

CS61C : Machine Structures

Lecture 2 – Introduction To C

2004-06-22

Kurt Meinz



inst.eecs.berkeley.edu/~cs61c

CS 61C L2 Introduction to C (pt 1) (1)

K. Meinz, Summer 2004 © UCB

Review

- Two's Complement



CS 61C L2 Introduction to C (pt 1) (2)

K. Meinz, Summer 2004 © UCB

Another Attempt ...

- Gedanken: Decimal Car Odometer
00003 → 00002 → 00001 → 00000 → 99999 → 99998
- Binary Odometer:
00011 → 00010 → 00001 → 00000 → 11111 → 11110
- With no obvious better alternative, pick representation that makes the math simple!
 - 99999_{ten} == -1_{ten}
 - 11111_{two} == -1_{ten} 11110_{two} == -2_{ten}
- This representation is **Two's Complement**



CS 61C L2 Introduction to C (pt 1) (3)

K. Meinz, Summer 2004 © UCB

2's Complement Properties

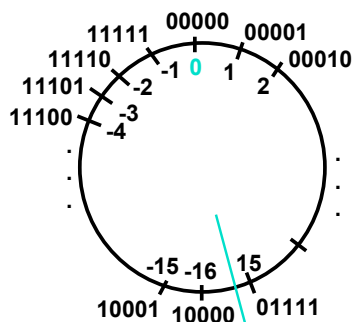
- As with sign and magnitude, leading 0s ⇒ positive, leading 1s ⇒ negative
 - 00000...xxx is ≥ 0, 11111...xxx is < 0
 - except 1...1111 is -1, not -0 (as in sign & mag.)
- Only 1 Zero!



CS 61C L2 Introduction to C (pt 1) (4)

K. Meinz, Summer 2004 © UCB

2's Complement Number "line": N = 5



- 2^{N-1} non-negatives
- 2^{N-1} negatives
- one zero
- how many positives?



CS 61C L2 Introduction to C (pt 1) (5)

K. Meinz, Summer 2004 © UCB

Two's Complement for N=32

0000 ... 0000 0000 0000 0000	=	0
0000 ... 0000 0000 0000 0001	=	1 _{ten}
0000 ... 0000 0000 0000 0010	=	2 _{ten}
...		
0111 ... 1111 1111 1111 1101	=	2,147,483,645 _{ten}
0111 ... 1111 1111 1111 1110	=	2,147,483,646 _{ten}
0111 ... 1111 1111 1111 1111	=	2,147,483,647 _{ten}
1000 ... 0000 0000 0000 0000	=	-2,147,483,648 _{ten}
1000 ... 0000 0000 0000 0001	=	-2,147,483,647 _{ten}
1000 ... 0000 0000 0000 0010	=	-2,147,483,646 _{ten}
...		
1111 ... 1111 1111 1111 1101	=	-3 _{ten}
1111 ... 1111 1111 1111 1110	=	-2 _{ten}
1111 ... 1111 1111 1111 1111	=	-1 _{ten}

- One zero; 1st bit called **sign bit**
- 1 "extra" negative: no positive 2,147,483,648_{ten}



CS 61C L2 Introduction to C (pt 1) (6)

K. Meinz, Summer 2004 © UCB

Two's Complement Formula

- Can represent positive **and negative** numbers in terms of the bit value times a power of 2:

$$d_{31} \times (2^{31}) + d_{30} \times 2^{30} + \dots + d_2 \times 2^2 + d_1 \times 2^1 + d_0 \times 2^0$$

- Example: 1101_{two}
 - $= 1x(-2^3) + 1x2^2 + 0x2^1 + 1x2^0$
 - $= -2^3 + 2^2 + 0 + 2^0$
 - $= -8 + 4 + 0 + 1$
 - $= -8 + 5$
 - $= -3_{\text{ten}}$

Cal

CS 61C L2 Introduction to C (pt 1) (7)

K. Meinz, Summer 2004 © UCB

Two's Complement shortcut: Negation

- 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 $111\dots111_{\text{two}}$
However, $111\dots111_{\text{two}} = -1_{\text{ten}}$
Let $x' \Rightarrow$ one's complement representation of x
Then $x + x' = -1 \Rightarrow x + x' + 1 = 0 \Rightarrow x' + 1 = -x$

- Example: -3 to +3 to -3

~~$x: 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1101_{\text{two}}$~~
 ~~$x': 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_{\text{two}}$~~
 ~~$+1: 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0011_{\text{two}}$~~
 ~~$(-3)_{\text{ten}}$~~
 ~~$+1: 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1101_{\text{two}}$~~
 ~~$+1: 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1101_{\text{two}}$~~

Cal

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

CS 61C L2 Introduction to C (pt 1) (8)

K. Meinz, Summer 2004 © 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:

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

Cal

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

CS 61C L2 Introduction to C (pt 1) (9)

K. Meinz, Summer 2004 © UCB

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 ($00\dots0$ or $11\dots1$) 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.

$00000\ 00001\ 00010\ \dots\ 11110\ 11111$
 unsigned

Cal

CS 61C L2 Introduction to C (pt 1) (10)

K. Meinz, Summer 2004 © UCB

Number Summary

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

$00000\ 00001\ \dots\ 01111$
 $10000\ \dots\ 11110\ 11111$

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

$00000\ 00001\ \dots\ 01111$
 $10000\ \dots\ 11110\ 11111$

Cal

Overflow: numbers ∞ ; computers finite, errors!

