Chapter 101

Coding & Computation

101.0.1

Coding & Computation\textsuperscript{1}
or
What’s all this about 1s and 0s?

101.0.2

Topics

Aim: Communicate certain fundamental concepts about computers and computing. Relax.

• Digital encoding
• (Digital) computation
• Fundamental programming
• Applied programming

\textsuperscript{1}File: coding-computation-slides.tex.
101.0.3

Digital encoding

- Digital $\approx$ discrete
  (Analog $\approx$ continuous)
- Encode $=$ to put into or represent with a code
- Code $\approx$ “a system of symbols”
  ...used for communication (Morse code, braille, “one if by land,...”),
  for instructing computers (machine code), &c.
  Code are discrete: elements are in or out. Period.

101.0.4

Digital encoding: Examples of codes

- Written words: “Bob,” “Carol,” “Ted,” “Alice”
  Words “stand for” things, events, relations, etc. Make up sentences, etc.
- Letters of the alphabet
  What an idea! With a few letters, all the words; with a few thousand
  words, an infinite number of sentences.
101.0.5
Digital encoding: Examples of codes

- Morse code: Dots: ■ ■ Dashes: ■■■■

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>■ ■</td>
<td>J</td>
<td>■■■■■■■■</td>
<td>S</td>
<td>■ ■ ■</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>■■■■■</td>
<td>K</td>
<td>■■■■■</td>
<td>T</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>■■■■■■■</td>
<td>L</td>
<td>■■■■■</td>
<td>U</td>
<td>■■■■■</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>■■■■■</td>
<td>M</td>
<td>■■■■■</td>
<td>V</td>
<td>■■■■■</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>■■■■■</td>
<td>N</td>
<td>■■■■■</td>
<td>W</td>
<td>■■■■■</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>■■■■■■■</td>
<td>O</td>
<td>■■■■■■■■</td>
<td>X</td>
<td>■■■■■■■■</td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>■■■■■■■■</td>
<td>P</td>
<td>■■■■■■■■</td>
<td>Y</td>
<td>■■■■■■■■</td>
<td></td>
</tr>
<tr>
<td>H</td>
<td>■■■■■■■■</td>
<td>Q</td>
<td>■■■■■■■■</td>
<td>Z</td>
<td>■■■■■■■■</td>
<td></td>
</tr>
<tr>
<td>I</td>
<td>■■■■■</td>
<td>R</td>
<td>■■■■■■■■</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(There’s more for other symbols, e.g. Ā is ■■■■■■■■)

101.0.6
Digital encoding: Comments on Morse code

- How many elementary / primitive symbols? Answer: 2. Why?

- How many symbols do we need to cover the letters of the 26 letters of the alphabet?
  Answer: $2^1 + 2^2 + \ldots + 2^n$ until $> 26$, i.e., $n = 4$.

- Note: Why not just $2^5$?

- Why do the different letters have the symbol patterns they have?
CHAPTER 101. CODING & COMPUTATION

101.0.7
Digital encoding: Examples of codes

- Braille: array of $3 \times 2 = 6$ dots, raised or not.
- Character codes in computing
  2. EBCDIC: IBM mainframes
  3. Unicode: For all the world.

101.0.8
ASCII

- Binary code: 1s and 0s. 7 or 8 bits (= binary digits)
- P is decimal 80 = $8 \times 10^1 + 0 \times 10^0$
- or in binary = $0 \times 2^7 + 1 \times 2^6 + 0 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 0 \times 2^0$
- or 01010000 in 8-bit binary. 01010000 in 7-bit binary, with parity even. 11010000 in 7-bit binary, with parity odd.
**101.0.9**

Number coding systems

- Decimal (10-based) has 10 symbols possible per ‘slot’: 0, 1, 2, . . . , 9
- Binary (2-based) has 2 symbols possible per ‘slot’: 0 and 1
- Octal (8-based) has 8 symbols: 0, 1, 2, 3, 4, 5, 6, and 7
- Hexadecimal (16-based) has 16 symbols: 0, 1, . . . , 9, A, B, C, D, E, and F

Number coding can be in any base: 2, 3, . . . , n. Why base 2 for computers? Why base 8, base 16?

