CS 61C:
Great Ideas in Computer Architecture
Lecture 2: *Introduction to C, Part I*

Instructors:
Vladimir Stojanovic & Nicholas Weaver

http://inst.eecs.berkeley.edu/~cs61c/

Berkeley EECS
ELECTRICAL ENGINEERING & COMPUTER SCIENCES

Agenda

- Everything is a Number
- Computer Organization
- Compile vs. Interpret

Key Concepts

- Inside computers, everything is a number
- But numbers usually stored with a fixed size
  - 8-bit bytes, 16-bit half words, 32-bit words, 64-bit double words, …
- Integer and floating-point operations can lead to results too big/small to store within their representations: *overflow*/*underflow*

Number Representation

- Value of i-th digit is $d \times Base^i$ where i starts at 0 and increases from right to left:
- $123_{10} = 1_{10} \text{ x } 10_{10}{}^2 + 2_{10} \text{ x } 10_{10}{}^1 + 3_{10} \text{ x } 10_{10}{}^0$

  $= 1\text{x}100_{10} + 2\text{x}10_{10} + 3\text{x}1_{10}$

  $= 100_{10} + 20_{10} + 3_{10}$

  $= 123_{10}$
- Binary (Base 2), Hexadecimal (Base 16), Decimal (Base 10) different ways to represent an integer
  - We'll use 1_{two}, 5_{ten}, 10_{hex} to be clearer (vs. 1_2, 4_8, 5_{10}, 10_{16})

Number Representation

- Hexadecimal digits: 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F
- $FFF_{hex} = 15_{ten} x\ 16_{ten}{}^{2} + 15_{ten} x\ 16_{ten}{}^{1} + 15_{ten} x\ 16_{ten}{}^{0}$
  $= 3840_{ten} + 240_{ten} + 15_{ten}$
  $= 4095_{ten}$
- $1111\ 1111\ 1111_{two} = FFF_{hex} = 4095_{ten}$
- May put blanks every group of binary, octal, or hexadecimal digits to make it easier to parse, like commas in decimal

Signed and Unsigned Integers

- C, C++, and Java have *signed integers*, e.g., 7, -255:

```
int x, y, z;
```

- C, C++ also have *unsigned integers*, which are used for addresses
- 32-bit word can represent 2^{32} binary numbers
- Unsigned integers in 32 bit word represent
  0 to 2^{32}-1 (4,294,967,295)

Unsigned Integers

$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_{two} = 0_{ten}$
$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_{two} = 1_{ten}$
$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_{two} = 2_{ten}$
... ...
$0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1101_{two} = 2{,}147{,}483{,}645_{ten}$
$0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_{two} = 2{,}147{,}483{,}646_{ten}$
$0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_{two} = 2{,}147{,}483{,}647_{ten}$
$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_{two} = 2{,}147{,}483{,}648_{ten}$
$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_{two} = 2{,}147{,}483{,}649_{ten}$
$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_{two} = 2{,}147{,}483{,}650_{ten}$
... ...
$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1101_{two} = 4{,}294{,}967{,}293_{ten}$
$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_{two} = 4{,}294{,}967{,}294_{ten}$
$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_{two} = 4{,}294{,}967{,}295_{ten}$

Signed Integers and Two's-Complement Representation

- Signed integers in C; want ½ numbers <0, want ½ numbers >0, and want one 0
- *Two's complement* treats 0 as positive, so 32-bit word represents 2^{32} integers from -2^{31} (–2,147,483,648) to $2^{31}-1$ (2,147,483,647)
  - Note: one negative number with no positive version
  - Book lists some other options, all of which are worse
  - Every computer uses two's complement today
- *Most-significant bit* (leftmost) is the *sign bit*, since 0 means positive (including 0), 1 means negative
  - Bit 31 is most significant, bit 0 is least significant

Two's-Complement Integers

Sign Bit

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

Ways to Make Two's Complement

- For N-bit word, complement to 2_{ten}^{N}
  - For 4 bit number $3_{ten}=0011_{two}$,

    two's complement (i.e. -3_{ten}) would be

    $16_{ten}-3_{ten}=13_{ten}$ or $10000_{two} - 0011_{two} = 1101_{two}$
- Here is an easier way:
  - Invert all bits and add 1
  - Computers actually do it like this, too