CS 61C L2 Introduction to C (pt 1) (11)

K. Meinz, Summer 2004 © UCB

Preview: Signed vs. Unsigned Variables

- Java just declares integers `int`
 - Uses two's complement
- C has declaration `int` also
 - Declares variable as a signed integer
 - Uses two's complement
- Also, C declaration `unsigned int`
 - Declares a unsigned integer
 - Treats 32-bit number as unsigned integer, so most significant bit is **part of the number**, not a sign bit

Cal

CS 61C L2 Introduction to C (pt 1) (12)


K. Meinz, Summer 2004 © UCB

Big Idea

- Next Topic: Numbers can Be Anything!

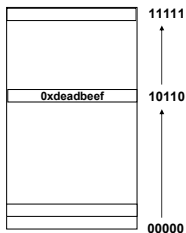


BIG IDEA: Bits can represent anything!!

- **REMEMBER:** N digits in base B $\Rightarrow B^N$ values
 - For binary in particular: N bits $\rightarrow 2^N$ values
- Characters?
 - 26 letters $\Rightarrow 5$ bits ($2^5 = 32$)
 - upper/lower case + punctuation $\Rightarrow 7$ bits (in 8) ("ASCII")
 - standard code to cover all the world languages $\Rightarrow 16$ bits ("Unicode") 
- Logical values?
 - 0 \Rightarrow False, 1 \Rightarrow True
- colors ? Ex: Red (00) Green (01) Blue (11)
- locations / addresses? commands?



Example: Numbers represented in memory



- Memory is a place to store bits
- A *word* is a fixed number of bits (eg, 32) at an address
- *Addresses* are naturally represented as unsigned numbers in C



New Topic

- Course Administration



Administrivia – Read the course handout

- Just about everything is in the course info handout.
 - Sec 2: Course is difficult over summer
 - Be prepared to commit 12 hrs/week in class and 20 hrs/week outside of class!
 - Sec 3: Textbooks: COD, K&R
 - Sec 4: Labs and Discussion
 - Go to your own this week
 - Log into your account!
 - Hand in survey/statement to TA.



Administrivia – Read the course handout

- Sec 10: Assignments:
 - 1) Online Pre-lecture Quizzes:
 - Mandatory (Effort)
 - About 20 over the semester
 - Wednesday's is up now (or very soon)
 - In general, will be up at least two days in advance
 - No late quizzes; no partners
 - 2) Labs
 - Mandatory (Correctness)
 - 2 per week
 - "Checked-off" by TA during section
 - » TA will ask questions – you answer them!
 - No late labs; "no partners"



Administrivia – Read the course handout

• Sec 10: Assignments:

- 3) Homeworks

- Mandatory Online Turnin
 - » Graded once on correctness
 - » Chance to get back points
- 2 Per week
- Both due on Sunday 8:00pm after assigned
- No late homeworks; no partners

- 4) Projects

- Mandatory (Correctness) Online Turnin
 - » Probably graded face-to-face.
- 1 Project roughly every 4 weeks
- No late projects; "no partners"



Administrivia – Read the course handout

• Sec 11: Grading:

20 reading quizzes	@	0.5 points each	=	10 pts
15 labs	@	2 points each	=	30 pts
15 homeworks	@	4 points each	=	60 pts
4 projects	@	12.5 points each	=	50 pts
3 midterms	@	30 points each	=	90 pts
1 final			=	60 pts
--				---
58 assignments				300 pts

- Midterms/Final: On Fridays, 3 hours, cover two weeks at a time



Administrivia – Read the course handout

• Sec 11: Grading

A+ 280-300	A 270-279	A- 260-269
B+ 250-259	B 240-249	B- 230-239
C+ 220-229	C 210-219	C- 200-209
D+ 190-199	D 180-189	D- 170-179

- I may adjust it in your favor



Administrivia – Read the course handout

• Sec 12: Assignment Grading

- Labs: checkoff by TA
- Quizzes: submit via www
- HW:
 - Submit via 'submit' program
 - Graded on correctness
 - If it appears that you put in honest effort, but got less than 90/100
 - » Sign up for face-to-face session with grader
 - » Look up solutions, understand them, figure out what you did wrong
 - » Convince grader that you now understand what you got wrong
 - » Grader will give you up to 90/100 points back!



Administrivia – Read the course handout

• Sec 13: Cheating

- Don't do it.
- Detection:
 - Automated programs,
 - Staff suspicions
 - Understanding of material
- Penalty:
 - If you confess → zero on assignment, "faculty disposition" to OSC (not noted in record)
 - If you don't → "Faculty referral" to OSC (noted in record if OSC finds against you)
- Please sign the "Statement on Cheating".



Big Idea

- Next Topic: Intro to C



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".



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.
- 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



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



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);
```

Cal

CS 61C L2 Introduction to C (pt 1) (37)

K. Meinz, Summer 2004 © UCB

Pointers

• How to change a variable pointed to?

• Use dereference * operator on left of =

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

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

Cal

CS 61C L2 Introduction to C (pt 1) (38)

K. Meinz, Summer 2004 © UCB

Pointers and Parameter Passing

• Java and C pass a parameter "by value"

• procedure/function 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

Cal

CS 61C L2 Introduction to C (pt 1) (39)

K. Meinz, Summer 2004 © 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

Cal

CS 61C L2 Introduction to C (pt 1) (40)

K. Meinz, Summer 2004 © UCB

Pointers

• 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!

Cal

CS 61C L2 Introduction to C (pt 1) (41)

K. Meinz, Summer 2004 © 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

Cal

CS 61C L2 Introduction to C (pt 1) (42)

K. Meinz, Summer 2004 © UCB