CENG 230

Introduction to C Programming

Week 2 – Overview of C

Sinan Kalkan

Some slides/content are borrowed from Tansel Dokeroglu, Nihan Kesim Cicekli.

Syllabus

Previously on CEng 230!

How to study?

Previously on CEng 230!

- Follow the lectures and the labs
- Read the textbook on a weekly basis
- Get your hands dirty
  - Do the exercises in front of the computer

Appointment

Previously on CEng 230!

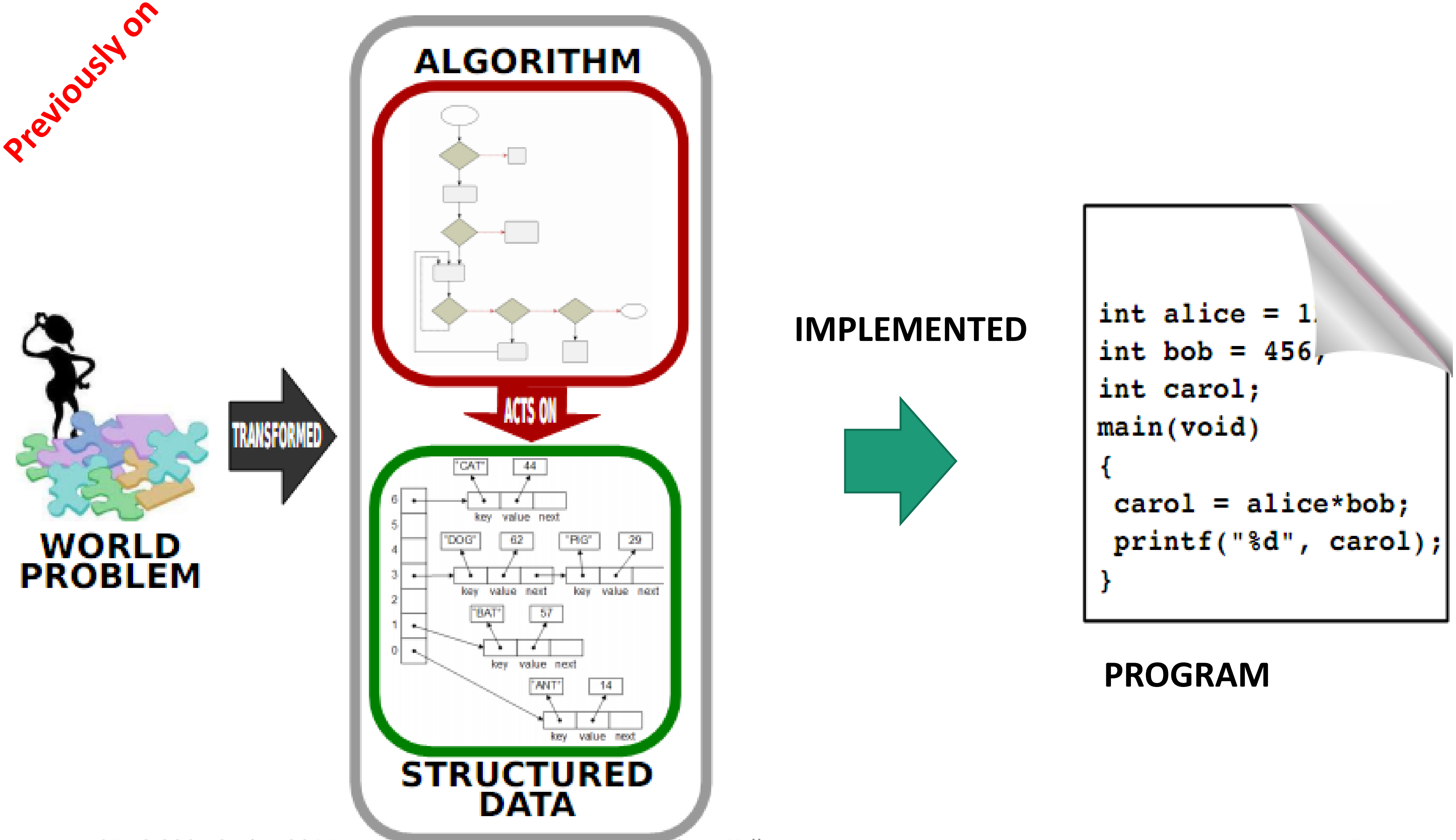
- No office hours. Make an appointment.
- Via email: skalkan@ceng.metu.edu.tr
- Office:

  Room B207,

  Department of Computer Engineering
- WWW:
  - http://kovan.ceng.metu.edu.tr/~sinan/

Program, Programming

Previously on CEng230!



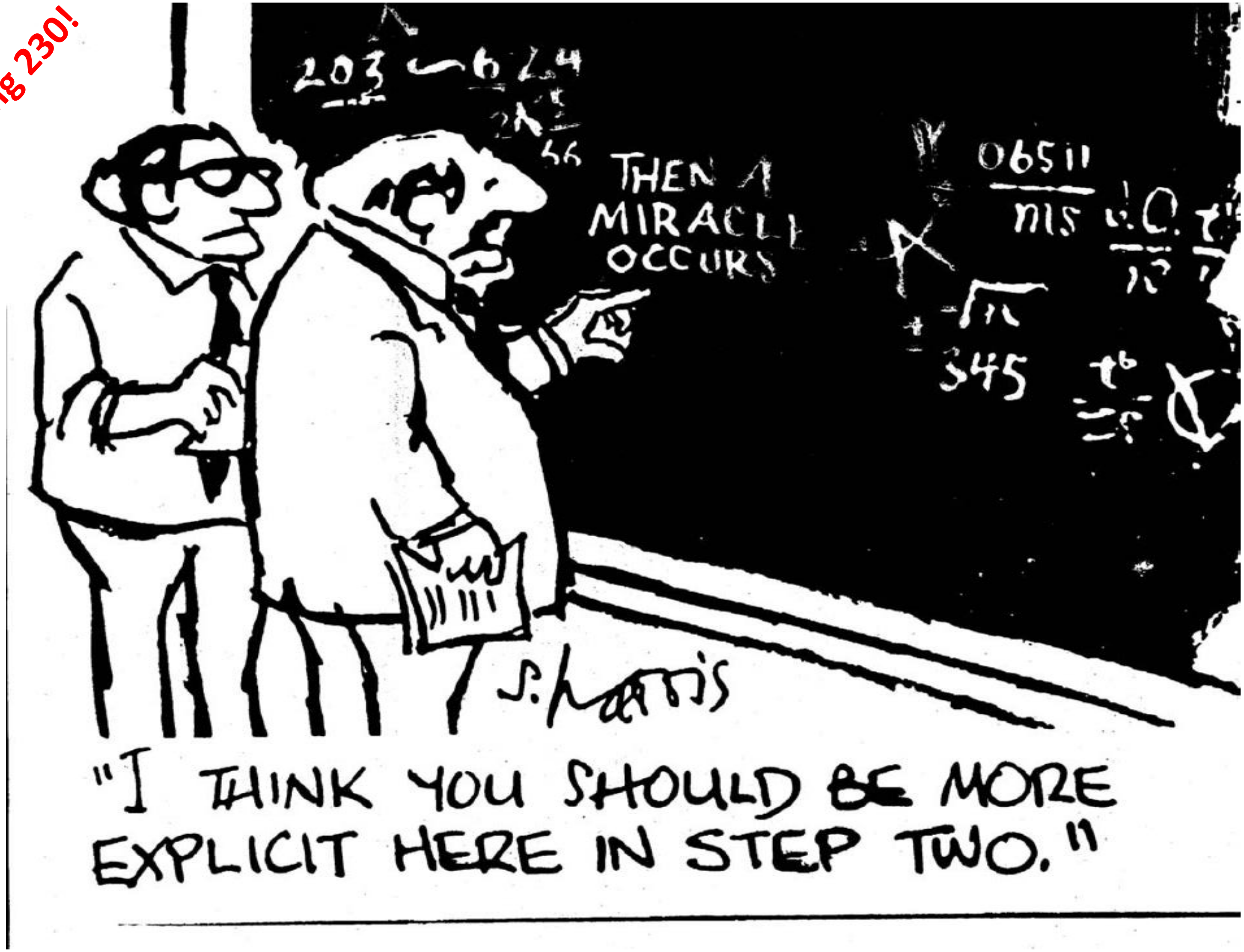
What is an algorithm?

Previously on CEng 230!