3_{ten}	0011_{two}
Bitwise complement	1100_{two}
	$+ \quad 1_{two}$
-3_{ten}	1101_{two}

Binary Addition Example

Carry

0010

3	0011
+2	0010
5	00101

Two's-Complement Examples

- Assume for simplicity 4 bit width, -8 to +7 represented

3	0011
+2	0010
5	0101

3	0011
+ (-2)	1110
1	1 0001

-3	1101
+ (-2)	1110
-5	1 1011

Carry into MSB = Carry Out MSB

Overflow when magnitude of result too big to fit into result representation

7	0111
+1	0001
-8	1000

Overflow!

-8	1000
+ (-1)	1111
+7	1 0111

Overflow!

Carry into MSB ≠ Carry Out MSB

Carry in = carry from less significant bits

Carry out = carry to more significant bits

Suppose we had a 5-bit word. What integers can be represented in two's complement?

- ☐ -32 to +31
- ☐ 0 to +31
- ☐ -16 to +15
- ☐ -15 to +16

Suppose we had a 5 bit word. What integers can be represented in two's complement?

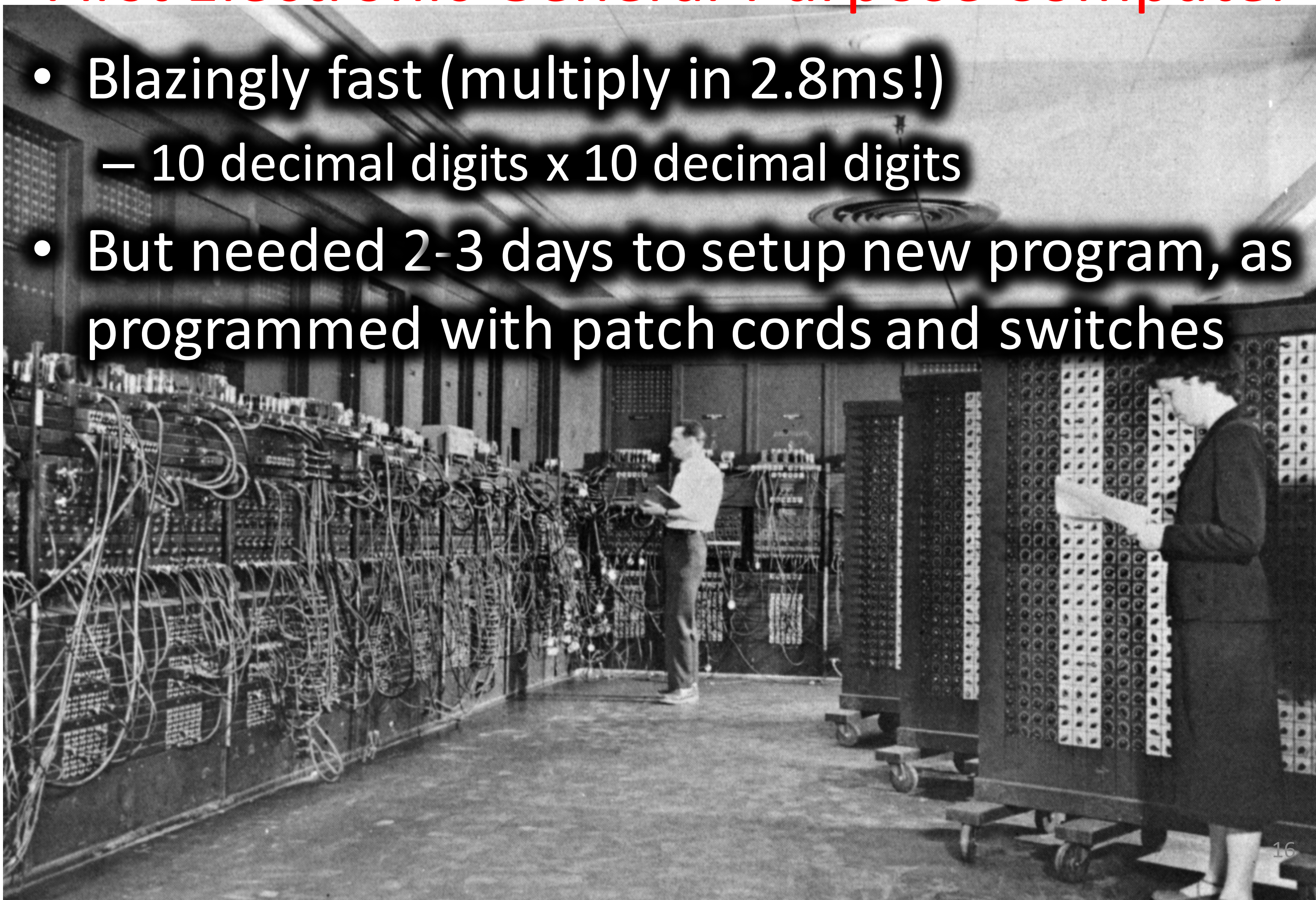
- ☐ -32 to +31
- ☐ 0 to +31
- ☐ -16 to +15
- ☐ -15 to +16

