tedious**

<table>
<thead>
<tr>
<th>16_{ten} - 3_{ten}</th>
<th>10000_{two} - 0011_{two}</th>
<th>15_{ten} - 3_{ten}</th>
<th>01111_{two} - 0011_{two}</th>
</tr>
</thead>
<tbody>
<tr>
<td>13_{ten}</td>
<td>1101_{two}</td>
<td>12_{ten}</td>
<td>1100_{two}</td>
</tr>
<tr>
<td></td>
<td></td>
<td>+ 1_{ten}</td>
<td>+ 0001_{two}</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16_{ten} - 13_{ten}</td>
<td>10000_{two} - 1101_{two}</td>
<td>13_{ten}</td>
<td>1101_{two}</td>
</tr>
</tbody>
</table>

"largest" 4-bit number + 1

**Clever & Elegant**

invert
Your Turn

- What is the decimal value of the following binary 8-bit 2’s complement number?

1110 0001\text{\textsubscript{\text{two}}}

<table>
<thead>
<tr>
<th>Answer</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>RED</td>
<td>$33_{\text{ten}}$</td>
</tr>
<tr>
<td>ORANGE</td>
<td>$-31_{\text{ten}}$</td>
</tr>
<tr>
<td>GREEN</td>
<td>$225_{\text{ten}}$</td>
</tr>
<tr>
<td>YELLOW</td>
<td>$-33_{\text{ten}}$</td>
</tr>
</tbody>
</table>
Addition
4-bit Example

Unsigned
\[
\begin{array}{c}
3_{\text{ten}} \\
+ \\
4_{\text{ten}} \\
\hline
7_{\text{ten}} \\
\end{array}
+ \begin{array}{c}
0011_{\text{two}} \\
\hline
0111_{\text{two}} \\
\end{array}
\]

Signed (Two’s Complement)
\[
\begin{array}{c}
3_{\text{ten}} \\
+ \\
4_{\text{ten}} \\
\hline
7_{\text{ten}} \\
\end{array}
+ \begin{array}{c}
0011_{\text{two}} \\
\hline
0111_{\text{two}} \\
\end{array}
\]

No special rules for two’s complement signed addition
Overflow
4-bit Example

Unsigned addition

\[
\begin{array}{c}
13_{\text{ten}} \\
+ 14_{\text{ten}} \\
\hline
27_{\text{ten}}
\end{array}
\quad \begin{array}{c}
1101_{\text{two}} \\
+ 1110_{\text{two}} \\
\hline
11011_{\text{two}}
\end{array}
\]

\text{carry-out and overflow}

\[
\begin{array}{c}
7_{\text{ten}} \\
+ 1_{\text{ten}} \\
\hline
8_{\text{ten}}
\end{array}
\quad \begin{array}{c}
0111_{\text{two}} \\
+ 0001_{\text{two}} \\
\hline
01000_{\text{two}}
\end{array}
\]

\text{no carry-out and no overflow}

Signed addition (Two’s Complement)

\[
\begin{array}{c}
-3_{\text{ten}} \\
+ -2_{\text{ten}} \\
\hline
-5_{\text{ten}}
\end{array}
\quad \begin{array}{c}
1101_{\text{two}} \\
+ 1110_{\text{two}} \\
\hline
11011_{\text{two}}
\end{array}
\]

\text{carry-out but no overflow}

\[
\begin{array}{c}
7_{\text{ten}} \\
+ 1_{\text{ten}} \\
\hline
8_{\text{ten}}
\end{array}
\quad \begin{array}{c}
0111_{\text{two}} \\
+ 0001_{\text{two}} \\
\hline
01000_{\text{two}}
\end{array}
\]

\text{no carry-out but overflow}

Carry-out \rightarrow \text{Overflow}
Addition Overflow Detection

4-bit Example

Unsigned addition
• Carry-out indicates overflow

Signed addition (Two’s Complement)
• Overflow if
  – Signs of operands are equal
  – Sign of result differs from sign of operands
• No overflow is possible when signs of operands differ