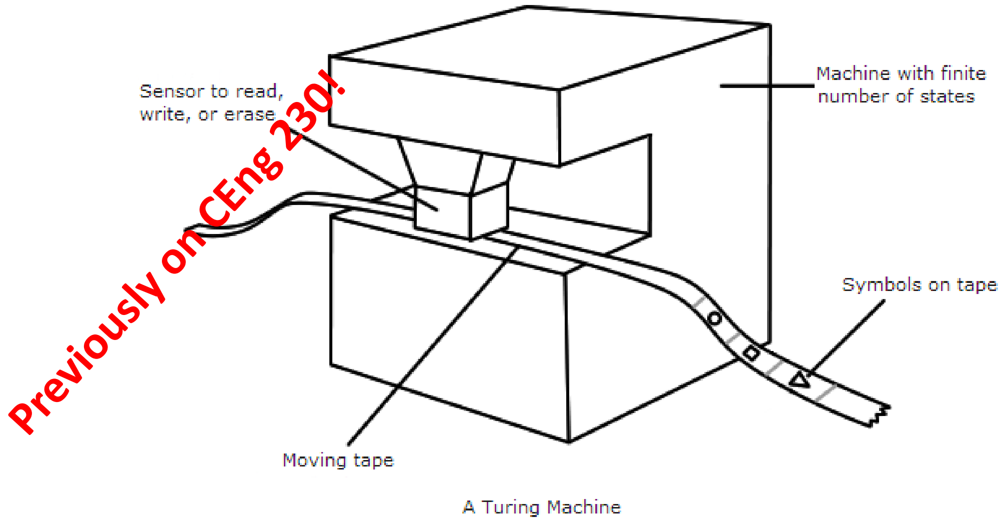
- An algorithm is a list that looks like
  - STEP 1: Do something
  - STEP 2: Do something
  - STEP 3: Do something
  - . .
  - . .
  - . .
  - STEP N: Stop, you are finished

From “Invitation to Computer Science”

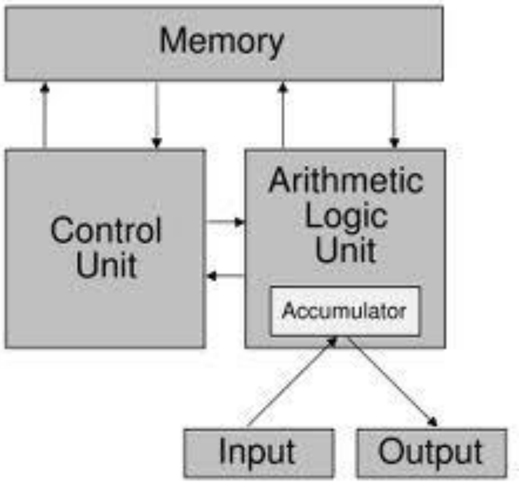
Previously on CEng 230!



From "Invitation to Computer Science"



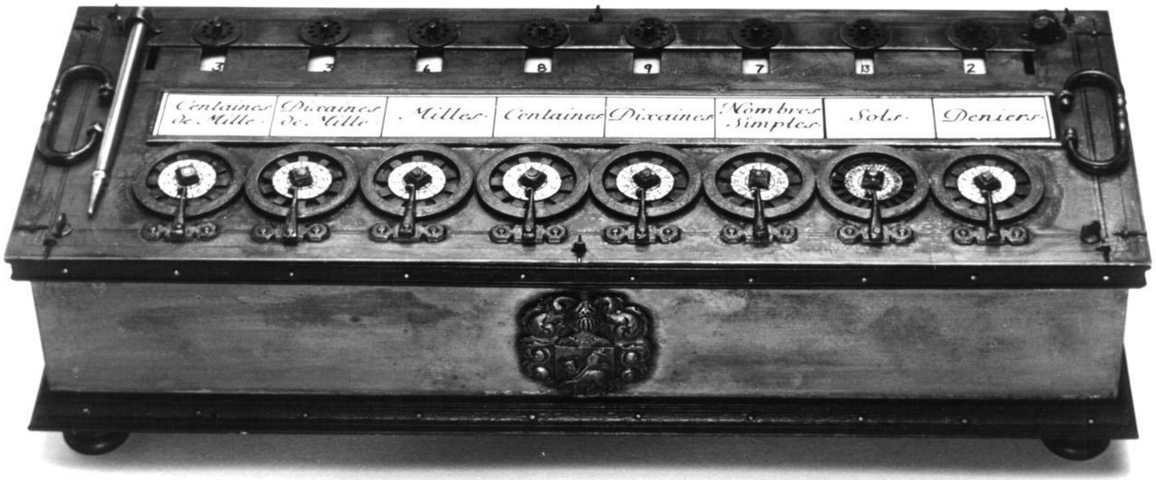
Turing Machine



Von Neumann Architecture

DIGITAL COMPUTATION

Previously on CEng 230!

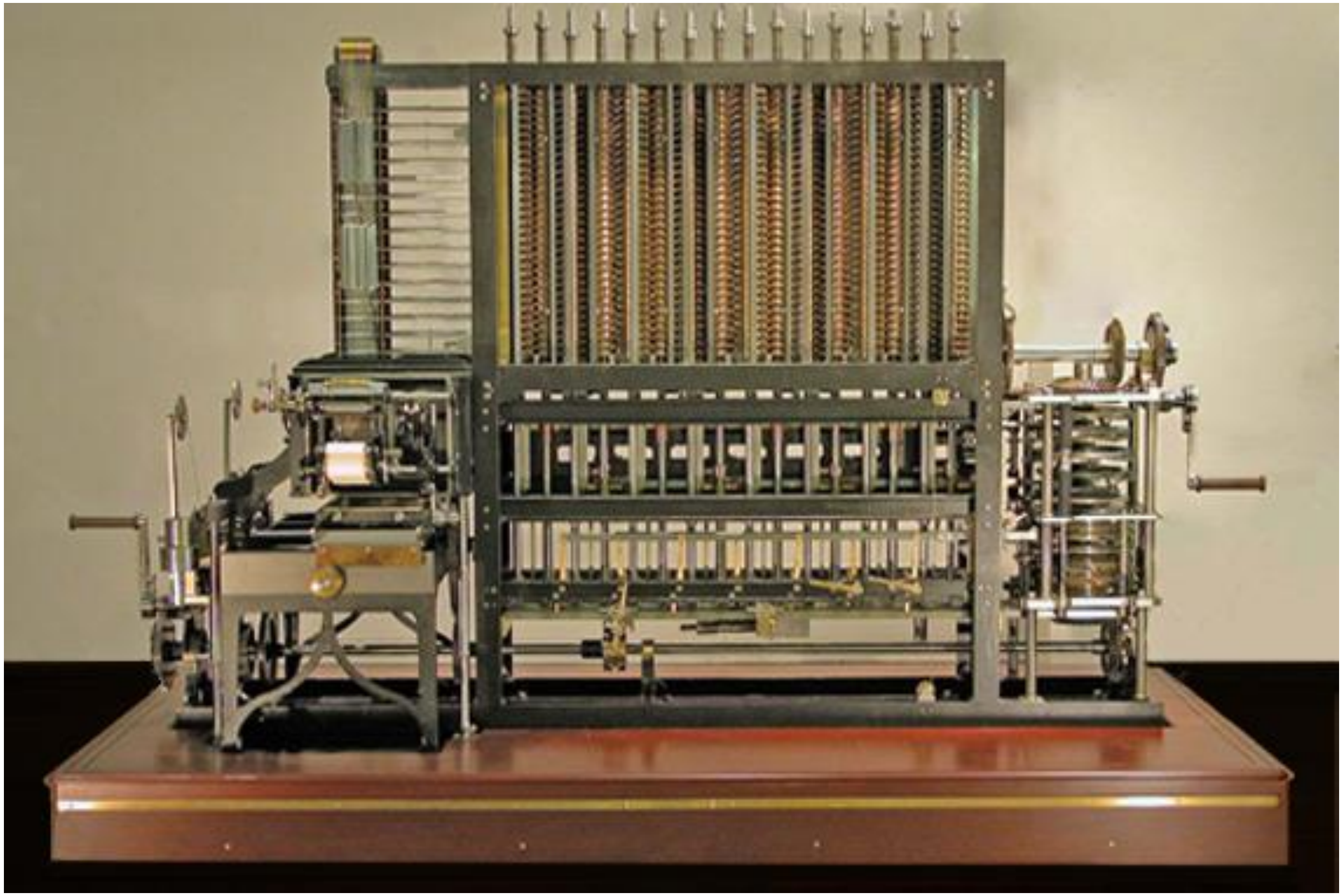


The Pascaline: One of the Earliest Mechanical Calculators

Slide from “Introduction to Computing”