Agenda

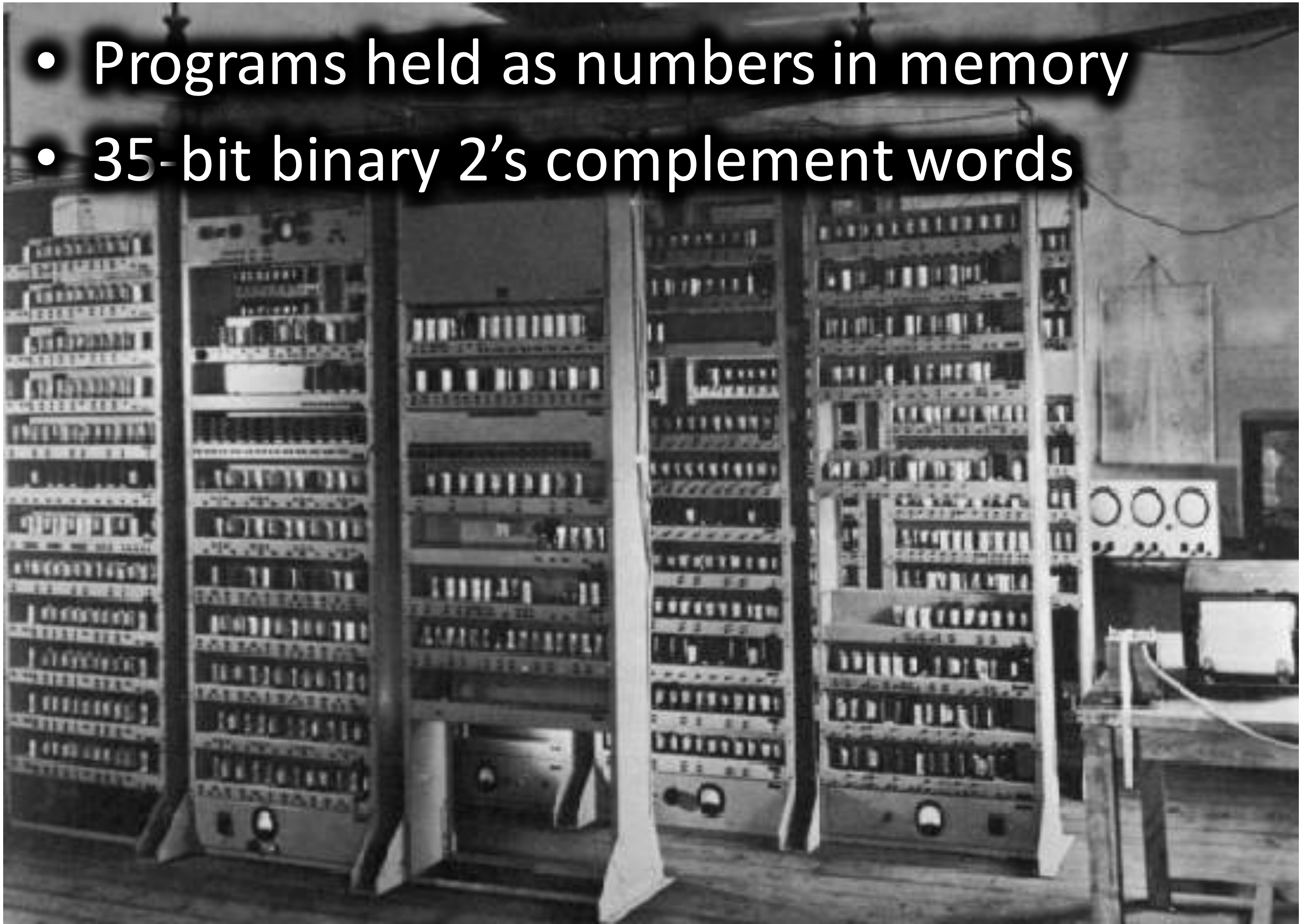
- Everything is a Number
- Computer Organization
- Compile vs. Interpret

