

CS313E: Elements of Software Design

Introduction

Dr. Bill Young
 Department of Computer Sciences
 University of Texas at Austin

Last updated: August 25, 2011 at 15:27

CS313E is the second course in the Elements of Software series. It is assumed that you know some high level programming language (C, Java, Python, etc.) at the level taught in CS303E.

The emphasis of this course will be on software development using object-oriented methodology. You will learn how:

- to design software using the Python programming language;
- to use the extensive libraries available in Python;
- to use simple data structures and algorithms to solve problems;
- to evaluate programs for their functionality and performance.

What is a Computer

What constitutes a counting device vs. a calculator vs. a computer?

Which are these:

- a pile of stones to keep track of your cattle
- an abacus
- a slide rule
- a pocket calculator
- a programmable pocket calculator
- your laptop

Computing Machines

Building computing machines depends on:

- the theoretical foundations
- engineering ability of the society

Early tinkerers were hampered by the inability of manufacturing in their day to create precise gears and linkages.

Today we build devices with millions of transistors smaller than a human hair.

In talking about computational devices, there are several crucial distinctions.

- digital vs. analog
- hard-wired vs. programmable
- special purpose vs. general purpose
- sequential vs. parallel
- synchronous vs. asynchronous
- *any others?*

Some of these are differences in kind, others in degree.

There are two interesting aspects to the problem: *What is a computer?*

- Is there a collection of *minimum requirements* for anything we'd want to call a computer?
- Is there a *maximum set of capabilities* of any computer we could ever build?

The second of these is really equivalent to the question: *Can we describe the class of possible "computations," such that no possible computer could ever compute something outside that class?*

What are the fundamental components of a computer?

- computational unit (ALU)
- control unit
- memory/storage
- input/output

Where is the program stored?

What are the fundamental components of a computer?

Though complex, a computer can be understood in terms of its interconnected components:

Central Processing Unit: or *processor* follows a set of simple instructions and performs basic arithmetic calculations. Contains:

- **Arithmetic/Logic Unit** (ALU) that performs basic calculations;
- **Control Unit** that follows the program's instructions and gives directions to the (ALU).

Memory: stores data and instructions;

Input/Output Unit: allows the processor to communicate with its environment, such as output to the monitor and input from keyboard;

Clock: unit that gives tempo to the other components.

Any computer does the following, essentially forever:

- ① *Fetch* an instruction from the program memory;
- ② *Decode* the instruction to determine what it says;
- ③ *Execute* the instruction;
- ④ repeat the entire sequence.

One loop of this procedure takes some number of *cycles* or “ticks” of the system clock.

The clock rate measures the fundamental speed of the machine. E.g., a computer that runs at 3GHz means that the clock (oscillating crystal) pulses at 3 billion cycles per second). The fetch-execute cycle will take some integral number of clock pulse, which may vary depending on the instruction.

Various kinds of information is stored and processed by a computer:

- numbers of various kinds
- text
- pictures
- sounds
- movies
- instructions

All of this information is stored in binary form.

It is easy to build reliable electronic components that store either of two states: magnetized or not, on or off, polarized or not, left or right, up or down, etc. It is much harder to build devices that have three states or ten states.

Binary is fine for numbers. But how about letters, text, pictures, sound, etc.

Any information that can be represented can be represented in binary.

Can all information be represented? How about the number π ?

Binary is fine for numbers. But how about letters, text, pictures, sound, etc.

Any information that can be represented can be represented in binary.

Can all information be represented? How about the number π ?

You can't represent π as a finite sequence of bits. But you can represent it by the phrase: "the circumference of a circle with unit diameter" which can be encoded in ASCII.

You also might represent it as a program to compute it to arbitrary precision.

Dec	Bin	Dec	Bin
0	0	8	1000
1	1	9	1001
2	10	10	1010
3	11	11	1011
4	100	12	1100
5	101	13	1101
6	110	14	1110
7	111	15	1111

and so on. Both binary and decimal are positional, meaning that the position of each symbol within a number represents its value. E.g., the number 10011 in binary represents 19 because:

$$19 = 1 * 2^4 + 0 * 2^3 + 0 * 2^2 + 1 * 2^1 + 1 * 2^0.$$

The decimal system is also called "base 10" and the binary system "base 2." You could use any natural number as a base.

Thus, any arithmetic you can do in the decimal system, you can do in binary. When you tell a computer to add two numbers, the arithmetic is actually done by the ALU in binary and the result displayed back to you in decimal.

