inst.eecs.berkeley.edu/~cs61c

CS61C : Machine Structures

Lecture 3 – Introduction to the C Programming Language

2006-08-31

Lecturer SOE Dan Garcia

www.cs.berkeley.edu/~ddgarcia

Ant jaw power ⇒ Cal researchers found the trap-jaw ant has the "fastest self-powered predatory strike in the animal kingdom". Must-see ant videos!!

www.berkeley.edu/news/media/releases/2006/08/21_ant.shtml

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:

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

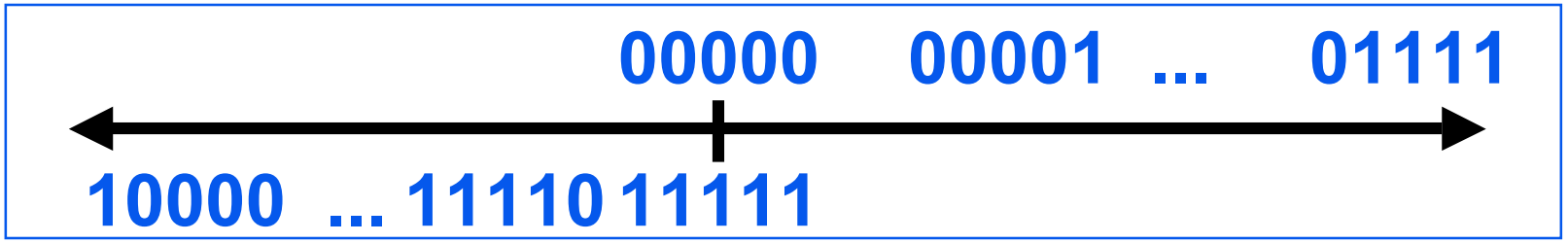
$$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1100_{two}$$

What if too big?

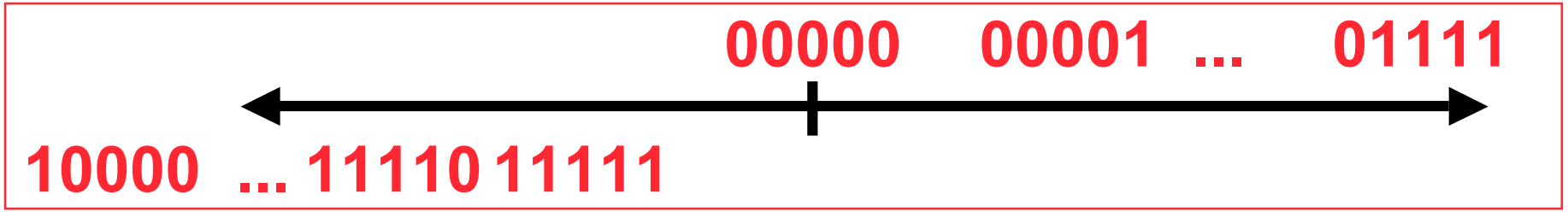
- Binary bit patterns above are simply representatives of numbers. Strictly speaking they are called "numerals".
- Numbers really have an ∞ number of digits
  - with almost all being same (00…0 or 11…1) except for a few of the rightmost digits
  - Just don't normally show leading digits
- If result of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

