CS101: Fundamentals of Computer Programming

Dr. Tejada
stejada@usc.edu
www-bcf.usc.edu/~stejada
Week 1
An Overview of Computer Science
What is Computer Science?

“Computer science is no more about computers than astronomy is about telescopes”

- Edsger Dijkstra
Areas in Computer Science

From wikipedia.org/Computer_Science
Scientific Problem Areas

Bioinformatics  Cognitive Science  Computational chemistry  Computational neuroscience

Computational physics  Numerical algorithms  Symbolic mathematics

From wikipedia.org/Computer_Science
History of Computing:

Counting Devices

- Prehistoric times
  - Marks on sticks or bones
- 2600 B.C.
  - Abacus invented
- ~ 1600 A.D.
  - First mechanical calculator (add and subtract)
Programmable Machines

- **Jacquard’s Loom (1801)**
  - Programmable woven designs
  - Use of punch cards as “instructions”
  - Incited labor riots in 1811 by people afraid of losing their jobs to the machine.
Mechanical Computers

- Primarily gear-based
- Difference Engine and Analytic Engine designed and partially implemented by Charles Babbage
  - Used mechanical levers, gears, and ball bearings, etc.
- Difference Engine (1823)
  - Prototype and not fully programmable
- Analytic Engine (1834)
  - Never completed
  - To be programmed with punch cards
  - Designed to perform 4 basic arithmetic ops. (add, sub, mul, div)

Charles Babbage and his Difference Engine
First Programmer

- Ada Lovelace
  - Mathematician and writer
  - Worked on Analytic Engine by Charles Babbage
  - Her notes are the first algorithm intended to be processed by a machine
Electronic Computers

- 1945 - ENIAC was first, fully electronic computer
- Used ~19,000 vacuum tubes as the fundamental switching (on/off) technology
- Weighed 30 tons, required 15,000 square feet, and maximum size number was 10 decimal digits (i.e. ±9,999,999,999)
Vacuum Tube Technology

- Digital, electronic computers use some sort of voltage controlled switch (on/off)
- Looks like a light bulb
- Usually 3 nodes
  - 1 node serves as the switch value allowing current to flow between the other 2 nodes (on) or preventing current flow between the other 2 nodes (off)
  - Example: if the switch input voltage is 5V, then current is allowed to flow between the other nodes

Switch Input (Hi or Lo Voltage) → Current can flow based on voltage of input switch
Vacuum Tube Disadvantages

- Relatively large
  - Especially when you need 19,000 to make 1 computer
- Unreliable
  - Can burn out just like a light bulb
- Dissipate a lot of heat (power)
Transistor

- Another switching device
- Invented by Bell Labs in 1948
- Uses semiconductor materials (silicon)
- Much smaller, faster, more reliable (doesn’t burn out), and dissipated less power

Individual Transistors
(About the size of your fingertip)
Integrated Circuits

Problem: Building a computer still meant wiring more than 19,000 individual transistors

Solution: Integrated Circuits

Many transistors can be built and wired on a single, contiguous piece of silicon (called an integrated chip)

Actual silicon wafer is quite small but can contain ~300 million transistors

Silicon wafer is then packaged to form the chips we are familiar with

Multiple Transistors wired together on an IC
Moore’s Law

Moore’s Law (~1960): # of transistors on a chip will double every 1.5 – 2 years
- Exponential increase in speed (clock frequency) and # of transistors
- In 1960’s => SSI (Small Scale Integrated Circuits)
  - 100’s of transistors
- Present => VLSI (Very Large Scale Integrated Circuits) and now ULSI (Ultra Large Scale Integrated Circuits)
  - 100’s of millions and even a billion transistors
What is a Modern Computer?

What are defining features of modern computers?

- **Processor**: Manipulates data using basic arithmetic and logical instructions.
- **Programmable**: Executes a user-defined set of instructions on the information (data).
- **Electronic**: Deals with information represented as voltages.
Hardware Components

- Computer hardware can be classified into three categories
  - Input/Output Devices
    - Supplies and consumes data
    - Supplies the program
    - Keyboard, Mouse, Monitor, Hard Drive
  - Memory (RAM/ROM)
    - Temporary storage for data and program
  - Processor (CPU)
    - Performs operations on data as indicated by SW program
    - Pentium, Celeron, etc.
Memory

- Set of cells that each store a group of bits (usually, 1 byte = 8 bits)
- Unique address assigned to each cell
- Used to reference the value in that location

<table>
<thead>
<tr>
<th>Address</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>11010010</td>
</tr>
<tr>
<td>1</td>
<td>01001011</td>
</tr>
<tr>
<td>2</td>
<td>10010000</td>
</tr>
<tr>
<td>3</td>
<td>11110100</td>
</tr>
<tr>
<td>4</td>
<td>01101000</td>
</tr>
<tr>
<td>5</td>
<td>11010001</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>1023</td>
<td>00001011</td>
</tr>
</tbody>
</table>

Memory Device
Why 1’s and 0’s

- Transistors are electronic devices used to build computer hardware
  - Like a switch (2 positions)
  - Conducting / Non-conducting
  - Output voltage of a transistor will either be high or low
- 1’s and 0’s are arbitrary symbols representing high and low voltage outputs.
- 2 states of the transistor lead to only 2 values in computer hardware.