Overflow rules depend on the operation (signed vs unsigned addition)
Adding Numbers of Different Bit Widths

Unsigned addition

\[
\begin{array}{c}
1101 \\
10100101 \\
????0010
\end{array}
\]

\[
+ \begin{array}{c}
(4\text{-bit, 13}_{\text{ten}}) \\
(8\text{-bit, 165}_{\text{ten}}) \\
(8\text{-bit, ???}_{\text{ten}})
\end{array}
\]

\[
\begin{array}{c}
00001101 \\
10100101 \\
10110010
\end{array}
\]

\[
+ \begin{array}{c}
(4\text{-bit, 13}_{\text{ten}}) \\
(8\text{-bit, 165}_{\text{ten}}) \\
(8\text{-bit, 178}_{\text{ten}})
\end{array}
\]

Zero extension

• What should we assume for upper four bits of first operand?
• Treat as zeros!
• “Zero-extension”
Adding Signed Numbers of Different Bit Widths

• What should we assume for upper four bits of first operand?
• Copy sign bit (MSB) of operand into upper bits
  – If sign is 1, copy 1s, if sign is 0 copy 0s
• “Sign-extension”
# Sign Extension

<table>
<thead>
<tr>
<th>Decimal</th>
<th>4-bit</th>
<th>8-bit</th>
<th>32-bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>$3_{\text{ten}}$</td>
<td>0011$_{\text{two}}$</td>
<td>0000 0011$_{\text{two}}$</td>
<td>0000 0000 0000 0011$_{\text{two}}$</td>
</tr>
<tr>
<td>$-3_{\text{ten}}$</td>
<td>1101$_{\text{two}}$</td>
<td>1111 1101$_{\text{two}}$</td>
<td>1111 1111 1111 1101$_{\text{two}}$</td>
</tr>
</tbody>
</table>

- When is this relevant?
- Any time you need a wider version of a narrower operand
  e.g., adding two integers of different widths
Your Turn

• Which range of decimals can be expressed with a 6-bit two’s complement number?

<table>
<thead>
<tr>
<th>Answer</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>RED</td>
<td>-32 ... 32</td>
</tr>
<tr>
<td>GREEN</td>
<td>-64 ... 63</td>
</tr>
<tr>
<td>ORANGE</td>
<td>-31 ... 32</td>
</tr>
<tr>
<td>YELLOW</td>
<td>-32 ... 31</td>
</tr>
</tbody>
</table>
Agenda

• Numbers wrap-up
• This is not on the exam!
• Break
• C Primer
• Administrivia, Break
• C Type declarations
• And in Conclusion, ...
HotChips 2017: Brains & Dinosaurs

• Top conference introducing new processor chips
- The Matrix Unit: 65,536 (256x256) 8-bit multiply-accumulate units
- 700 MHz clock rate
- Peak: 92T operations/second
  - 65,536 * 2 * 700M
- >25X as many MACs vs GPU
- >100X as many MACs vs CPU
- 4 MiB of on-chip Accumulator memory
- 24 MiB of on-chip Unified Buffer (activation memory)
- 3.5X as much on-chip memory vs GPU
- Two 2133MHz DDR3 DRAM channels
- 8 GiB of off-chip weight DRAM memory
Z14, IBM Mainframe, 50+ years on

**Micro-Architecture**
- 10 cores per CP-chip
- 5.2GHz
- Cache Improvements:
  - 128KB I$ + 128KB D$
  - 2x larger L2 D$ (4MB)
  - 2x larger L3 Cache
  - symbol ECC
- New translation & TLB design:
  - Logical-tagged L1 directory
  - Pipelined 2nd level TLB
  - Multiple translation engines
- Pipeline Optimizations:
  - Improved instruction delivery
  - Faster branch wakeup
  - Improved store hazard avoidance
  - 2x double-precision FPU bandwidth
  - Optimized 2nd generation SMT2
- Better Branch Prediction:
  - 33% Larger BTB1 & BTB2
  - New Perceptron & Simple Call/Return Predictor