Review

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

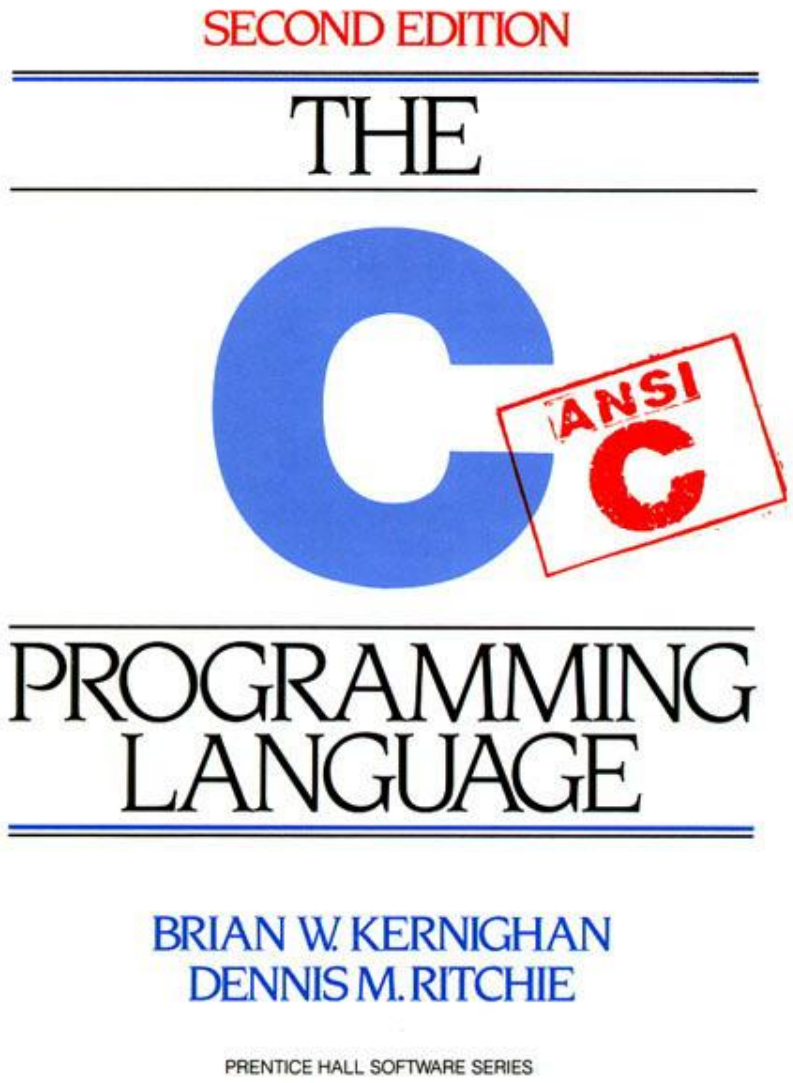


- 2's complement universal in computing: cannot avoid, so learn



- Overflow: numbers ∞; computers finite, errors!

Introduction to C



Has there been an update to ANSI C?

- Yes! It's called the "C99" or "C9x" std
  - Thanks to Jason Spence for the tip
- References

  `http://en.wikipedia.org/wiki/Standard_C_library`

  `http://home.tiscalinet.ch/t_wolf/tw/c/c9x_changes.html`
- Highlights
  - `<inttypes.h>`: convert integer types (#38)
  - `<stdbool.h>` for boolean logic def's (#35)
  - `restrict` keyword for optimizations (#30)
  - Named initializers (#17) for aggregate objs

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

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 */

printf
```

*Newer C compilers allow Java style comments as well!

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++)`

*C compilers now allow this in the case of "for" loops.

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`

C Syntax: `main`

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

