


inst.eecs.berkeley.edu/~cs61c  
**CS61C : Machine Structures**


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

2010-01-25

Lecturer SOE Dan Garcia  
[www.cs.berkeley.edu/~ddgarcia](http://www.cs.berkeley.edu/~ddgarcia)



MIPS Supercomputer ⇒  
 China's next supercomputer (the Dawning 6000) will be built using the Loongson (MIPS) processor and run Linux. Currently, the top 500 supercomputers are mostly x86 chips. You'll learn MIPS in CS61C!



[www.technologyreview.com/computing/24374/](http://www.technologyreview.com/computing/24374/)

CS61C L03 Introduction to C (pt 1) (1) Garcia, Spring 2010 © UCB

**And in review...** META: We often make design decisions to make HW simple

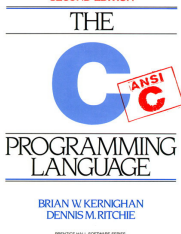
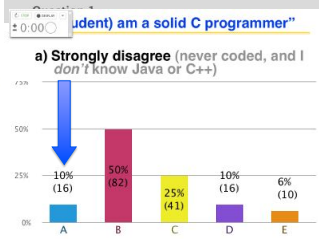
- We represent “things” in computers as particular bit patterns:  $N$  bits  $\Rightarrow 2^N$  things
- These 5 integer encodings have different benefits; 1s complement and sign/mag have most problems.
- unsigned (C99's `uintN_t`):  
 $00000 \quad 00001 \quad \dots \quad 01111 \quad 10000 \quad \dots \quad 11111$
- 2's complement (C99's `intN_t`) universal, learn!  
 $10000 \quad \dots \quad 11110 \quad 11111$
- Overflow: numbers  $\infty$ ; computers finite, errors!

META: Ain't no free lunch

CS61C L03 Introduction to C (pt 1) (2) Garcia, Spring 2010 © UCB

**Introduction to C**

- Officially, “some” C experience is required before CS61C
  - 2010Sp an experiment!
- What to do if you're in that 10%?
  - Start early, ask questions as soon as they come up!

CS61C L03 Introduction to C (pt 1) (3) Garcia, Spring 2010 © 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](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)

CS61C L03 Introduction to C (pt 1) (4) Garcia, Spring 2010 © 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”
    - <http://oreilly.com/catalog/javanut/excerpt/>
  - Brian Harvey's course notes
    - On CS61C class website

CS61C L03 Introduction to C (pt 1) (5) Garcia, Spring 2010 © UCB

**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)

CS61C L03 Introduction to C (pt 1) (6) Garcia, Spring 2010 © UCB



## Pointers

### • How to create a pointer:

& operator: get address of a variable

```
int *p, x;  p [?] x [?]
```

*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.*

```
x = 3;      p [?] x [3]
```

```
p = &x;    p [ ] x [3]
```

### • How get a value pointed to?

\* "dereference operator": get value pointed to

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



## Pointers

### • How to change a variable pointed to?

• Use dereference \* operator on left of =

```
p [ ] x [3]
```

```
*p = 5;  p [ ] x [5]
```



## 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 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**



## 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) + 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



## And in conclusion...

- All declarations go at the beginning of each function except if you use C99.
- 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



## 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        | • <b>Manual</b> memory management                             |
|                                      | • <b>Pointers</b>   |



## C vs. Java™ Overview (2/2)

- | Java  | C   |
|---|---|
| • <b>High</b> memory overhead from class libraries                        | • <b>Low</b> memory overhead                                      |
| • <b>Relatively Slow</b>  | • <b>Relatively Fast</b>  |
| • Arrays initialize to <b>zero</b>  | • Arrays initialize to <b>garbage</b>                             |
| • Syntax:<br><pre>/* comment */<br/>// comment<br/>System.out.print</pre> | • Syntax: *<br><pre>/* comment */<br/>// comment<br/>printf</pre> |

\* 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