---

**101.0.10**

Digital encoding.

- We encode from a list of atomic symbols (e.g., the alphabet) and compose more complex things by combining these symbols (e.g., words are composed of letters of the alphabet, sentences are composed of words).

- At the most abstract, general level, we can use numbers to be our atomic symbols (numerals, actually).

- So, e.g., P = 80 decimal = 01010000 binary in ASCII.

- Let dash (-----) = 0, dot (••••) = 1. Then in Morse code, P = 1001 binary
101.0.11

Digital encoding: Comments

- Conventionality (arbitrariness)
  Why not P = 1010111? etc.

- Generality
  Can one type of encoding encode everything that another type of en-
  coding can encode? Does it matter whether we do decimal or binary?
  Why? (Prove it!)

- Wait: 01010000 is a (binary) number, yet it’s an encoding of P. Which
  is it? How can you tell? Why isn’t it AH in Morse code? Or a sound
  or a picture or a movie?

101.0.12

Digital computation

- ...or a program instruction?

- Roughly, a computation is a manipulation and/or recognition of a dig-
  ital encoding

- A computer is a machine that does computations, that manipulates,
  recognizes, and acts on digital encodings

- Our computers work on binary (1s and 0s) encodings. Why?
Is that all? Can’t some computers do more than others?

Yes, that’s possible. Size, speed, of course.

Actually, just a few instructions are sufficient to compute all possible manipulations on binary digital encodings, and these are in turn fully general.

An amazing fact.

So, all real computers are fundamentally equivalent, just some are bigger and faster than others.

What about interacting with the world? ‘I/O’ as we say.

Same thing, just hooked to I/O devices.

Clarification: manipulations aren’t just arithmetic; they’re anything (on binary encodings).
101.0.15
Fundamental programming

- Computers run by executing program instructions, one after another.
- The program instructions instruct the computer to manipulate, recognize, and act upon digital (binary) encodings.
- How are program instructions represented to the computer? As binary encodings. What about the data used by the program? Same thing. How does it know?
- Basic cycle: (1) fetch the next instruction (from memory into the CPU), (2) execute the instruction, (3) figure out where the next instruction is and go to (1).

101.0.16
Applied programming

- Don’t like them 1s and 0s (machine language)
- So, ‘higher-level’ languages: metaphor
- Compiler: takes your ‘higher-level’ jottings and translates them into machine language, so your program can be executed.
- Note: machine language programs are specific to the machine type you are running: Intel X, Intel Y, Macintosh, Sun, Alpha, IBM, etc.
101.0.17

Applied programming

• Interpreter: Accepts a compact ‘semi-compiled’ (byte-code) version of your jottings and executes it by translating it on the fly to machine code and sending it off for execution. Visual Basic, Excel.

• Possibility: Interpreters for each type of machine (hardware), but all can then execute the same byte-code. “Write once, run everywhere.”

• Think of the Internet.

• And: Java.

101.0.18

Applied programming: On the Internet, etc.

Where does/can code execute?

• On your PC (your client, whether Wintel, Mac, Linux, Unix, . . .): Your browser (Internet Explorer, Netscape, Hot Metal, . . .)

• On your PC: Spreadsheet programs, word processing programs, . . .
101.0.19
Where does code execute? (con’t.)

- On the server: The Web server program that responds to requests from your browser and serves up files to you.
- On the server: Business programs that run in response to your inputs from your browser: shopping carts, billing, etc. (Think of buying something on the Web, or participating in an auction.)
- On the server: Business programs that create HTML pages and send them to you on the fly. PHP, ASP, JSP.

101.0.20
Where does code execute? (con’t.)

- On your PC, via your browser: JavaScript, VBScript for graphics and animation (but primitively)
- On your PC, via your browser: ActiveX (Microsoft) and Java applets Downloaded in real time from the server! Why? Pluses? Minuses? Worries?

101.1 Bibliographic Note