The voltage here determines if current can flow between drain and source.

Circuit Diagram of a Switch
- Circuit is open (off) – no current can flow
- Circuit is closed (on) – current can flow

Schematic Symbol of a Transistor

Functional View of a Transistor as a Switch

Circuit Diagram of a Switch

High Voltage
- +5V
- +12V

or

Low Voltage
- 0V
- -12V
Computer Memory

- **Byte:**
  - A sequence of eight bits

- **Kilobyte (KB):** $2^{10}$ bytes = 1024 bytes

- **ASCII (American Standard Code for Information Interchange):**
  - 128 characters
  - A is encoded as 1000001 (66th character)
  - 3 is encoded as 0110011
<table>
<thead>
<tr>
<th>Unit</th>
<th>Symbol</th>
<th>Bits/Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Byte</td>
<td></td>
<td>8 bits</td>
</tr>
<tr>
<td>Kilobyte</td>
<td>KB</td>
<td>$2^{10}$ bytes = 1024 bytes</td>
</tr>
<tr>
<td>Megabyte</td>
<td>MB</td>
<td>$1024 \text{ KB} = 2^{10} \text{ KB} = 2^{20}$ bytes = 1,048,576 bytes</td>
</tr>
<tr>
<td>Gigabyte</td>
<td>GB</td>
<td>$1024 \text{ MB} = 2^{10} \text{ MB} = 2^{30}$ bytes = 1,073,741,824 bytes</td>
</tr>
<tr>
<td>Terabyte</td>
<td>TB</td>
<td>$1024 \text{ GB} = 2^{10} \text{ GB} = 2^{40}$ bytes = 1,099,511,627,776 bytes</td>
</tr>
<tr>
<td>Petabyte</td>
<td>PB</td>
<td>$1024 \text{ TB} = 2^{10} \text{ TB} = 2^{50}$ bytes = 1,125,899,906,842,624 bytes</td>
</tr>
<tr>
<td>Exabyte</td>
<td>EB</td>
<td>$1024 \text{ PB} = 2^{10} \text{ PB} = 2^{60}$ bytes = 1,152,921,504,606,846,976 bytes</td>
</tr>
<tr>
<td>Zettabyte</td>
<td>ZB</td>
<td>$1024 \text{ EB} = 2^{10} \text{ EB} = 2^{70}$ bytes = 1,180,591,620,717,411,303,424 bytes</td>
</tr>
</tbody>
</table>

**Table 1-1 Binary Units**
Main Memory

- **Random access memory**
  - Directly connected to the CPU
- All programs must be loaded into main memory before they can be executed
- All data must be brought into main memory before it can be manipulated
- When computer power is turned off, everything in main memory is lost
Secondary Storage

- **Secondary storage**: device that stores information permanently
- **Examples of secondary storage:**
  - Hard disks
  - Flash drives
  - Floppy disks
  - Zip disks
  - CD-ROMs
  - Tapes
Input/Output Devices

- **Input devices** feed data and programs into computers
  - Keyboard
  - Mouse
  - Secondary storage

- **Output devices** display results
  - Monitor
  - Printer
  - Secondary storage
Software

- Without software, the computer is useless
- Software is developed with programming languages
  - C++ is a programming language
- **Software**: programs that do specific tasks
- **System programs** control the computer
  - **Operating system** monitors the overall activity of the computer and provides services such as:
    - Memory management, Input/output activities, Storage management
- **Application programs** perform a specific task
  - Word processors, Spreadsheets, Games, etc.....
Machine Language

- Early computers were programmed in machine language
- To calculate wages = rate * hours in machine language:
  - 100100 010001    //Load
  - 100110 010010    //Multiply
  - 100010 010011    //Store
The Evolution of Programming Languages

- Assembly language instructions are mnemonic
- Assembler: translates a program written in assembly language into machine language

