inst.eecs.berkeley.edu/~cs61c

CS61C : Machine Structures

Lecture 3 – Introduction to the C Programming Language (pt 1)

2007-01-22

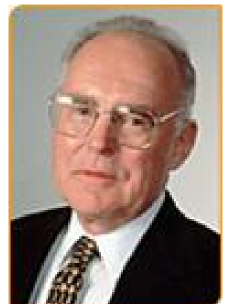
There is one handout today at the front and back of the room!

Lecturer SOE Dan Garcia

www.cs.berkeley.edu/~ddgarcia

HP overcomes Moore's Law? ⇒

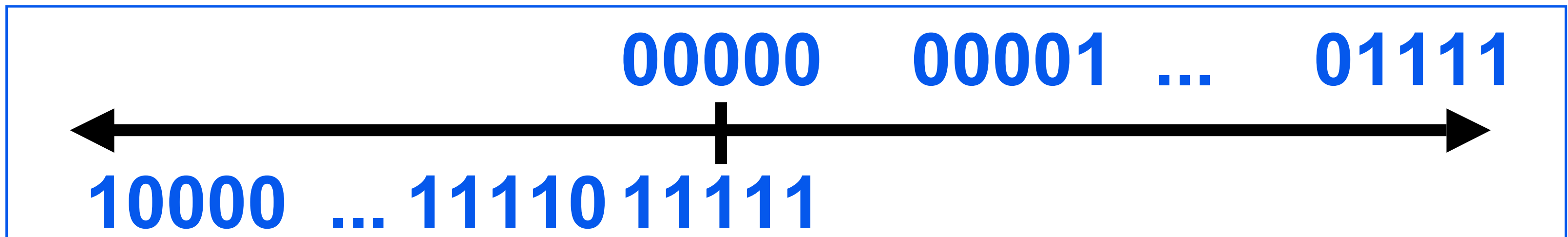
Their design: "field programmable nanowire interconnect (FPNI)" uses a technique that "will enable chip makers to pack eight times as many transistors as is currently possible on a standard 45nm field programmable gate array (FPGA) chip."



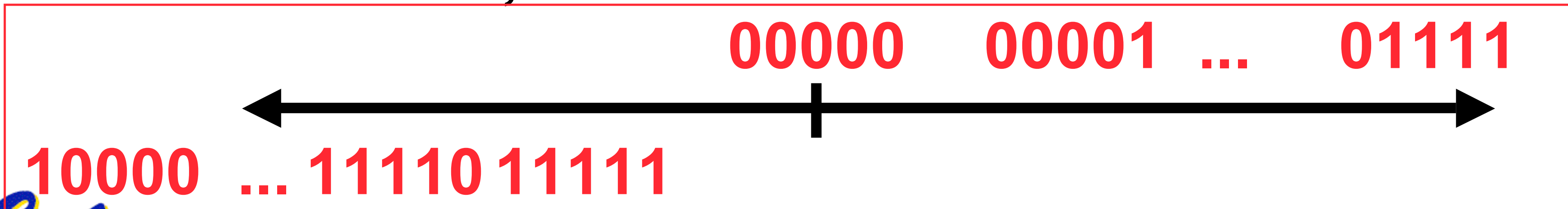
hardware.slashdot.org/hardware/07/01/17/1333232.shtml

Number review...

- We represent "things" in computers as particular bit patterns: N bits $\Rightarrow 2^N$
- Decimal for human calculations, binary for computers, hex to write binary more easily
- 1's complement - mostly abandoned



- 2's complement universal in computing: cannot avoid, so learn



- Overflow: numbers ∞; computers finite,errors!