ENIAC (U.Penn., 1946)
First Electronic General-Purpose Computer

- Blazingly fast (multiply in 2.8ms!)
  - 10 decimal digits x 10 decimal digits
- But needed 2-3 days to setup new program, as programmed with patch cords and switches

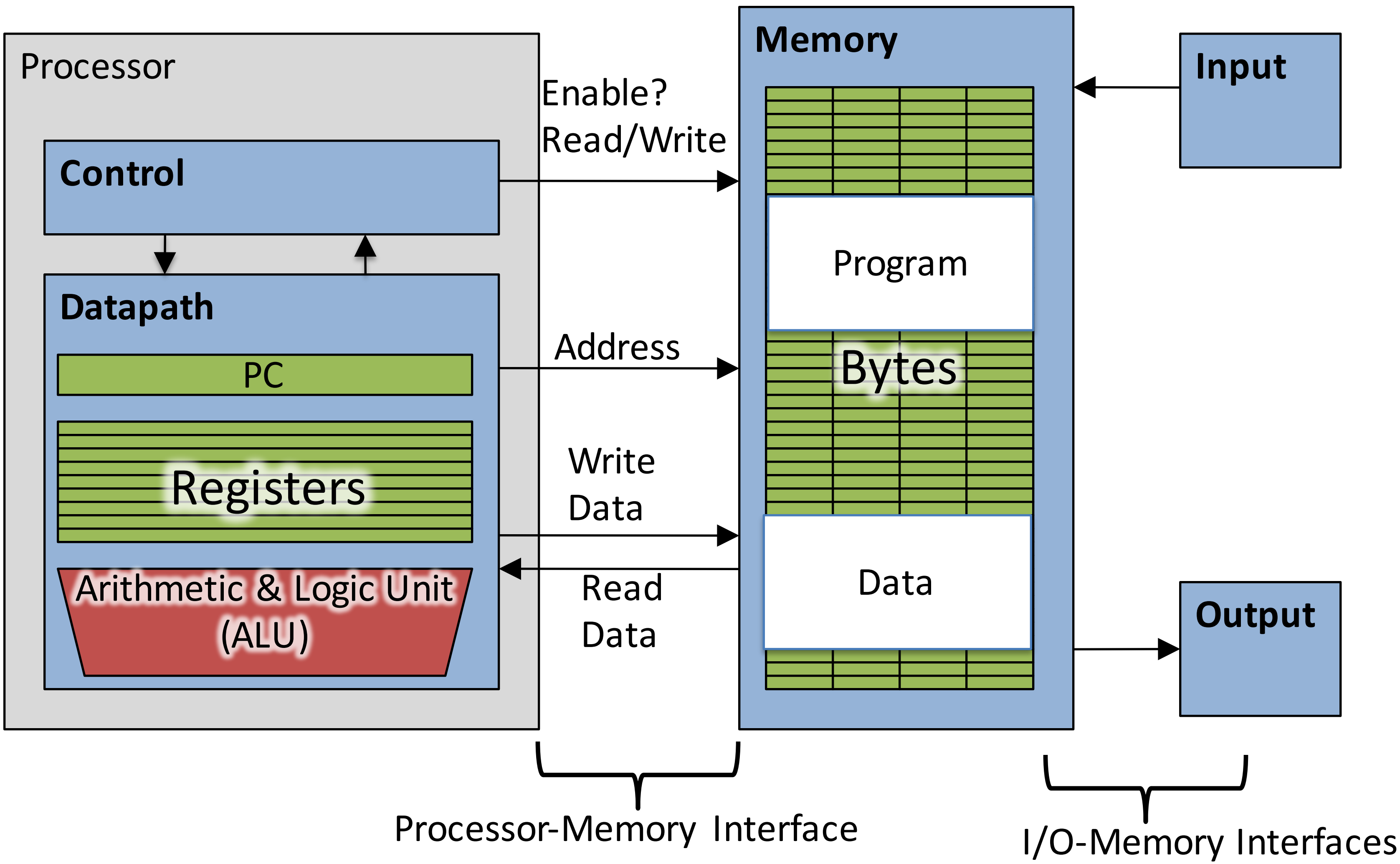


EDSAC (Cambridge, 1949)
First General Stored-Program Computer



- Programs held as numbers in memory
- 35-bit binary 2's complement words

Components of a Computer



Great Idea: Levels of Representation/Interpretation

High Level Language Program (e.g., C)

```
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
```

We are here!

Compiler

Assembly Language Program (e.g., MIPS)

```
lw    $t0, 0($2)
lw    $t1, 4($2)
sw    $t1, 0($2)
sw    $t0, 4($2)
```

Anything can be represented as a *number*, i.e., data or instructions