Difference engine

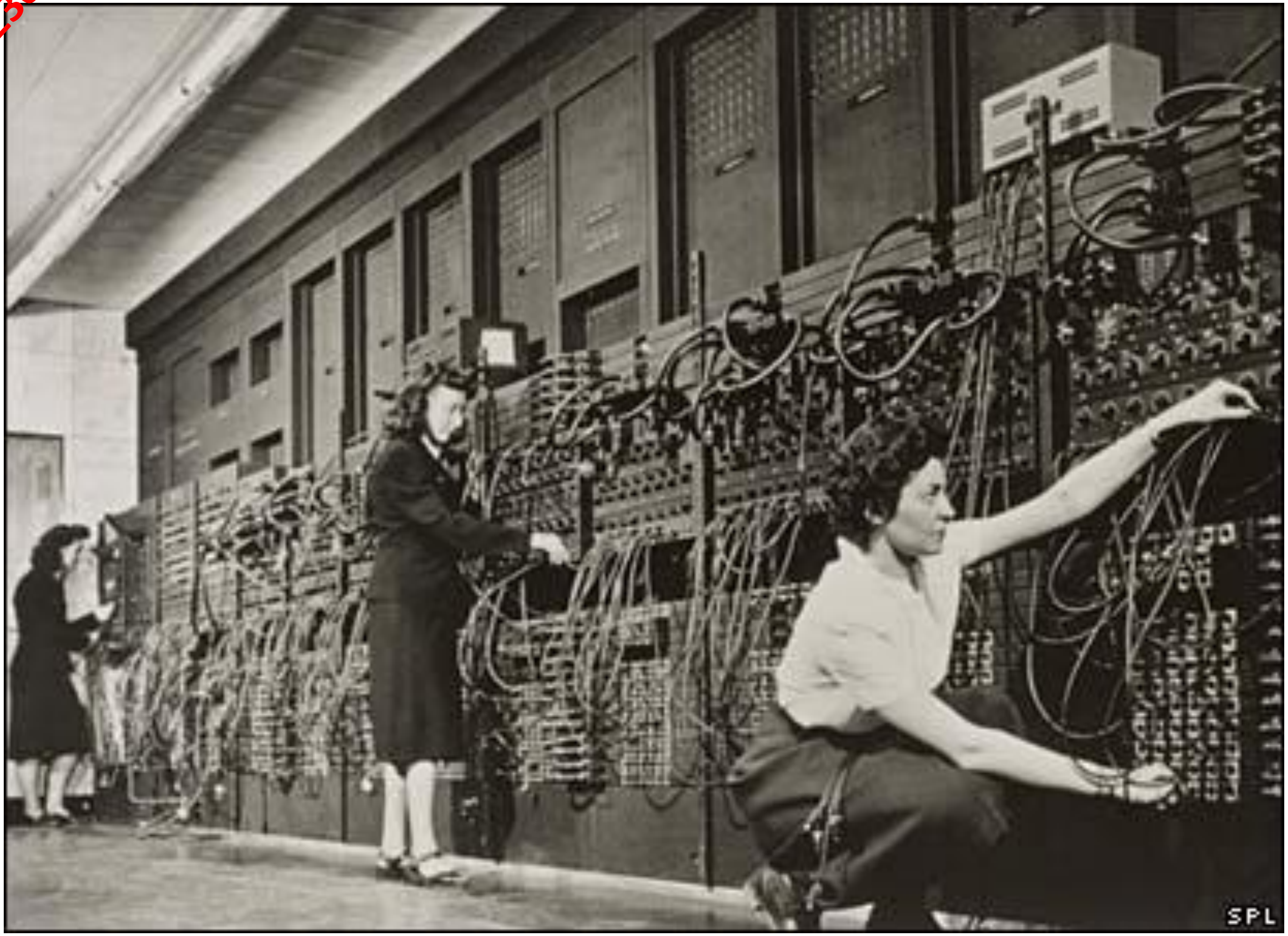
Previously on CEng 230!



http://www.youtube.com/watch?v=0anIyVGeWOI

Slide from "Introduction to Computing"

Previously on CEng 230!



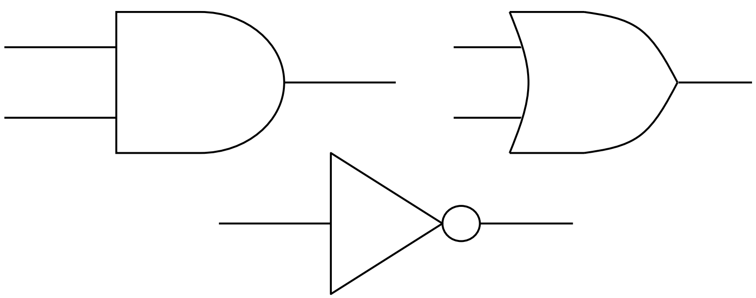
Programming the ENIAC

Slide from "Introduction to Computing"

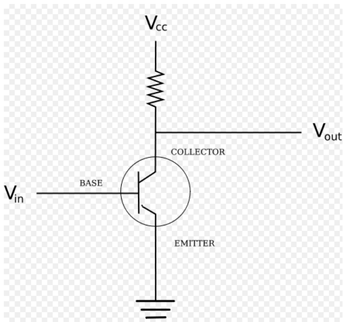
Previously on CEng 230!

A computer

Devices



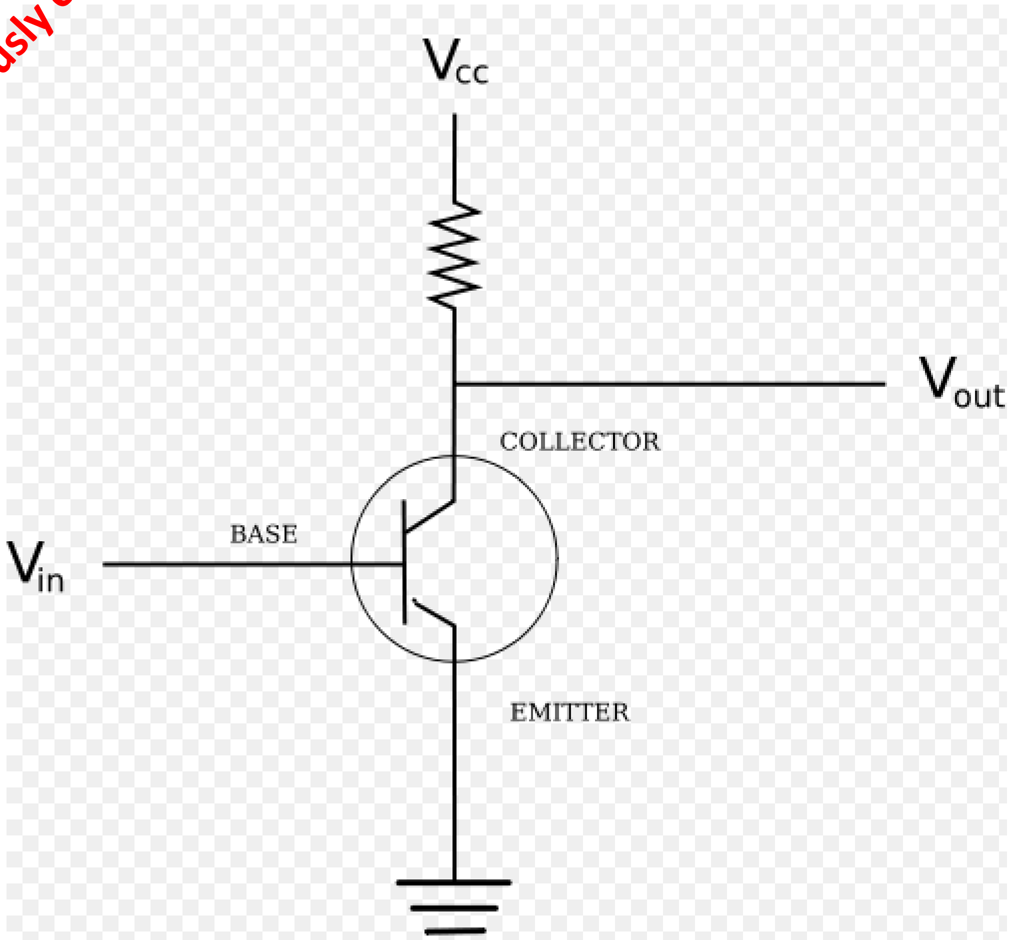
Gates



Transistors

A transistor

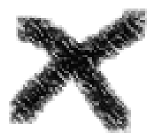
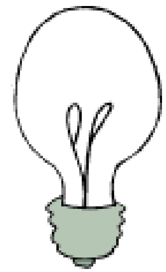
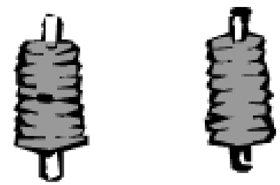
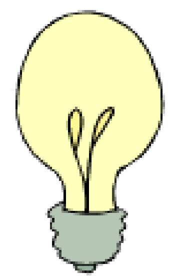
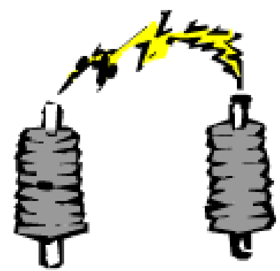
Previously on CEng 230!