Two's Complement shortcut: Negation

*Check out `www.cs.berkeley.edu/~dsw/twos_complement.html`

- Change every 0 to 1 and 1 to 0 (invert or complement), then add 1 to the result
- Proof*: Sum of number and its (one's) complement must be $111...111_{two}$
  - However, $111...111_{two} = -1_{ten}$
  - Let x' ⇒ one's complement representation of x
  - Then x + x' = -1 ⇒ x + x' + 1 = 0 ⇒ **-x = x' + 1**
- Example: -3 to +3 to -3

x : $1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1101_{two}$
x': $0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_{two}$
+1: $0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0011_{two}$
()': $1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1100_{two}$
+1: $1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1101_{two}$

You should be able to do this in your head…

)7 © UCB

Two's comp. shortcut: Sign extension

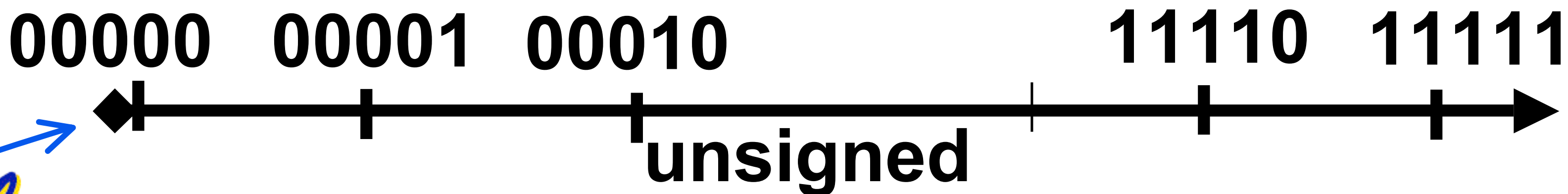
- **Convert 2's complement number rep. using n bits to more than n bits**
- **Simply replicate the most significant bit (sign bit) of smaller to fill new bits**
  - 2's comp. positive number has infinite 0s
  - 2's comp. negative number has infinite 1s
  - Binary representation hides leading bits; sign extension restores some of them
  - 16-bit -4_{ten} to 32-bit:

$$1111\ 1111\ 1111\ 1100_{two}$$

$$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1100_{two}$$

What if too big?

- Binary bit patterns above are simply **representatives** of numbers. Strictly speaking they are called "numerals".
- Numbers really have an ∞ number of digits
  - with almost all being same (00…0 or 11…1) except for a few of the rightmost digits
  - Just don't normally show leading digits
- If result of add (or -, *, /) cannot be represented by these rightmost HW bits, **overflow** is said to have occurred.



Garcia, Spring 2007 © UCB

Peer Instruction Question

X = 1111 1111 1111 1111 1111 1111 1111 1100_{two}

Y = 0011 1011 1001 1010 1000 1010 0000 0000_{two}

A. X > Y (if signed)

B. X > Y (if unsigned)

C. An encoding for Babylonians could have 2^N non-negative numbers w/N bits!

	ABC
0:	FFF
1:	FFT
2:	FTF
3:	FTT
4:	TFF
5:	TFT
6:	TTF
7:	TTT

Garcia, Spring 2007 © UCB

Introduction to C

Garcia, Spring 2007 © UCB

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://en.wikipedia.org/wiki/C99`

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

- Highlights
  - Declarations anywhere, 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)
  - `restrict` keyword for optimizations (#30)

Garcia, Spring 2007 © UCB

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 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

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

Garcia, Spring 2007 © UCB

C vs. Java™ Overview (1/2)

Java	C
• Object-oriented (OOP)	• No built-in object abstraction. Data separate from methods.
• “Methods”	• “Functions”
• Class libraries of data structures	• C libraries are lower-level
• Automatic memory management	• Manual memory management
	• Pointers

C vs. Java™ Overview (2/2)

Java

- High memory overhead from class libraries
- Relatively Slow
- Arrays initialize to zero
- Syntax:

```
/* comment */
// comment
System.out.print
```

C

- Low memory overhead
- Relatively Fast
- Arrays initialize to garbage
- Syntax: *

```
/* comment */
// comment
printf
```