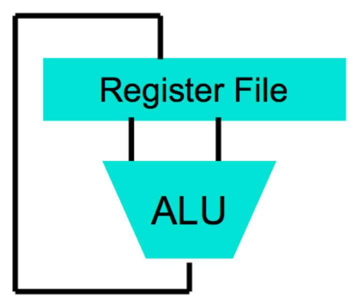
Assembler

Machine Language Program (MIPS)

```
0000 1001 1100 0110 1010 1111 0101 1000
1010 1111 0101 1000 0000 1001 1100 0110
1100 0110 1010 1111 0101 1000 0000 1001
0101 1000 0000 1001 1100 0110 1010 1111
```

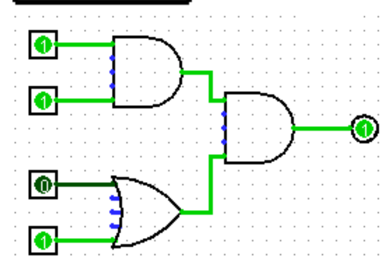
Machine Interpretation

Hardware Architecture Description (e.g., block diagrams)



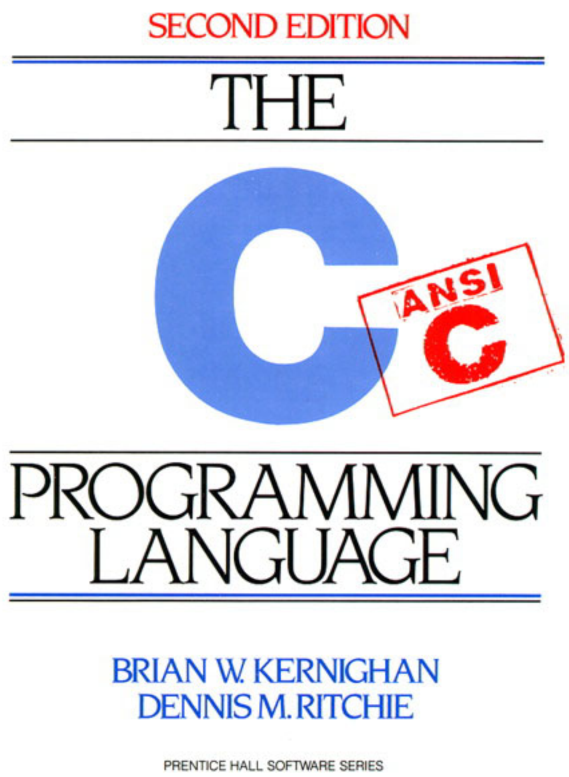
Architecture Implementation

Logic Circuit Description (Circuit Schematic Diagrams)



Introduction to C
"The Universal Assembly Language"

- "Some" experience is required before CS61C
  C++ or Java OK



- Class pre-req included classes teaching Java
- Python used in two labs
- C used for everything else

Language Poll!