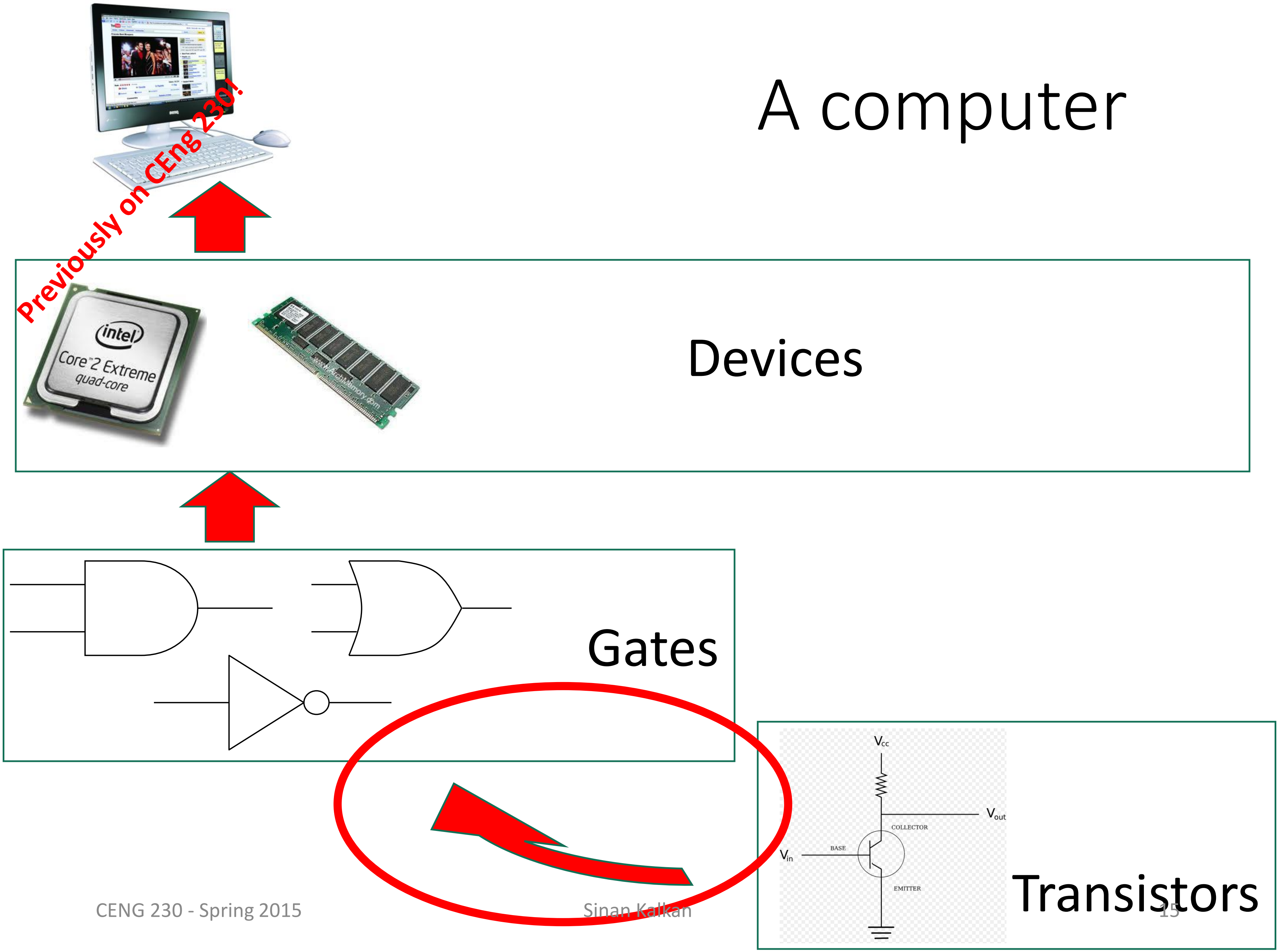
This circuit functions as a switch. In other words, based on the *control* voltage, the circuit either passes Vin to output or not.

Everything in a PC is Binary … well, almost …

Previously on CEng 230!

States of a Bit			
0	 2+2=5 FALSE	 OFF	 LOW VOLTAGE
1	 2+2=4 TRUE	 ON	 HIGH VOLTAGE

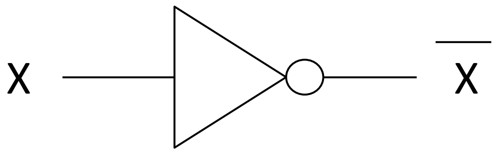
A computer


Previously on CEng 230!
intel
Core 2 Extreme
quad-core
Devices
Gates
Transistors
Vcc
Vout
COLLECTOR
BASE
Vin
EMITTER

NOT Gate

Previously on CEng 230!
Previously on CEng 230!

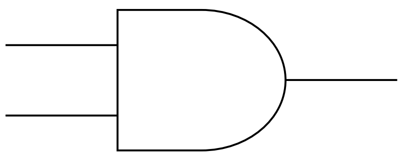
X	$\overline{X}$
0	1
1	0



AND gate

Previously on CEng 230!

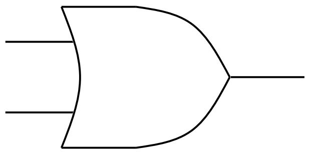
X	Y	X·Y
0	0	0
0	1	0
1	0	0
1	1	1



OR Gate

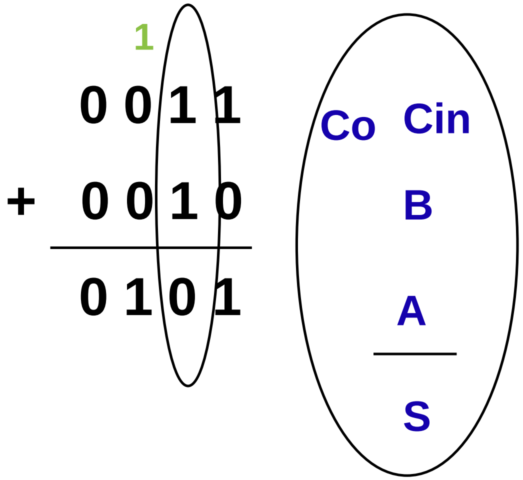
Previously on CEng 230!

X	Y	X+Y
0	0	0
0	1	1
1	0	1
1	1	1



1-bit full-adder

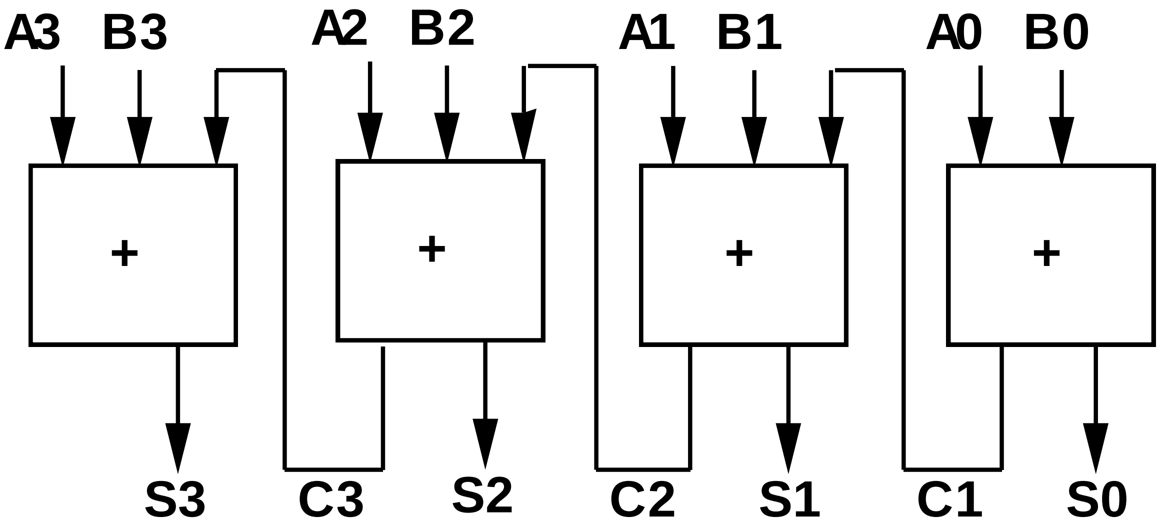
Previously on CENg 230!



A	B	CI	S	CO
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

N-bit Adder

Previously on CEng 230!



Data Representation

Previously on CEng 230!

- Based on 1s and 0s
  - So, everything is represented as a set of binary numbers
- We will now see how we can represent:
  - Integers: 3, 1234435, -12945 etc.
  - Floating point numbers: 4.5, 124.3458, -1334.234 etc.
  - Characters: /, &, +, -, A, a, ^, 1, etc.
  - …

Binary Representation of Numeric Information

Previously on CEng 230!

- Decimal numbering system
  - Base-10
  - Each position is a power of 10

    $3052 = 3 \times 10^3 + 0 \times 10^2 + 5 \times 10^1 + 2 \times 10^0$

- Binary numbering system
  - Base-2
  - Uses ones and zeros
  - Each position is a power of 2

    $(1101)_2 = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$

Decimal-to-binary Conversion

Previously on CEng 230!

- Divide the number until zero:
  - 35 / 2 = 17x2 + 1
  - 17 / 2 = 8x2 + 1
  - 8 / 2 = 4x2 + 0
  - 4 / 2 = 2x2 + 0
  - 2 / 2 = 1x2 + 0
- Therefore, 35 has the binary representation: 100011

IEEE 32bit Floating-Point Number Representation

Previously on CEng 230!