**Architecture**
- PauseLess Garbage Collection
- Vector Single & Quad precision
- Long-multiply support (RSA, ECC)
- Register-to-register BCD arithmetic

**Accelerators**
- Redesigned in-core crypto-accelerator
- Improved performance
- New functions (GCM, TRNG, SHA3)
- Optimized in-core compression accelerator
  - Improved start/stop latency
  - Huffman encoding for better compression ratio
  - Order-preserving compression
z14 On-Drawer and System Topology

CP chip, 696 sqmm, 14nm, 17 layers of metal
- 10 cores, each 2+4MB I+D L2 cache
- Shared 128MB L3 cache
SC chip, 696 sqmm, 14nm, 17 layers of metal
- System interconnect & coherency logic
- Shared 672MB L4 cache

Max System:
- 24 CP sockets in SMP interconnect
- 32TB RAIM-protected memory
- 40 PCI gen3x16 fanouts to IO-drawers
- 320 IO cards
Break!
Agenda

• Numbers wrap-up
• This is not on the exam!
• Break
• C Primer
• Administrivia, Break
• C Type declarations
• And in Conclusion, ...
Levels of Representation

![Diagram showing Levels of Representation]

- **High Level Language Program (e.g., C)**
- **Assembly Language Program (e.g., RISC-V)**
- **Machine Language Program (RISC-V)**
- **Compiler**
- **Assembler**
- **Machine Interpretation**
- **Hardware Architecture Description (e.g., block diagrams)**
- **Architecture Implementation**
- **Logic Circuit Description (Circuit Schematic Diagrams)**

### Current Focus

```
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
```

- `lw t0, 0(s2)`
- `lw t1, 4(s2)`
- `sw t1, 0(s2)`
- `sw t0, 4(s2)`

Anything can be represented as a number, i.e., data or instructions:

```
0000 1001 1100 0110 1010 1111 0101 1000
1010 1111 0101 1000 0000 1001 1100 0110
1100 0110 1010 1111 0101 1000 0000 1001
1101 1100 0110 1010 1111
```
Introduction to C
“The Universal Assembly Language”

• Languages used in 61C:
  - C
  - Assembly
  - Python used in two labs

• Prior programming experience helpful, e.g.
  - Java
  - C++
Survey

Who has never written a program in Java or C, C++, Objective-C?
Hello World

C
#include <stdio.h>

int main(void) {
    printf("Hello World!\n");
    return 0;
}

Java
public class L02_HelloWorld {
    public static void main(String args[]) {
        System.out.println("Hello World!");
    }
}
Compilation & Running

$ gcc HelloWorld.c
$ ./a.out
Hello World!
$
C Compilation Simplified Overview

(more later in course)

foo.c → Compiler → foo.o

bar.c → Compiler → bar.o

Linker → lib.o

Pre-built object file libraries

Machine code executable file

C source files (text)

Compiler/assembler combined here

Machine code object files

CS 61c Lecture 2: C Programming Language
Compilation versus Interpretation

C (compiled)

• Compiler (& linker) translates source into machine language
• Machine language program is loaded by OS and directly executed by the hardware

Python (interpreted)

• Interpreter is written in some high-level language (e.g. C) and translated into machine language
• Loaded by OS and directly executed by processor
• Interpreter reads source code (e.g. Python) and “interprets” it
Java “Byte Code”

- Java compiler (javac) translates source to “byte code”
- “Byte code” is a particular assembly language
  - Just like i86, RISC-V, ARM, ...
  - Can be directly executed by appropriate machine
    - implementations exist(ed), not commercially successful
  - More typically, “byte code” is
    - interpreted on target machine (e.g. i86) by java program
    - compiled to target machine code (e.g. by JIT)
  - Program runs on any computer with a “byte code” interpreter (more or less)
Compilation

• Excellent run-time performance:
  – Much faster than interpreted code (e.g. Python)
  – Usually faster than Java (even with JIT)