Please raise hand for *first* one of following you can say yes to

- ☐ I have programmed in C, C++, C#, or Objective-C
- ☐ I have programmed in Java
- ☐ I have programmed in FORTRAN, Cobol, Algol-68, Ada, Pascal, or Basic
- ☐ None of the above

Intro to C

- *C is not a "very high-level" language, nor a "big" one, and is not specialized to any particular area of application. But its absence of restrictions and its generality make it more convenient and effective for many tasks than supposedly more powerful languages.*

  – Kernighan and Ritchie

- Enabled first operating system not written in assembly language: *UNIX* - A portable OS!

Intro to C

- Why C?: *we can write programs that allow us to exploit underlying features of the architecture – memory management, special instructions, parallelism*
- C and derivatives (C++/Obj-C/C#) still one of the most popular application programming languages after >40 years!

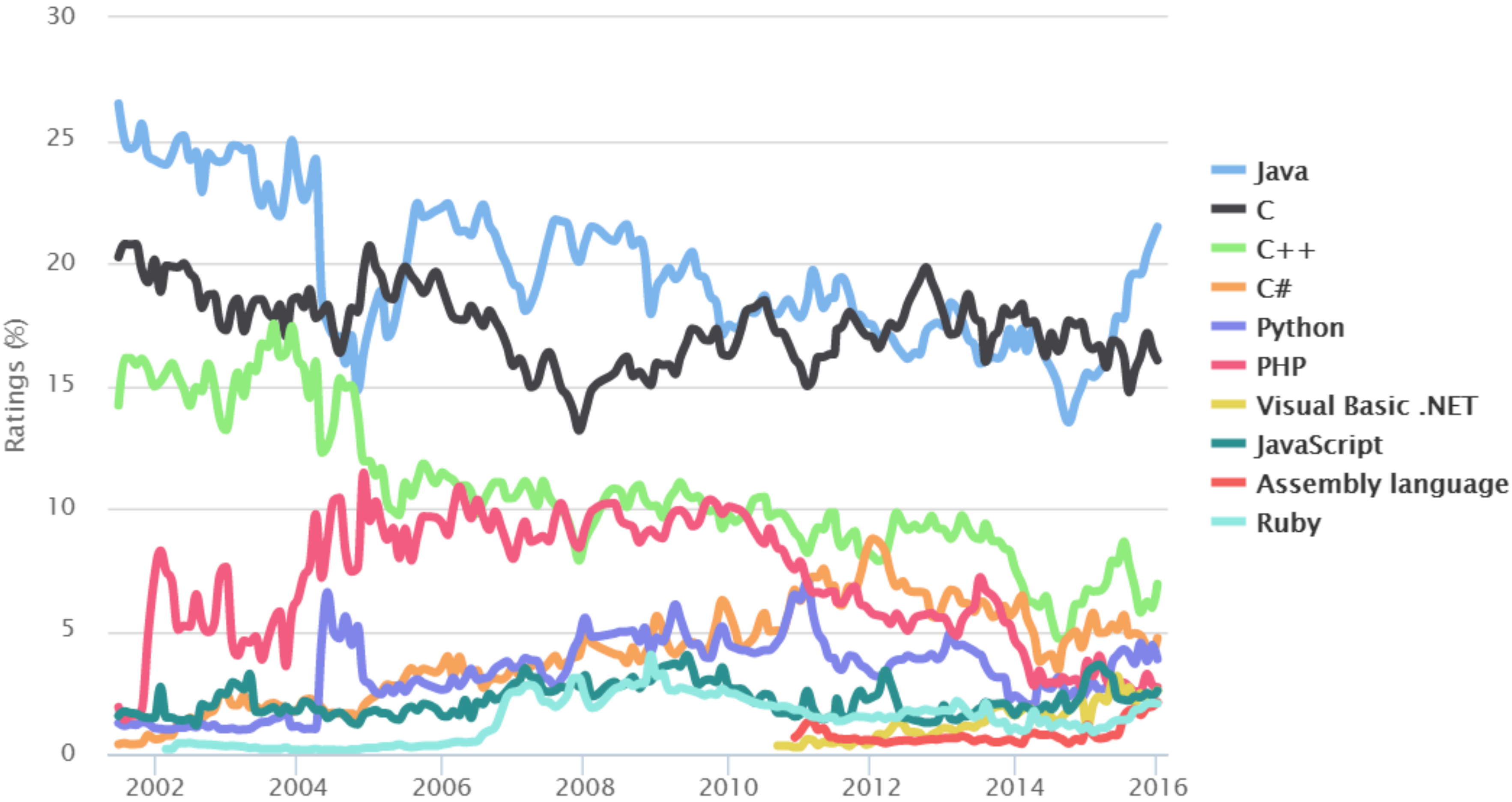
TIOBE Index of Language Popularity

Jan 2016	Jan 2015	Change	Programming Language	Ratings	Change
1	2	↑	Java	21.465%	+5.94%
2	1	↓	C	16.036%	-0.67%
3	4	↑	C++	6.914%	+0.21%
4	5	↑	C#	4.707%	-0.34%
5	8	↑	Python	3.854%	+1.24%
6	6		PHP	2.706%	-1.08%
7	16	⇈	Visual Basic .NET	2.582%	+1.51%
8	7	↓	JavaScript	2.565%	-0.71%
9	14	⇈	Assembly language	2.095%	+0.92%
10	15	⇈	Ruby	2.047%	+0.92%
11	9	↓	Perl	1.841%	-0.42%

The ratings are based on the number of skilled engineers world-wide, courses and third party vendors.

http://www.tiobe.com

TIOBE Programming Community Index



Disclaimer

- 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
    - Check online for more sources
  - "JAVA in a Nutshell," O'Reilly
    - Chapter 2, "How Java Differs from C"
    - http://oreilly.com/catalog/javanut/excerpt/index.html
  - Brian Harvey's helpful transition notes
    - On CS61C class website: pages 3-19
    - http://inst.eecs.berkeley.edu/~cs61c/resources/HarveyNotesC1-3.pdf
- Key C concepts: Pointers, Arrays, Implications for Memory management