- Example: 12.375
- The digits before the dot:
  - $(12)_{10} \rightarrow (1100)_2$
- The digits after the dot:
  - 1st Way: 0.375 → 0x½ + 1x¼ + 1x1/8 → 011
  - 2nd Way: Multiply by 2 and get the integer part until 0:
    - 0.375 x 2 = 0.750 = 0 + 0.750
    - 0.750 x 2 = 1.50 = 1 + 0.50
    - 0.50 x 2 = 1.0 = 1 + 0.0
- $(12.375)_{10} = (1100.011)_2$
- NORMALIZE: $(1100.011)_2 = (1.100011)_2 \times 2^3$
- Exponent: 3, adding 127 to it, we get 1000 0010
- Fraction: 100011
- Then our number is: 0 10000010 10001100000000000000000

Binary Representation of Textual Information (cont'd)

Previously on CEng 230!

ASCII
7 bits long

Decimal	Binary	Val.
48	00110000	0
49	00110001	1
50	00110010	2
51	00110011	3
52	00110100	4
53	00110101	5
54	00110110	6
55	00110111	7
56	00111000	8
57	00111001	9
58	00111010	:
59	00111011	;
60	00111100	<
61	00111101	=
62	00111110	>
63	00111111	?
64	01000000	@
65	01000001	A
66	01000010	B

Dec.	Unicode	Charac.
0x30	0x0030	0
0x31	0x0031	1
0x32	0x0032	2
0x33	0x0033	3
0x34	0x0034	4
0x35	0x0035	5
0x36	0x0036	6
0x37	0x0037	7
0x38	0x0038	8
0x39	0x0039	9
0x3A	0x003A	:
0x3B	0x003B	;
0x3C	0x003C	<
0x3D	0x003D	=
0x3E	0x003E	>
0x3F	0x003F	?
0x40	0x0040	@
0x41	0x0041	A
0x42	0x0042	B

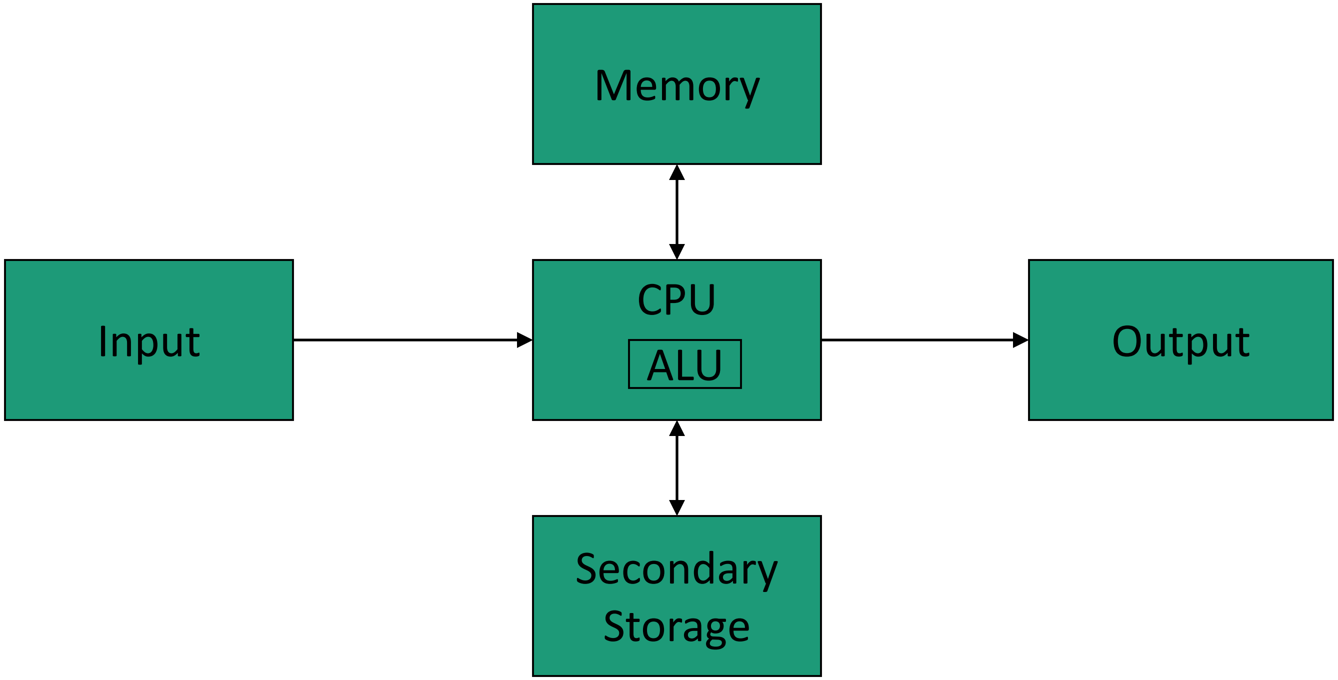
Unicode
16 bits long

Partial listings only!

Computer Organization

Previously on CEng 230!

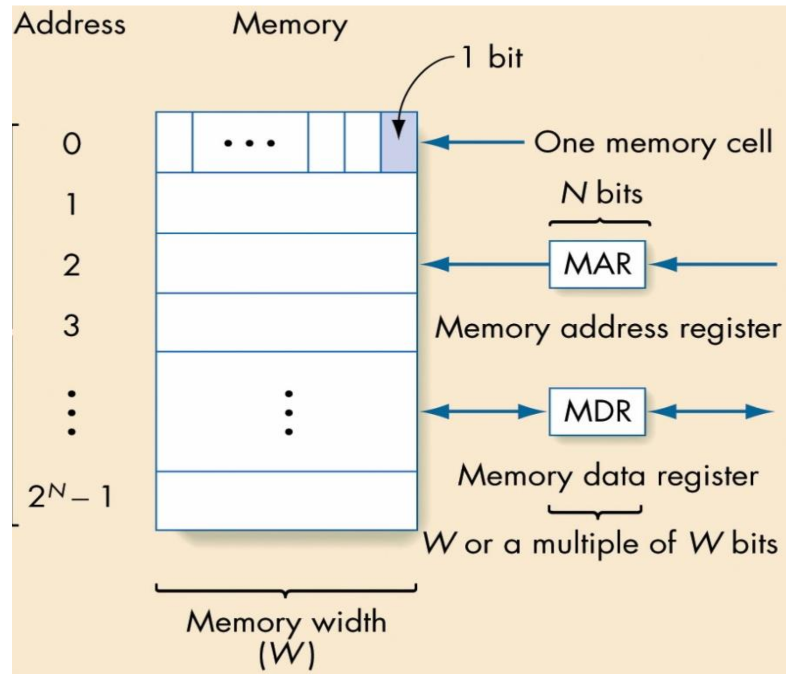
- Logical organization of computer



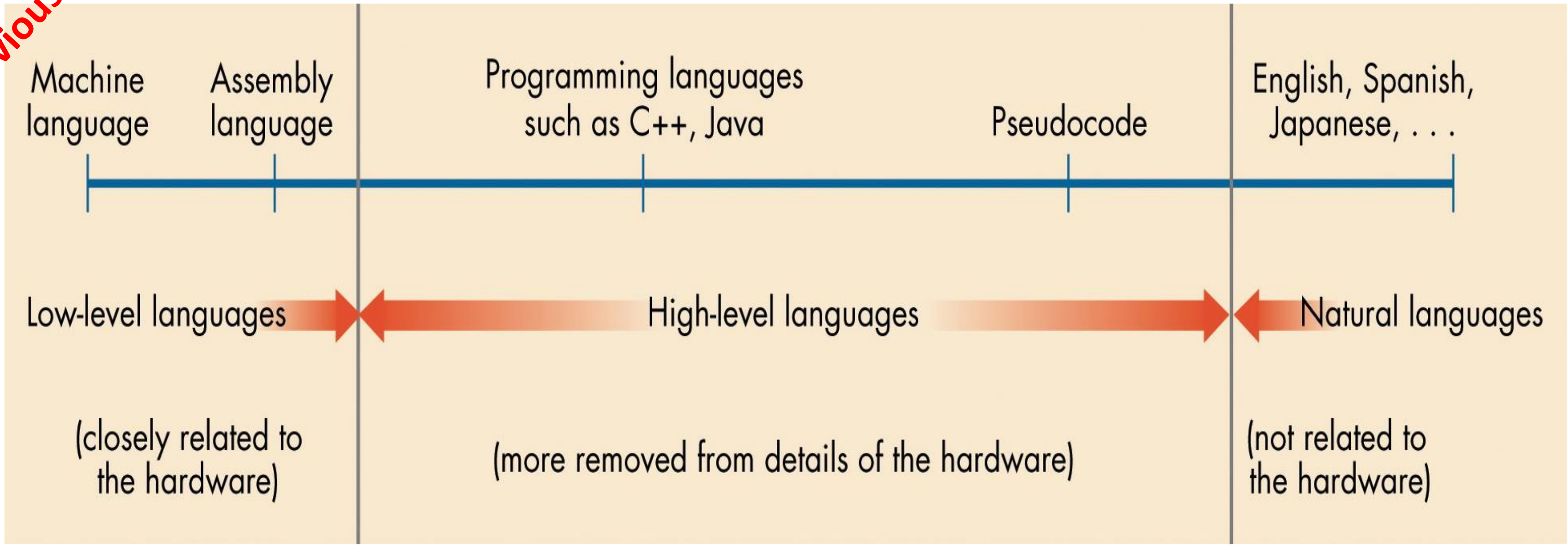
Memory and Cache (continued)

Previously on CEng 230!