• Note: Computers only run machine code
  – Compiled application program, or
  – Interpreter (that interprets source code)
Compilation: Disadvantages

- Compiled files, including the executable, are
  - architecture-specific, depending on processor type
    - e.g., RISC-V vs. ARM
  - and the operating system
    - e.g., Windows vs. Linux
- Executable must be rebuilt on each new system
  - i.e., “porting your code” to a new architecture
- “Change → Compile → Run [repeat]” iteration cycle can be slow during development
  - Recompile only parts of program that have changed
  - Tools (e.g. make) automate this
C Dialects

$ gcc x.c

```c
int main(void) {
    const int \textbf{SZ} = 5;
    int a[\textbf{SZ}]; // declare array
    for (int \textbf{i=0}; \textbf{i}<\textbf{SZ}; \textbf{i}++) a[\textbf{i}] = 0;
    return 0;
}
```

$ gcc --ansi --Wpedantic x.c

```c
#define \textbf{SZ} 5
int main(void) {
    int a[\textbf{SZ}]; /* declare array */
    int \textbf{i};
    for (\textbf{i=0}; \textbf{i}<\textbf{SZ}; \textbf{i}++) a[\textbf{i}] = 0;
    return 0;
}
```
C Pre-Processor (CPP)

- C source files pass through macro processor, CPP, before compilation
- CPP replaces comments with a single space
- CPP commands begin with “#”
  - `#include "file.h" /* Inserts file.h */`
  - `#include <stdio.h> /* Loads from standard loc */`
  - `#define M_PI (3.14159) /* Define constant */`
  - `#if/#endif /* Conditional inclusion of text */`
- Use –save-temps option to gcc to see result of preprocessing
Agenda

• Numbers wrap-up
• This is not on the exam!
• Break
• C Primer
• Administrivia, Break
• C Type declarations
• And in Conclusion, ...
Administratrivia

• Notify us of all known variances
  – DSP special accommodations: Email head TA (Steven Ho)
  – Exam conflicts: Fill out the form on Piazza
    (special accommodations can be made only in exceptional cases)
  – If you cannot get an instructional account, email head TA

• Notify head TA by Tuesday, 9/5/2017
  – see http://www-inst.eecs.berkeley.edu/~cs61c/fa17/
Break!
Agenda

• Numbers wrap-up
• This is not on the exam!
• Break
• C Primer
• Administrivia, Break
• **C Type declarations**
• And in Conclusion, ...
Typed Variables in C

```c
int a = 4;
float f = 1.38e7;
char c = 'x';
```

- Declare before use
- Type cannot change
- Like Java

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>integers, positive or negative</td>
<td>0, 82, -77, 0xAB87</td>
</tr>
<tr>
<td>unsigned int</td>
<td>ditto, no negatives</td>
<td>0, 8, 37</td>
</tr>
<tr>
<td>float</td>
<td>(single precision) floating point</td>
<td>3.2, -7.9e-10</td>
</tr>
<tr>
<td>char</td>
<td>text character or symbol</td>
<td>‘x’, ‘F’, ‘?’</td>
</tr>
<tr>
<td>double</td>
<td>high precision/range float</td>
<td>1.3e100</td>
</tr>
<tr>
<td>long</td>
<td>integer with more bits</td>
<td>427943</td>
</tr>
</tbody>
</table>
Constants and Enumerations in C

• Constants
  – Assigned in typed declaration, cannot change
  – E.g.
    ▪ const float pi = 3.1415;
    ▪ const unsigned long addr = 0xaf460;

• Enumerations

```c
#include <stdio.h>

int main() {
    typedef enum {red, green, blue} Color;
    Color pants = green;
    switch (pants) {
        case red:
            printf("red pants are hip\n"); break;
        case green:
            printf("green pants are weird\n"); break;
        default:
            printf("yet another color\n");
    }
    printf("pants = %d\n", pants);
}
```
Integers: Python vs. Java vs. C