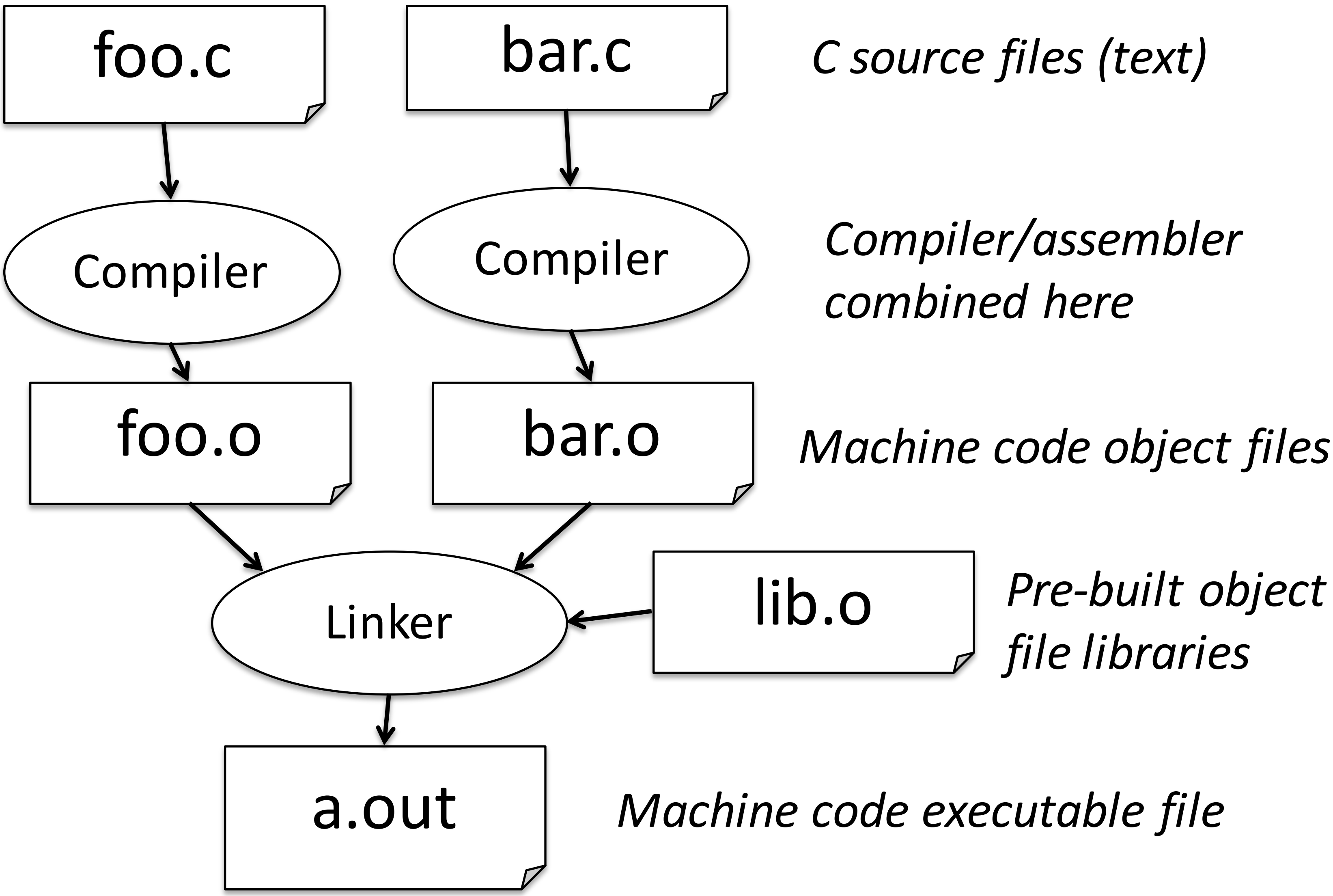
Agenda

- Everything is a Number
- Computer Organization
- Compile vs. Interpret

Compilation: Overview

- C *compilers* map C programs into architecture-specific machine code (string of 1s and 0s)
  - Unlike *Java*, which converts to architecture-independent *bytecode*
  - Unlike *Python* environments, which *interpret* the code
  - These differ mainly in exactly when your program is converted to low-level machine instructions ("levels of interpretation")
  - For C, generally a two 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); we'll talk about that later

C Compilation Simplified Overview (more later in course)



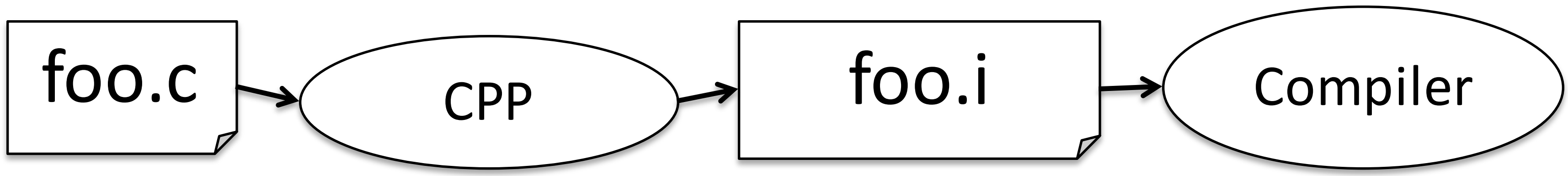
Compilation: Advantages

- Excellent run-time performance: generally much faster than Scheme or Java for comparable code (because it optimizes for a given architecture)
- Reasonable compilation time: enhancements in compilation procedure (Makefiles) allow only modified files to be recompiled

Compilation: Disadvantages

- Compiled files, including the executable, are architecture-specific, depending on processor type (e.g., MIPS vs. RISC-V) and the operating system (e.g., Windows vs. Linux)
- Executable must be rebuilt on each new system
  – I.e., "porting your code" to a new architecture
- "Change → Compile → Run [repeat]" iteration cycle can be slow during development
  – but Make tool only rebuilds changed pieces, and can do compiles in parallel (linker is sequential though -> Amdahl's Law)

C Pre-Processor (CPP)



- C source files first pass through macro processor, CPP, before compiler sees code
- CPP replaces comments with a single space
- CPP commands begin with "#"
- #include "file.h" /* Inserts file.h into output */
- #include <stdio.h> /* Looks for file in standard location */
- #define M_PI (3.14159) /* Define constant */
- #if/#endif /* Conditional inclusion of text */
- Use –save-temps option to gcc to see result of preprocessing
- Full documentation at: **`http://gcc.gnu.org/onlinedocs/cpp/`**