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

Garcia, Spring 2007 © UCB

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

*C99 overcomes these limitations

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`

Garcia, Spring 2007 © UCB

C Syntax: `main`

- To get the main function to accept arguments, use this:

```
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).
    - Example: `unix% sort myFile`
  - `argv` is a pointer to an array containing the arguments as strings (more on pointers later).

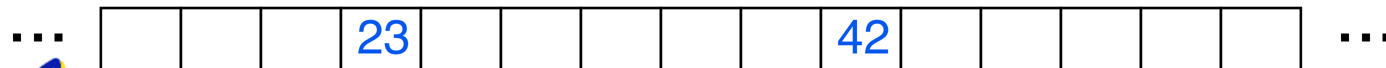
Garcia, Spring 2007 © UCB

Administrivia

- **Upcoming lectures**
  - C pointers and arrays in detail
- **HW**
  - HW0 due in discussion next week
  - HW1 due next Wed @ 23:59 PST
  - HW2 due following Wed @ 23:59 PST
- **Reading**
  - K&R Chapters 1-5 (lots, get started now!)
  - First quiz due Sun
- **Email me Ki - Me - Gi - … mnemonics!**
  - The subject should be “kibi mebi gibi acronym”

Address vs. Value

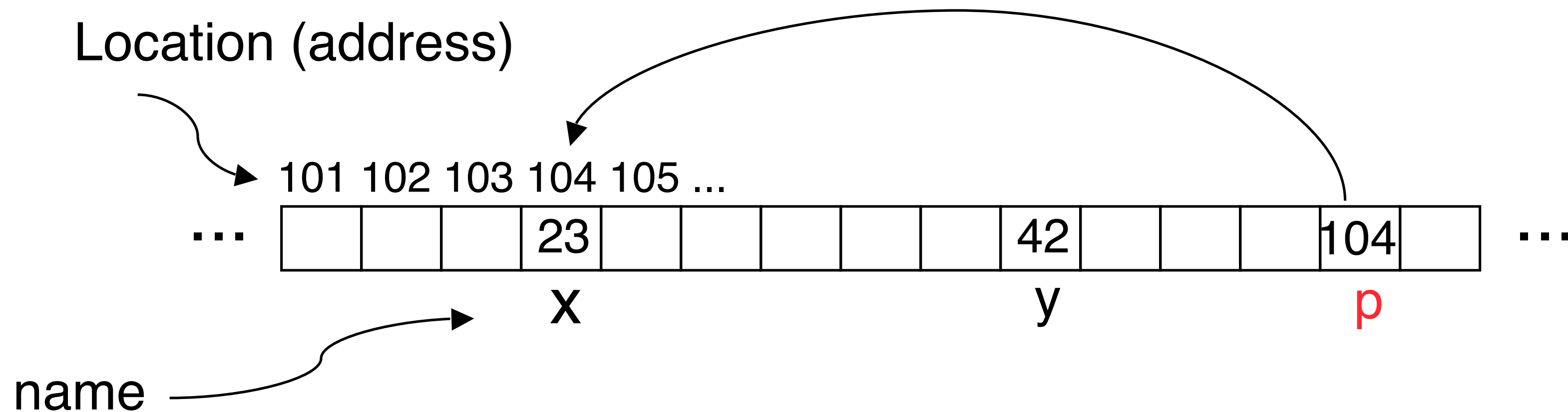
- Consider memory to be a single huge array:
  - Each cell of the array has an address associated with it.
  - Each cell also stores some value.
  - Do you think they use signed or unsigned numbers? Negative address?!
- Don't confuse the address referring to a memory location with the value stored in that location.



Garcia, Spring 2007 © UCB

Pointers

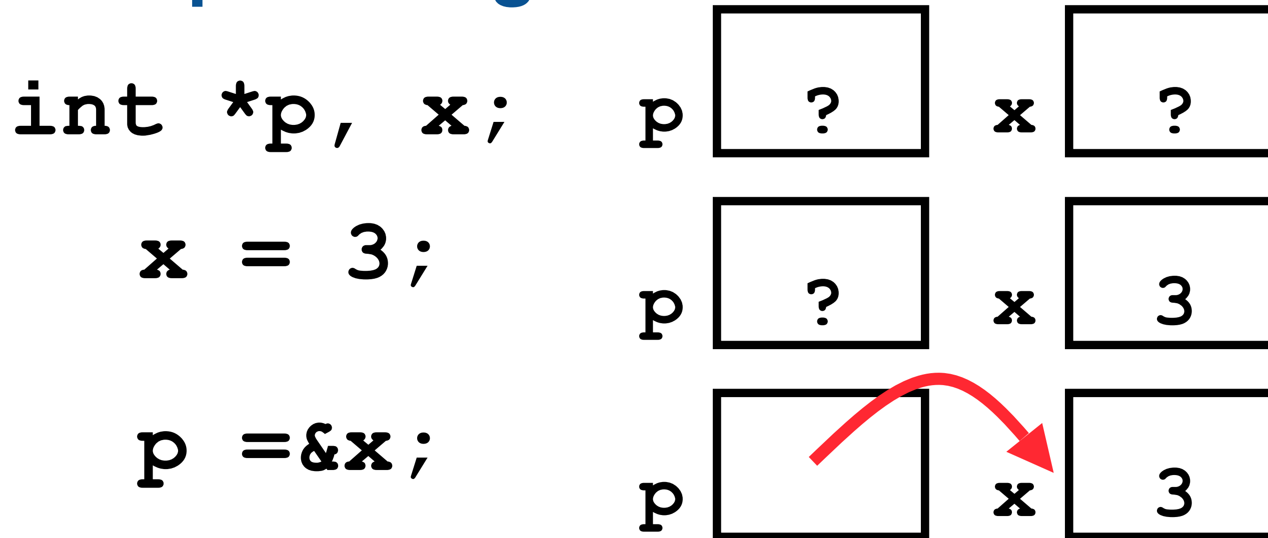
- An address refers to a particular memory location. In other words, it points to a memory location.
- **Pointer**: A variable that contains the address of a variable.



Garcia, Spring 2007 © UCB

Pointers

- **How to create a pointer:**
  - `&` operator: get address of a variable