```
int main (int argc, char *argv[])
```

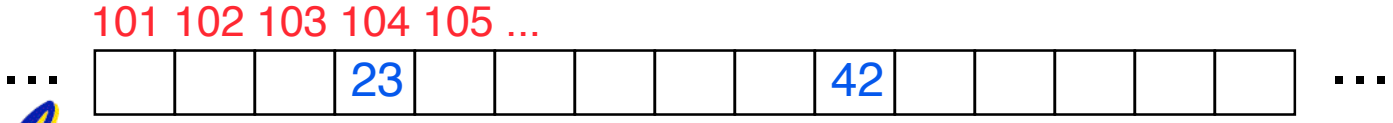
- What does this mean?
  - `argc` will contain the number of strings on the command line (the executable counts as one, plus one for each argument).
    - Example: `unix% sort myFile`
  - `argv` is a pointer to an array containing the arguments as strings (more on pointers later).

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
- CPS will start next wednesday
  - I've heard you can sell your CPS back to store
- Monday is a holiday, don't come here
- Email me Ki - Me - Gi - … mnemonics!

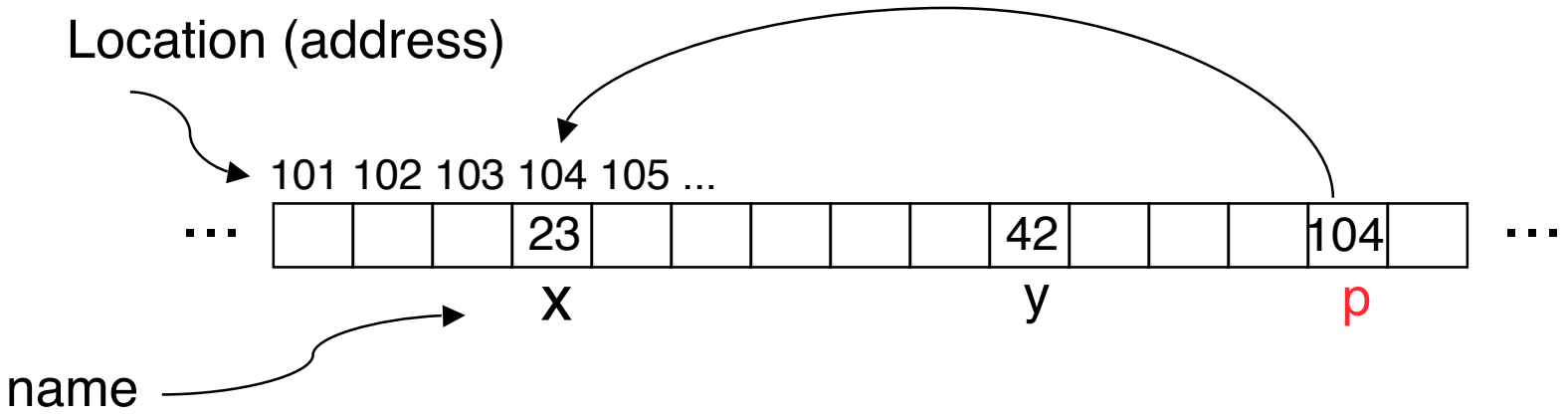
Address vs. Value

- 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

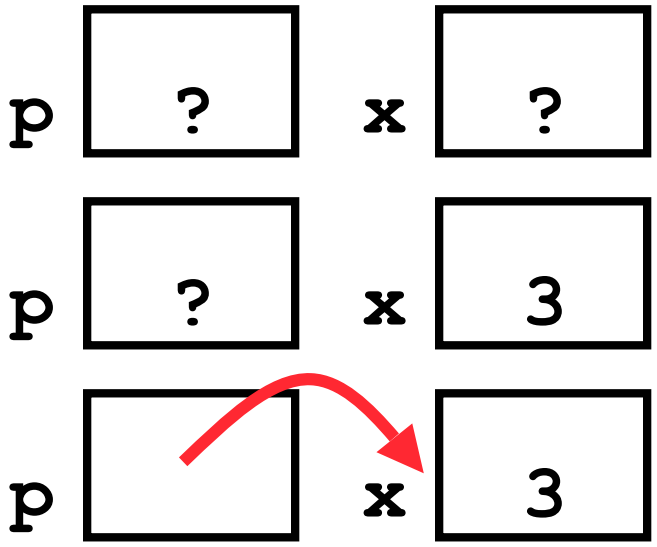
- **An address refers to a particular memory location. In other words, it points to a memory location.**
- **Pointer: A variable that contains the address of a variable.**



Pointers

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