<table>
<thead>
<tr>
<th>Language</th>
<th>sizeof(int)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Python</td>
<td>&gt;=32 bits (plain ints), infinite (long ints)</td>
</tr>
<tr>
<td>Java</td>
<td>32 bits</td>
</tr>
<tr>
<td>C</td>
<td>Depends on computer; 16 or 32 or 64</td>
</tr>
</tbody>
</table>

- **C: int**
  - integer type that target processor works with most efficiently

- **Only guarantee:**
  - sizeof(long long) ≥ sizeof(long) ≥ sizeof(int) ≥ sizeof(short)
  - Also, short >= 16 bits, long >= 32 bits
  - All could be 64 bits

- **Impacts portability between architectures**
Variable Sizes: Machine Dependent!

```c
#include <stdio.h>
int main(void) {
    printf("sizeof ... (bytes)\n");
    printf("char: %lu\n", sizeof(char));
    printf("short: %lu\n", sizeof(short));
    printf("int: %lu\n", sizeof(int));
    printf("unsigned int: %lu\n", sizeof(unsigned int));
    printf("long: %lu\n", sizeof(long));
    printf("long long: %lu\n", sizeof(long long));
    printf("float: %lu\n", sizeof(float));
    printf("double: %lu\n", sizeof(double));
}
```

<table>
<thead>
<tr>
<th>Type</th>
<th>Size (bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>1</td>
</tr>
<tr>
<td>short</td>
<td>2</td>
</tr>
<tr>
<td>int</td>
<td>4</td>
</tr>
<tr>
<td>unsigned int</td>
<td>4</td>
</tr>
<tr>
<td>long</td>
<td>8</td>
</tr>
<tr>
<td>long long</td>
<td>8</td>
</tr>
<tr>
<td>float</td>
<td>4</td>
</tr>
<tr>
<td>double</td>
<td>8</td>
</tr>
</tbody>
</table>
Boolean

• No boolean datatype in C
  – Declare if you wish:
    
    ```
    typedef int boolean;
    const boolean false = 0;
    const boolean true = 1;
    ```

• What evaluates to FALSE in C?
  – 0 (integer)
  – NULL (a special kind of pointer: more on this later)

• What evaluates to TRUE in C?
  – Anything that isn’t false is true
  – Similar to Python:
    only 0’s or empty sequences are false, everything else is true!
Functions in C

- Like Java
- Declare return & argument types
- `void` for no value returned
- Functions MUST be declared before they are used
Uninitialized Variables

Code

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

void undefined_local() {
    int x;  /* undefined */
    printf("x = %d\n", x);
}

void some_calc(int a) {
    a = a%2 ? rand() : -a;
}

int main(void) {
    for (int i=0; i<5; i++) {
        some_calc(i*i);
        undefined_local();
    }
}
```

Output

```
$ gcc test.c
$ ./a.out
x = 0
x = 16807
x = -4
x = 282475249
x = -16
```
Struct’s in C

• Struct’s are structured groups of variables
• A bit like Java classes, but no methods
• E.g. \#include <stdio.h>

```c
int main(void) {
    typedef struct { int x, y; } Point;
    Point p1;
    p1.x = 0;  p1.y = 123;

    Point p2 = { 77, -8 };
    printf("p2 at (%d,%d)\n", p2.x, p2.y);
}
```
More C ...

• Lecture does not cover C completely
  – You’ll still need your C reference for this course
  – K&R, *The C Programming Language*
  – & other references on the course website

• Next few lectures’ focus:
  – Pointers & Arrays
  – Memory management
And In Conclusion,…

- Signed integers represented in 2’s complement
- C Programming Language
  - Popular (still!)
  - Similar to Java, but
    - no classes
    - explicit pointers (next lecture)
  - Beware
    - variables not initialized
    - variable size (# of bits) is machine & compiler dependent
- C is compiled to machine code
  - Unlike Python or Java, which are interpreted
  - Compilation is faster than interpretation