- RAM (Random Access Memory)
  Often called *memory, primary memory*
  - Memory made of addressable "cells"
  - Cell size is 8 bits
    - Nowadays, it is 32 or 64 bits.
  - All memory cells accessed in equal time
  - Memory address
    - Unsigned binary number N long
    - Address space is then 2^N cells



Previously on CEng 230!



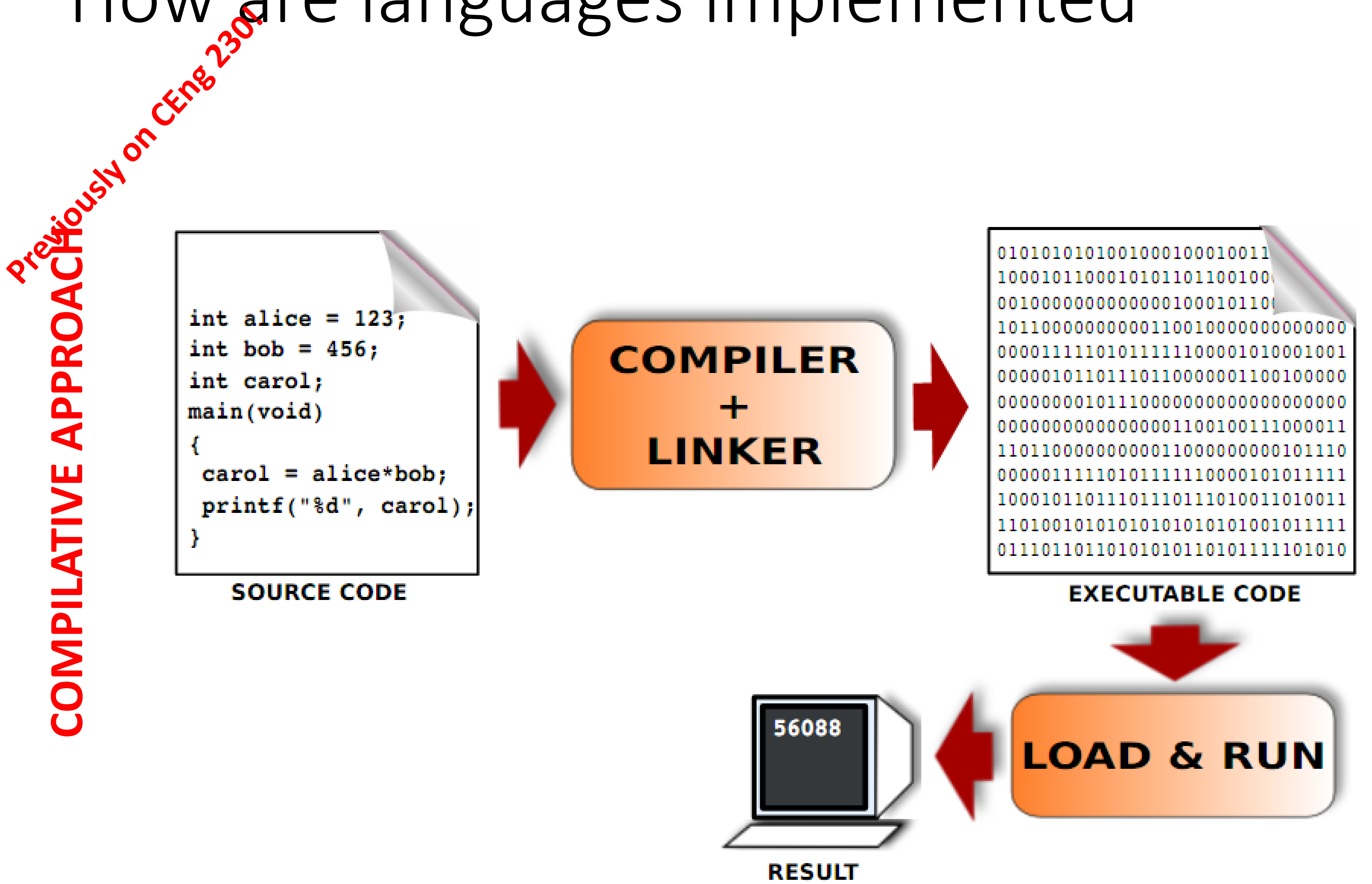
Previously on CEng 230!

```
01010101 01001000 10001001 11100101 10001011 00010101 10110010 00000011
00100000 00000000 10001011 00000101 10110000 00000011 00100000 00000000
00001111 10101111 11000010 10001001 00000101 10111011 00000011 00100000
00000000 10111000 00000000 00000000 00000000 00000000 11001001 11000011
...
11001000 00000001 00000000 00000000  00000000 00000000
```

```
main:
        pushq   %rbp
        movq    %rsp, %rbp
        movl    alice(%rip), %edx
        movl    bob(%rip), %eax
        imull   %edx, %eax
        movl    %eax, carol(%rip)
        movl    $0, %eax
        leave
        ret
alice:
        .long   123
bob:
        .long   456
```

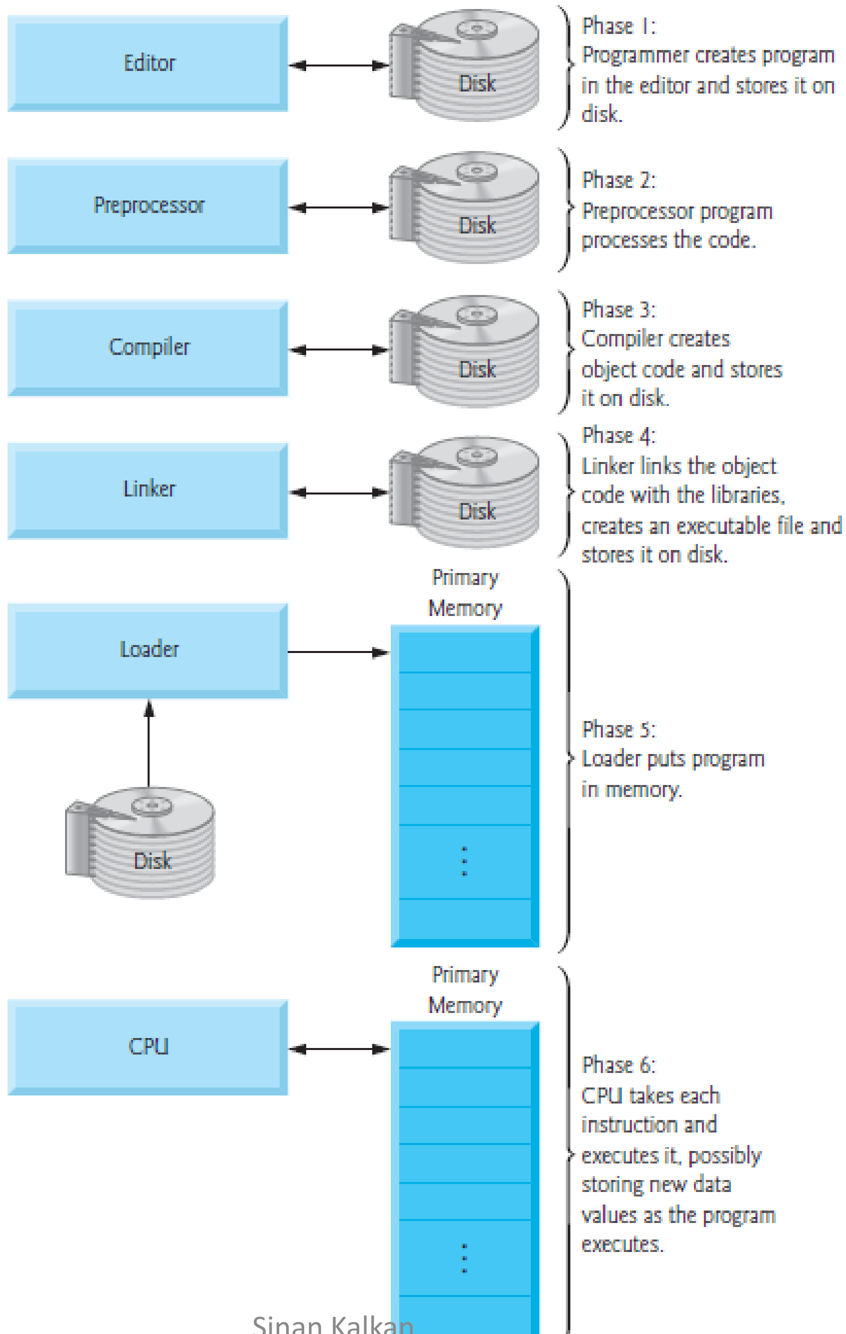
```
int alice = 123;
int bob = 456;
int carol;
main(void)
{
  carol = alice*bob;
}
```

How are languages implemented



Previously on CEng 230!

C language development environment



Bugs, Errors

Previously on CEng 230!