```
int *p, x;
   x = 3;
   p =&x;
```



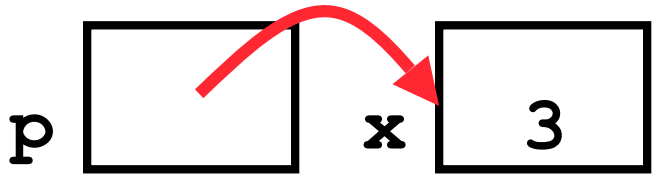
Note the "*" gets used 2 different ways in this example. In the declaration to indicate that `p` is going to be a pointer, and in the `printf` to get the value pointed to by `p`.

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

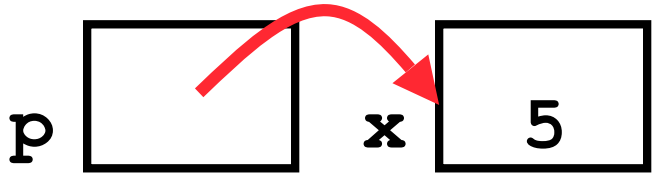
```
printf("p points to %d\n",*p);
```

Pointers

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




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

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

How many errors?

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
1
2
3
4
5
6
7
8
9
(1) 0

How many errors? I get 7.

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

Administrivia : Lab priority

Rank order of seating priority

1. 61c registered for that section
2. 61c registered for another section
3. 61c waitlisted for that section
4. 61c waitlisted for another section
5. Concurrent enrollment

If low on list for busy section, think of moving to the early or late sections (usually more empty seats)

Kilo, Mega, Giga, Tera, Peta, Exa, Zetta, Yotta

- Kilted Meghans giggle terribly petting exalted zellous yodas [CL]
- Kissing me gives terrible peeps exactly zero, yo! [CL]
- Killer Megan gives Terrible Peter's excellent zebra yoghurt [YC]
- “Kiss me”, giant Terrible Peter exclaimed zealously, yo [YC]
- Kind Merchants Give Texan People Extra Zesty Yogurt [AW]
- Kittens' Meows Give to Terrific Peals of Extraordinarily Zealous Yowls [AW]
- Killer Mercenary Giants Temporarily Pester Exercising Zebras in Yorkshire [AW]
- Kiss me girl, terrible people examine zebras, yo. [JD]
- Kiss me, given ten pens extracted zen-like yo [AG]
- KIssing ME GIrl, TElls of my PEnchant for EXtra ZEsty Yoghurt [TM]
- Kissing me gingerly, Ted Peterson exclaimed, "Zesty, yo!” [DH]
- Kiss me girl teach petty exasperations zestful yodeling [AR]
- Kind Megan Gibson teaches people extremely zestful yoga [AC]
- Kissing mediocre girls/gimmicks teaches/tells people to expect zero/zest from you [MT]
- Kiss me, giant tease, people excuse zealous young [CR]
- Kicking mean girls and teasing pedestrians excite zealous youngsters [MH]
- Killin' me! Giant teacher's pet exaggerates zealously yo [KN]
- Kind Merlin gives tense people exceptional zebra yogurt [KL]
- Kinky metaphysics gibberish teaches people exquisite Zen yodeling [JC]
- Kingly men giving tedious penance exhibit zealous yowls [MH]
- Kinky mean girls terrorizing petty ex-boyfriends zeroing-on you [HC]
- Kind Merlin Gives Ten People Extremeley Zealous Yodas [RC]
- Kiss Me Goat Te Procure Extra Zloties, Yo [RG]

Kilo, Mega, Giga, Tera, Peta, Exa, Zetta, Yotta

1. King Mega gives Teddy pets, except zebra, yo [HL]
2. Kim's melodious giddiness terrifies people, excepting zealous yodelers [DW]
3. Kirby Messed Gigglypuff Terribly, (then) Perfectly Exterminated Zelda and Yoshi [CB]
4. Killed meat gives teeth peace except zebra yogurt [CR]
5. Kind Men Give Tense People Extra Zeal (for) Yoga [VK/DG]
6. Killing melee gives terror; peace exhibits Zen yoga [CR]
7. Killing messengers gives terrible people exactly zero, yo [CL]
8. Kindergarten means giving teachers perfect examples (of) zeal (&) youth
9. Kissing mediocre girls teaches people (to) expect zero (from) you [MT]
10. Kinky Mean Girls Teach Penis-Extending Zen Yoga [AW]
11. Kissing Mel Gibson, Teddy Pendergrass exclaimed, "Zesty, yo!" [DH / AC/DG]

CB

Administrivia : You have a question?

- Do not email Dan (& expect response)
  - Hundreds of emails in inbox
  - Email doesn't scale to classes with 120+ students!
- Tips on getting an answer to your question:
  - Ask a classmate
  - Ask Dan after or before lecture
  - The newsgroup, ucb.class.cs61c
    - Read it : Has your Q been answered already?
    - If not, ask it and check back
  - Ask TA in section, lab or OH
  - Ask Dan in OH
  - Ask Dan in lecture (if relevant to lecture)
  - Send your TA email
  - Send your Head TAs email
  - Send Dan email