You can probably think of ways you *might* represent other types of numbers such as negatives, fractions, scientific notation, numbers with a decimal point, ranges, etc.

negatives	$-n$	represent sign with additional bit
fractions	n/m	pair of numbers n and m
decimal fractions	$n.m$	pair of numbers n and m
scientific notation	$n.m \times 10^e$	triple of numbers n , m , e
range of numbers	$[n \dots m]$	pair of numbers n and m

This is not necessarily how these numbers are represented in modern computing systems, but *they could be*.

Actually, integers (positive and negative) are represented in *two's complement* notation. Reals (fractions and exponentiated numbers) are typically represented as *floating point* numbers.

The representation is *arbitrary*, but is chosen to make operations simple. What about text? The following is part of the ASCII (American Standard Code for Information Interchange) representation for characters.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
32		!	"	#	\$	%	&	'	()	*	+	,	-	.	/
48	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
64	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
96	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
112	p	q	r	s	t	u	v	w	x	y	z	{	—	}		

The standard ASCII table defines 128 character codes (from 0 to 127), of which, the first 32 are control codes (non-printable), and the remaining 96 character codes are representable characters.

The order of the character encoding is called the *collating sequence*. The goal is to make it easy to sort strings by comparing numbers.

Note that the capital letters are consecutive codes (65..80) and the lower case letters are consecutive codes (97..122), with space code 32.

How do you convert a letter from upper to lower case and vice versa? How do you alphabetize two strings? Why does the space character have a code lower than the letters?

Storing ASCII

Notice that standard ASCII defines 128 (2^7) characters. Thus, only 7 bits are needed to represent any single ASCII character. ASCII characters are actually stored in an 8-bit *byte*. How is the extra bit used.

The extra bit may be used for parity (explained next slide) or for extending the ASCII character set with characters like: ÷, Å, and ≤.

The extensions are not all compatible but a common one is shown on the webpage:

<http://www.asciitable.com>

Parity

Alternatively, the last bit may be used for *parity*. If the system uses *even parity*, the bit is set to 0 or 1 to make the number of 1 bits in the byte even.

Example: if the data is 1001100, then the last bit is set to 1 and 10011001 is stored to ensure that there are an even number (4) of 1 bits.

Using this scheme, if any byte is found not to have even parity, it must have been corrupted and needs to be retransmitted.

Question: Explain an *odd parity* system.

A text *string* is simply a sequence of characters, including the space character (ASCII 32). An arbitrary English language text can be represented as a sequence of ASCII codes, stored in successive bytes in memory.

Line endings are represented differently in different systems. Early *teletype* printers used the CR and LF control characters, respectively, to move the print head left and then scroll down one line.

- The Macintosh uses a CR (ASCII 13)
- Linux uses a LF (ASCII 10)
- Windows uses both.

Most FTP (file transfer) programs substitute appropriate line endings if transferring in “text mode.” If in “binary mode” no translations occur.

Recall that the machine really runs by the *fetch-execute cycle*. Fetch an instruction from memory, decode it, execute it, then do it again.

But those instructions are in *machine language*, which is represented as binary numbers stored in a block of memory locations. This is *really* hard to program in.

So programmers have designed *high-level programming languages* to make the task of programming simpler. Python is such a language.

What Do We Need?

What do we need to have a usable programming language? That's debatable, but here are some useful aspects:

- Program structure
- Character set
- Rules for defining variables
- Types
- Operators - Arithmetic, Conditionals, Boolean
- Conditionals: if-else, switch
- Loops: while, do-while, for
- Methods
- Arrays
- Simple input and output

Program Structure: Hello World

Traditionally, the first program you write in any new programming language is the Hello World program. Here it is in **Java**:

```
public class HelloWorld {

    public static void main(String[] args) {

        System.out.println("Hello World");

    } // main

} // class HelloWorld
```

This must be in a file called **HelloWorld.java**. Why?

Here's what I did to compile and run the program:

```
> javac HelloWorld.java
> java HelloWorld
Hello World
```

The command **javac** runs the Java compiler.

Now here is the Hello World program in Python:

```
print ("Hello World")
```

Why so much simpler?

How to Run

Here's how you actually run the Hello World program:

```
felix:~/cs313e/slides> python
Python 3.1.2 (r312:79147, Sep 27 2010, 09:45:41)
[GCC 4.4.3] on linux2
Type "help", "copyright", "credits" or "license" for more :
>>> print ("Hello World")
Hello World
```