- Syntax Errors

  Area = 3.1415 * R * R

  Area = 3.1415 x R x R

- Run-time Errors

```
>>> def SqrtDelta(a,b,c):
>>>         return sqrt(b*b - 4*a*c)
>>>
>>> print SqrtDelta(1,3,1)
2.23606797749979
>>> print SqrtDelta(1,1,1)
ValueError: math domain error
```

Bugs, Errors

Previously on CEng 230!

- Logical Errors

$$root_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

```
>>> root1 = (- b + sqrt(b*b - 4*a*c)) / 2*a
```

- Design Errors

$$x^3 + ax^2 + bx + c = 0$$

$$root_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

Previously on CEng 230!

C

Previously on Ceng 230!

History of C

- C

– Developed by Denis M. Ritchie at AT&T Bell Labs in **1972** as a systems programming language

– Used to develop UNIX

– Used to write modern operating systems

– Hardware independent (portable)

- Standardization

– Many slight variations of C existed, and were incompatible

– Committee formed to create a "unambiguous, machine independent“ definition

– Standard created in 1989, updated in 1999

Previously on CEng 230!

```c
/* Fig. 2.1: fig02_01.c
   A first program in C */
#include <stdio.h>

/* function main begins program execution */
int main( void )
{
   printf( "Welcome to C!\n" );

   return 0; /* indicate that program ended successfully */
} /* end function main */
```

```
Welcome to C!
```

Fig. 2.1 | A first program in C.

Notes

- The lectures notes:

  http://www.kovan.ceng.metu.edu.tr/~sinan/ceng230/

  - You can also just google my name → Follow "Courses" → "Ceng 230"

- Location:
  - Starting from "5 March, 2015", lectures will be held in BMB-1 (in Computer Engineering Dept.)

- Midterm:
  - 28 April, 2015 at 17:40

Today

- Continue with the overview of C
- Introduction of the basic concepts

- Let me collect the assignment

Variables and identifiers

C has the following basic built-in datatypes.

- int
- float
- double
- char

TABLE 2.4 Type double Constants (real numbers)

Valid double Constants	Invalid double Constants
3.14159	150 (no decimal point)
0.005	.12345e (missing exponent)
12345.0	15e-0.3 (0.3 is invalid exponent)
15.0e-04 (value is 0.0015)	
2.345e2 (value is 234.5)	12.5e.3 (.3 is invalid exponent)
1.15e-3 (value is 0.00115)	34,500.99 (comma is not allowed)
12e+5 (value is 1200000.0)	

Valid Identifiers

`letter_1, letter_2, inches, cent, CENT_PER_INCH, Hello, variable`

TABLE 2.2 Invalid Identifiers

Invalid Identifier	Reason Invalid
`1Letter`	begins with a letter
`double`	reserved word
`int`	reserved word
`TWO*FOUR`	character * not allowed
`joe's`	character ' not allowed

int1 and Int1 are not the same identifiers/variables

Keywords			
auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while
Keywords added in C99			
_Bool _Complex _Imaginary inline restrict			

Fig. 2.15 | C's keywords.

Basic Input/Output in C

Output

- **printf**(format string, var1, var2, …)
  - Format string contains:
    - d,i: integers
    - f: float, double
    - e: float, double in exponential notation
    - c: character
    - s: string

Input

- **scanf**(format string, &var1, &var2, …)
  - var1, var2, ..: variables!
  - Format string contains:
    - d,i: integers
    - f: float, double
    - e: float, double in exponential notation
    - c: character
    - s: string

```c
/* Fig. 2.3: fig02_03.c
   Printing on one line with two printf statements */
#include <stdio.h>

/* function main begins program execution */
int main( void )
{
   printf( "Welcome " );
   printf( "to C!\n" );

   return 0; /* indicate that program ended successfully */
} /* end function main */
```

Fig. 2.3 | Printing on one line with two `printf` statements. (Part 1 of 2.)

```
Welcome to C!
```

```c
/* Fig. 2.4: fig02_04.c
   Printing multiple lines with a single printf */
#include <stdio.h>

/* function main begins program execution */
int main( void )
{
   printf( "Welcome\nto\nC!\n" );

   return 0; /* indicate that program ended successfully */
} /* end function main */
```

```
Welcome
to
C!
```

Fig. 2.4 | Printing multiple lines with a single printf.

Escape sequence	Description
`\n`	Newline. Position the cursor at the beginning of the next line.
`\t`	Horizontal tab. Move the cursor to the next tab stop.
`\a`	Alert. Sound the system bell.
`\\`	Backslash. Insert a backslash character in a string.
`\"`	Double quote. Insert a double-quote character in a string.

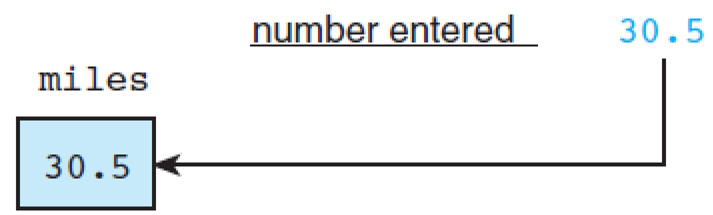
Fig. 2.2 | Some common escape sequences .

TABLE 2.8 Placeholders in Format Strings

Placeholder	Variable Type	Function Use
%c	char	printf/scanf
%d	int	printf/scanf
%f	double	printf
%lf	double	scanf

FIGURE 2.6

Effect of `scanf("%lf", &miles);`




```
int first, second;
scanf("%d%d", &first, &second);
```

```
double miles;  /* distance in miles        */
scanf("%lf", &miles);
```

```
/* Fig. 2.5: fig02_05.c
   Addition program */
#include <stdio.h>

/* function main begins program execution */
int main( void )
{
   int integer1; /* first number to be input by user  */
   int integer2; /* second number to be input by user */
   int sum; /* variable in which sum will be stored */

   printf( "Enter first integer\n" ); /* prompt */
   scanf( "%d", &integer1 ); /* read an integer */

   printf( "Enter second integer\n" ); /* prompt */
   scanf( "%d", &integer2 ); /* read an integer */

   sum = integer1 + integer2; /* assign total to sum */

   printf( "Sum is %d\n", sum ); /* print sum */

   return 0; /* indicate that program ended successfully */
} /* end function main */
```

```
Enter first integer
45
Enter second integer
72
Sum is 117
```

Fig. 2.5 | Addition program (Part 2 of 2)

Variable names such as `integer1`, `integer2` and `sum` actually correspond to locations in the computer's memory. Every variable has a name, a type and a value.

In the addition program of Fig. 2.5, when the statement (line 13)

```
scanf( "%d", &integer1 ); /* read an integer */
```

is executed, the value typed by the user is placed into a memory location to which the name `integer1` has been assigned. Suppose the user enters the number `45` as the value for `integer1`. The computer will place `45` into location `integer1` as shown in Fig. 2.6.

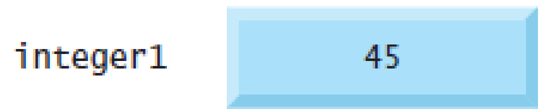


Fig. 2.6 | Memory location showing the name and value of a variable.

Whenever a value is placed in a memory location, the value replaces the previous value in that location; thus, placing a new value into a memory location is said to be destructive.

Formating the output of integer values

Specifying the format of an integer value displayed by a C program is fairly easy. You simply add a number between the `%` and the `d` of the `%d` placeholder in the `printf` format string. This number specifies the **field width**—the number of columns to use for the display of the value. The statement

```
printf("Results: %3d meters = %4d ft. %2d in.\n",
       meters, feet, inches);
```

indicates that `3` columns will be used to display the value of `meters`, `4` columns will be used for `feet`, and `2` columns will be used for `inches` (a number between `0` and `11`). If `meters` is `21`, `feet` is `68`, and `inches` is `11`, the program output will be

```
Results:  21 meters =   68 ft. 11 in.
```

TABLE 2.14 Displaying 234 and –234 Using Different Placeholders

Value	Format	Displayed Output	Value	Format	Displayed Output
`234`	`%4d`	`▮234`	`-234`	`%4d`	`-234`
`234`	`%5d`	`▮▮234`	`-234`	`%5d`	`▮-234`
`234`	`%6d`	`▮▮▮234`	`-234`	`%6d`	`▮▮-234`
`234`	`%1d`	`234`	`-234`	`%2d`	`-234`

Formating the output of double values

TABLE 2.16 Formatting Type double Values

Value	Format	Displayed Output	Value	Format	Displayed Output
3.14159	%5.2f	▮3.14	3.14159	%4.2f	3.14
3.14159	%3.2f	3.14	3.14159	%5.1f	▮▮3.1
3.14159	%5.3f	3.142	3.14159	%8.5f	▮3.14159
.1234	%4.2f	0.12	-.006	%4.2f	-0.01
-.006	%8.3f	▮▮-0.006	-.006	%8.5f	-0.00600
-.006	%.3f	-0.006	-3.14159	%.4f	-3.1416

Operators and Expressions

C operation	Arithmetic operator	Algebraic expression	C expression
Addition	+	$f + 7$	`f + 7`
Subtraction	–	$p - c$	`p - c`
Multiplication	*	bm	`b * m`
Division	/	x / y or $\frac{x}{y}$ or $x \div y$	`x / y`
Remainder	%	$r \bmod s$	`r % s`

```
printf( "Welcome to \%d", (3/2) );
```

Output is : 1

Operator(s)	Operation(s)	Order of evaluation (precedence)
( )	Parentheses	Evaluated first. If the parentheses are nested, the expression in the innermost pair is evaluated first. If there are several pairs of parentheses "on the same level" (i.e., not nested), they're evaluated left to right.
* / %	Multiplication Division Remainder	Evaluated second. If there are several, they're evaluated left to right.
+ -	Addition Subtraction	Evaluated last. If there are several, they're evaluated left to right.

