And in review...

META: We often make design decisions to make HW simple

- We represent “things” in computers as particular bit patterns: N bits => 2^N things
- These 5 integer encodings have different benefits; 1s complement and sign/mag have most problems.

- unsigned (C99's uintN_t):
  00000 00001 ... 01111 10000 ... 11111

- 2’s complement (C99’s intN_t) universal, learn!

  00000 00001 ... 01111 10000 ... 11111

Overflow: numbers =; computers finite, errors!

META: Ain’t no free lunch

- First stretchable OLED!
  ⇒ Still in the early research stage, but engineers at sister campus UCLA have developed an organic light-emitting diode that stretches, which could lead to electronics that can be rolled up like cloth.

- www.technologyreview.com/computing/38439/

“Before this class, I (student) would say I am a solid C programmer”

a) Strongly disagree (never coded, and I don’t know Java or C++)
b) Mildly disagree (never coded, but I do know Java and/or C++)
c) Neutral (I’ve coded a little in C)
d) Mildly agree (I’ve coded a fair bit in C)
e) Strongly agree (I’ve coded a lot in C)

Has there been an update to ANSI C?

- Yes! It’s called the “C99” or “C9x” std
  ⇒ You need “gcc -std=c99” to compile

- References
  http://home.tiscalinet.ch/t_wolf/tw/c/c9x_changes.html

- Highlights
  - Declarations in for loops, like Java (§15)
  - Java-like // comments (to end of line) (§10)
  - Variable-length non-global arrays (§33)
  - <inttypes.h>: explicit integer types (§38)
  - <stdbool.h> for boolean logic def’s (§35)

Disclaimer

- Important: You will not learn how to fully code in C in these lectures! You’ll still need your C reference for this course.
  - K&R is a must-have reference
  - Check online for more sources
  - “JAVA in a Nutshell,” O’Reilly.
    - Chapter 2, “How Java Differs from C”
  - Brian Harvey’s course notes
  - On CS61C class website

Compilation: Overview

C compilers take C and convert it into an architecture specific machine code (string of 1s and 0s).

- Unlike Java which converts to architecture independent bytecode.
- Unlike most Scheme environments which interpret the code.
- These differ mainly in when your program is converted to machine instructions.
- For C, generally a 2 part process of compiling .c files to .o files, then linking the .o files into executables. Assembling is also done (but is hidden, i.e., done automatically, by default)
Compilation: Advantages

• Great run-time performance: generally much faster than Scheme or Java for comparable code (because it optimizes for a given architecture)

• OK compilation time: enhancements in compilation procedure (Makefiles) allow only modified files to be recompiled

Compilation: Disadvantages

• All compiled files (including the executable) are architecture specific, depending on both the CPU type and the operating system.

• Executable must be rebuilt on each new system.
  - Called “porting your code” to a new architecture.

• The “change→compile→run [repeat]” iteration cycle is slow

C Syntax: main

• To get the main function to accept arguments, use this:
  ```c
  int main (int argc, char *argv[])
  ```

• What does this mean?
  - `argc` will contain the number of strings on the command line (the executable counts as one, plus one for each argument). Here `argc` is 2:
  ```c
  unix% sort myFile
  ```
  `argv` is a pointer to an array containing the arguments as strings (more on pointers later).

C Syntax: Variable Declarations

• Very similar to Java, but with a few minor but important differences

• All variable declarations must go before they are used (at the beginning of the block)*

• A variable may be initialized in its declaration; if not, it holds garbage!

• Examples of declarations:
  - correct: {
    ```c
    int a = 0, b = 10;
    ...
    ```
  - Incorrect:* for `(int i = 0; i < 10; i++)`

  `C99 overcomes these limitations`

Address vs. Value

• Consider memory to be a single huge array:
  - Each cell of the array has an address associated with it.
  - Each cell also stores some value.
  - Do you think they use signed or unsigned numbers? Negative address?!

• Don’t confuse the **address** referring to a memory location with the **value** stored in that location.

Pointers

• An address refers to a particular memory location. In other words, it points to a memory location.