Note the "*" gets used 2 different ways in this example. In the declaration to indicate that `p` is going to be a pointer, and in the `printf` to get the value pointed to by `p`.

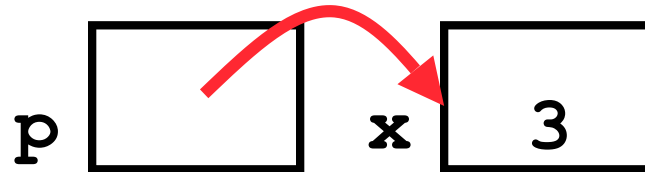
- **How get a value pointed to?**
  - `*` "dereference operator": get value pointed to

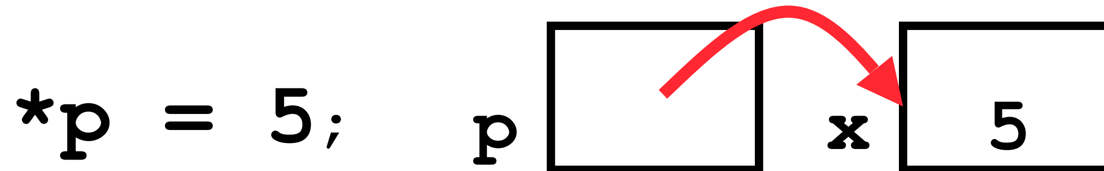
```
printf("p points to %d\n",*p);
```

Garcia, Spring 2007 © UCB

Pointers

- How to change a variable pointed to?
  - Use dereference * operator on left of =





Garcia, Spring 2007 © UCB

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

Garcia, Spring 2007 © UCB

Pointers and Parameter Passing

- How to get a function to change a value?

```
void addOne (int *p) {
  *p = *p + 1;
}

int y = 3;

addOne(&y);
```

`y` is now = 4

Garcia, Spring 2007 © UCB

Pointers

- 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!

Peer Instruction Question

```
void main(); {
  int *p, x=5, y; // init
  y = *(p = &x) + 10;
  int z;
  flip-sign(p);
  printf("x=%d,y=%d,p=%d\n",x,y,p);
}
flip-sign(int *n){*n = -(*n)}
```

#Errors
0
1
2
3
4
5
6
7
8
9

How many syntax/logic errors?

Garcia, Spring 2007 © UCB

Peer Instruction Answer

```
void main(); {
  int *p, x=5, y; // init
  y = *(p = &x) + 10;
  int z;
  flip-sign(p);
  printf("x=%d,y=%d,p=%d\n",x,y,*p);
}
flip-sign(int *n){*n = -(*n);}
```

#Errors
0
1
2
3
4
5
6
7
8
9

How many syntax/logic errors? I get 5.
(signed printing of pointer illogical)

Garcia, Spring 2007 © UCB

And in conclusion…

- All declarations go at the beginning of each function.
- Only 0 and NULL evaluate to FALSE.
- 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

Garcia, Spring 2007 © UCB