TABLE 2.9 Arithmetic Operators

Arithmetic Operator	Meaning	Examples
`+`	addition	`5 + 2 is 7` `5.0 + 2.0 is 7.0`
`-`	subtraction	`5 - 2 is 3` `5.0 - 2.0 is 3.0`
`*`	multiplication	`5 * 2 is 10` `5.0 * 2.0 is 10.0`
`/`	division	`5.0 / 2.0 is 2.5` `5 / 2 is 2`
`%`	remainder	`5 % 2 is 1`

TABLE 2.10 Results of Integer Division

3 / 15 = 0	18 / 3 = 6
15 / 3 = 5	16 / −3 = -5
16 / 3 = 5	0 / 4 = 0
17 / 3 = 5	4 / 0 is undefined

TABLE 2.11 Results of % Operation

3 % 5 = 3	5 % 3 = 2
4 % 5 = 4	5 % 4 = 1
5 % 5 = 0	15 % 5 = 0
6 % 5 = 1	15 % 6 = 3
7 % 5 = 2	15 % −7 = 1
8 % 5 = 3	15 % 0 is undefined

TABLE 2.13 Mathematical Formulas as C Expressions

Mathematical Formula	C Expression
1. $b^2 - 4ac$	`b * b - 4 * a * c`
2. $a + b - c$	`a + b - c`
3. $\frac{a + b}{c + d}$	`(a + b) / (c + d)`
4. $\frac{1}{1 + x^2}$	`1 / (1 + x * x)`
5. $a \times -(b + c)$	`a * -(b + c)`

Rules for Evaluating Expressions

a. *Parentheses rule:* All expressions in parentheses must be evaluated separately. Nested parenthesized expressions must be evaluated from the inside out, with the innermost expression evaluated first.

b. *Operator precedence rule:* Operators in the same expression are evaluated in the following order:

unary `+`, `-`	first
`*`, `/`, `%`	next
binary `+`, `-`	last

c. *Associativity rule:* Unary operators in the same subexpression and at the same precedence level (such as + and −) are evaluated right to left (*right associativity*). Binary operators in the same subexpression and at the same precedence level (such as + and −) are evaluated left to right (*left associativity*).

Step 1.	`y = 2 * 5 * 5 + 3 * 5 + 7;` `2 * 5 is 10`	*(Leftmost multiplication)*
Step 2.	`y = 10 * 5 + 3 * 5 + 7;` `10 * 5 is 50`	*(Leftmost multiplication)*
Step 3.	`y = 50 + 3 * 5 + 7;` `3 * 5 is 15`	*(Multiplication before addition)*
Step 4.	`y = 50 + 15 + 7;` `50 + 15 is 65`	*(Leftmost addition)*
Step 5.	`y = 65 + 7;` `65 + 7 is 72`	*(Last addition)*
Step 6.	`y = 72`	*(Last operation—place* `72` *in* `y`*)*

Fig. 2.11 | Order in which a second-degree polynomial is evaluated.

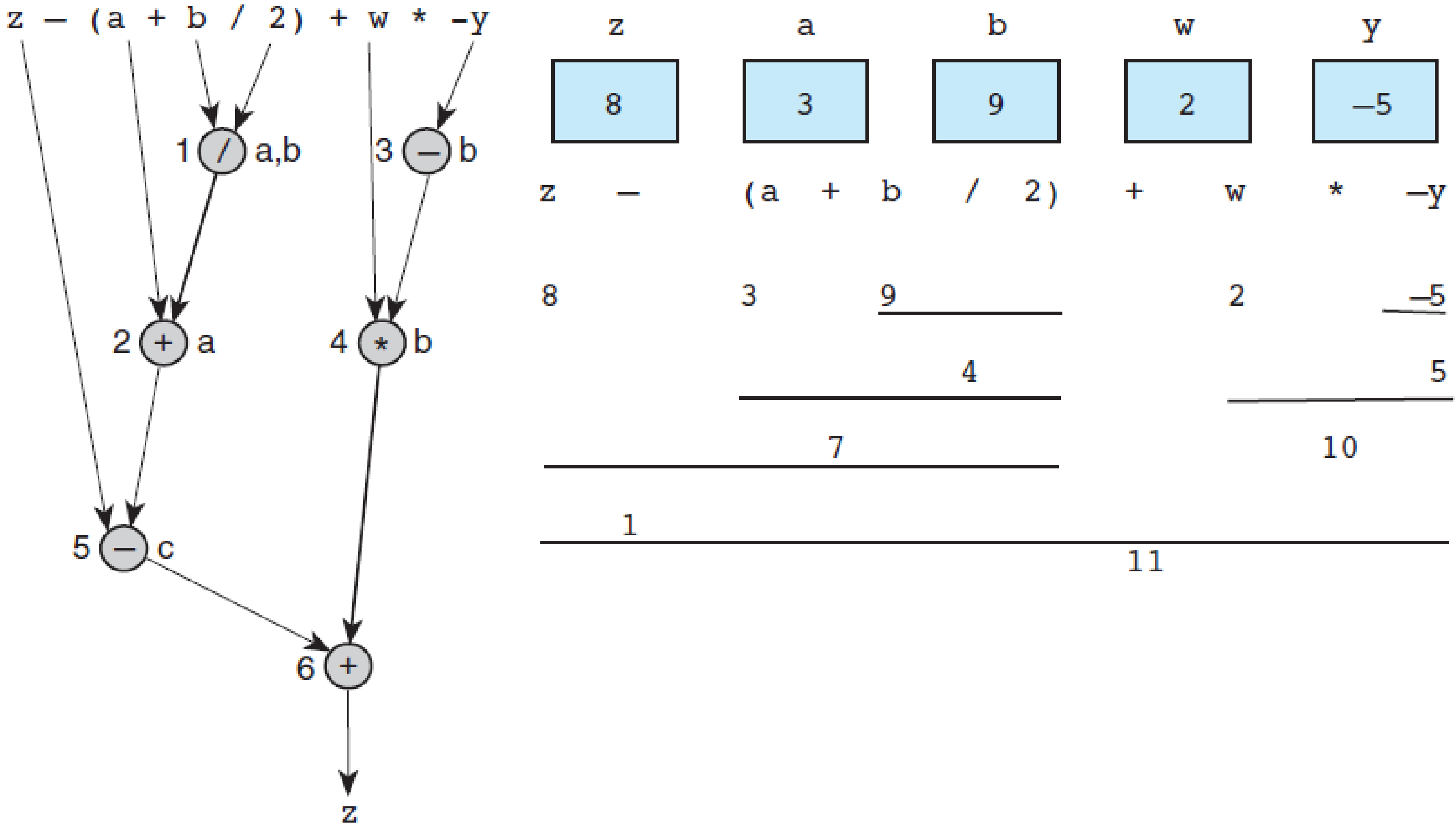
Evaluation of a Second-Degree Polynomial

To develop a better understanding of the rules of operator precedence, let's see how C evaluates a second-degree polynomial.

```
y = a * x * x + b * x + c;
  6   1   2   4   3   5
```

Step 1.	`y = 2 * 5 * 5 + 3 * 5 + 7;` `2 * 5 is 10`	*(Leftmost multiplication)*
Step 2.	`y = 10 * 5 + 3 * 5 + 7;` `10 * 5 is 50`	*(Leftmost multiplication)*
Step 3.	`y = 50 + 3 * 5 + 7;` `3 * 5 is 15`	*(Multiplication before addition)*
Step 4.	`y = 50 + 15 + 7;` `50 + 15 is 65`	*(Leftmost addition)*
Step 5.	`y = 65 + 7;` `65 + 7 is 72`	*(Last addition)*
Step 6.	`y = 72`	*(Last operation—place `72` in `y`)*

Fig. 2.11 | Order in which a second-degree polynomial is evaluated.


z – (a + b / 2) + w * -y
1 / a,b
3 – b
2 + a
4 * b
5 – c
6 +
z
z a b w y
8 3 9 2 –5
z – (a + b / 2) + w * –y
8 3 9 2 –5
4 5
7 10
1
11

Homework

- Write a C code that calculates the roots of the following equation:

$$ax^2 + bx + c = 0$$

- Your program should read a, b and c from standard input.

- Bring the print-out of the C code to the next lecture.

- Hint: You will need to use the sqrt() function defined in the math.h library. Google it for more detail.