• **Pointer**: A variable that contains the address of a variable.
Pointers

• How to create a pointer:
  & operator: get address of a variable
  int *p, x;  \[ p \to x \]
  \[ x = 3; \]
  \[ p = \&x; \]

• How get a value pointed to?
  * "dereference operator": get value pointed to
    printf("p points to %d\n", *p);

Pointers and Parameter Passing

• Java and C pass parameters “by value”
  procedure/function/method gets a copy of the parameter, so changing the copy cannot change the original
  void addOne (int x) {
    x = x + 1;
  }
  int y = 3;
  addOne (y);
  y is still = 3

• Pointers are used to point to any data type (int, char, a struct, etc.).

• Normally a pointer can only point to one type (int, char, a struct, etc.).
  • void * is a type that can point to anything (generic pointer)
  • Use sparingly to help avoid program bugs... and security issues... and a lot of other bad things!

Pointers and Parameter Passing

• How to change a variable pointed to?
  • Use dereference * operator on left of =
  \[ p \to x \]
  \[ *p = 5; \]

• How to change a variable pointed to?
  • Use dereference * operator on left of =
  \[ p \to x \]
  \[ *p = 5; \]

Peer Instruction Question

void main(); {
  int *p, x=5, y; // init
  y = *(p = \&x) + 1;
  int z;
  flip-sign(p);
  printf("x=%d,y=%d,p=%d\n",x,y,p);
}
flip-sign(int *n){*n = -(*n)}

How many syntax+logic errors in this C99 code?

#Errors
a)1
b)2
c)3
d)4
e)5
Peer Instruction Answer

```c
void main(){
    int *p, x=5, y; // init
    y = *(p = &x) + 1;
    int z;
    flip-sign(p);
    printf("x=%d,y=%d,p=%d\n",x,y,*p);
}  
flip-sign(int *n){*n = -*n;)
```

How many syntax+logic errors in this C99 code?

I get 5...
(signed ptr print is logical err)

And in conclusion...

- All declarations go at the beginning of each function except if you use C99.
- All data is in memory. Each memory location has an address to use to refer to it and a value stored in it.
  - A pointer is a C version of the address.
    - "follows" a pointer to its value
    - Gets the address of a value

C vs. Java™ Overview (1/2)

<table>
<thead>
<tr>
<th>Java</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Object-oriented (OOP)</td>
<td>• No built-in object abstraction. Data separate from methods.</td>
</tr>
<tr>
<td>• &quot;Methods&quot;</td>
<td>• &quot;Functions&quot;</td>
</tr>
<tr>
<td>• Class libraries of data structures</td>
<td>• C libraries are lower-level</td>
</tr>
<tr>
<td>• Automatic memory management</td>
<td>• Manual memory management</td>
</tr>
<tr>
<td></td>
<td>• Pointers</td>
</tr>
</tbody>
</table>

C vs. Java™ Overview (2/2)

<table>
<thead>
<tr>
<th>Java</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>• High memory overhead from class libraries</td>
<td>• Low memory overhead</td>
</tr>
<tr>
<td>• Relatively Slow</td>
<td>• Relatively Fast</td>
</tr>
<tr>
<td>• Arrays initialize to zero</td>
<td>• Arrays initialize to garbage</td>
</tr>
<tr>
<td>• Syntax: /* comment */</td>
<td>• Syntax: /* comment */</td>
</tr>
<tr>
<td></td>
<td>// comment</td>
</tr>
<tr>
<td></td>
<td>System.out.print</td>
</tr>
<tr>
<td></td>
<td>printf</td>
</tr>
</tbody>
</table>

*You need newer C compilers to allow Java style comments, or just use C99

C syntax: True or False?

- What evaluates to FALSE in C?
  - 0 (integer)
  - NULL (pointer: more on this later)
  - no such thing as a Boolean*
- What evaluates to TRUE in C?
  - everything else...
    - (same idea as in scheme: only #f is false, everything else is true!)

*Boolean types provided by C99's stdbool.h

C syntax: flow control

- Within a function, remarkably close to Java constructs in methods (shows its legacy) in terms of flow control
  - if-else
  - switch
  - while and for
  - do-while