<table>
<thead>
<tr>
<th>Assembly Language</th>
<th>Machine Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOAD</td>
<td>100100</td>
</tr>
<tr>
<td>STOR</td>
<td>100010</td>
</tr>
<tr>
<td>MULT</td>
<td>100110</td>
</tr>
<tr>
<td>ADD</td>
<td>100101</td>
</tr>
<tr>
<td>SUB</td>
<td>100011</td>
</tr>
</tbody>
</table>

**TABLE 1-2** Examples of Instructions in Assembly Language and Machine Language
The Evolution of Programming Languages

- **High-level languages** include Basic, FORTRAN, COBOL, Pascal, C, C++, C#, Scheme and Java
- **Compiler**: translates a program written in a high-level language into machine language
- The equation \( \text{wages} = \text{rate} \times \text{hours} \) can be written in C++ as:

\[
\text{wages} = \text{rate} \times \text{hours};
\]
Processing a C++ Program

```cpp
#include <iostream>
using namespace std;
int main()
{
    cout << "Hello, World" << endl;
    return 0;
}

Sample Run:
Hello, World
```
Processing a C++ Program

**FIGURE 1-3** Processing a C++ program

- **Step 1:** Editor
- **Step 2:** Preprocessor
- **Step 3:** Compiler with Syntax Error
- **Step 4:** Library
- **Step 5:** Linker
- **Step 6:** Loader
- **Execution**
Software Development Cycle

- **Algorithm:**
  - Step-by-step problem-solving process
  - Solution achieved in finite amount of time

- Programming is a process of problem solving
The Problem Analysis–Coding–Execution Cycle

- **Step 1: Analyze the problem**
  - Outline the problem and its requirements
  - Design steps (algorithm) to solve the problem

- **Step 2: Implement the algorithm**
  - Implement the algorithm in code
  - Verify that the algorithm works

- **Step 3: Maintenance**
  - Use and modify the program if the problem domain changes
The Art of Problem Solving

- It has been proven that the process of problem solving (or algorithm development) is not an algorithmic process.
- In 1945, George Polya (a mathematician), wrote a famous book, *How to Solve It.* Identified four principles for solving problems:
  - 1. Understand the Problem
  - 2. Devise a Plan
  - 3. Carry out the Plan
  - 4. Look Back
Understanding the Problem

- To ensure that you understand a problem, you can ask yourself the following questions:
  - Do you understand all the words used in stating the problem?
  - What are you asked to find or show?
  - Can you restate the problem in your own words?
  - Can you think of a picture or a diagram that might help you understand the problem?
  - Is there enough information to enable you to find a solution?
  - Does program require user interaction?
  - Does program manipulate data?
  - What is the output?
Devise a Plan

Polya mentions that there are many reasonable ways to solve problems. The skill at choosing an appropriate strategy is best learned by solving many problems. A partial list of strategies is below:

<table>
<thead>
<tr>
<th>Guess and check</th>
<th>Make an orderly list</th>
<th>Eliminate possibilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use symmetry</td>
<td>Consider special cases</td>
<td>Use direct reasoning</td>
</tr>
<tr>
<td>Solve an equation</td>
<td>Look for a pattern</td>
<td>Draw a picture</td>
</tr>
<tr>
<td>Solve a simpler problem</td>
<td>Use a model</td>
<td>Work backward</td>
</tr>
<tr>
<td>Use a formula</td>
<td>Be ingenious</td>
<td></td>
</tr>
</tbody>
</table>
Carry Out the Plan

- This step is usually easier than devising the plan.
- In general, all you need is care and patience, given that you have the necessary skills.
- Keep trying with the plan that you have chosen.
- Check the correctness of algorithm:
  - Can test using sample data
  - Some mathematical analysis might be required
- If it continues not to work discard it and choose another.
Look Back

- Much can be gained by taking the time to reflect and look back at what you have done, what worked and what didn't.
- Doing this will enable you to predict what strategy to use to solve future problems.
- It may also help you optimize your solution.
Algorithm Development

- Problem: Design an algorithm to find the perimeter and area of a rectangle
- The perimeter and area of the rectangle are given by the following formulas:

  \[
  \text{perimeter} = 2 \times (\text{length} + \text{width}) \\
  \text{area} = \text{length} \times \text{width}
  \]
Algorithm Development

- Algorithm:
  - Get length of the rectangle
  - Get width of the rectangle
  - Find the perimeter using the following equation:
    \[ \text{perimeter} = 2 \times (\text{length} + \text{width}) \]
  - Find the area using the following equation:
    \[ \text{area} = \text{length} \times \text{width} \]
Algorithm Development

- Problem: Calculate students’ grades and class average
  - 10 students in a class; each student has taken five tests; each test is worth 100 points
- Design algorithms to:
  - Calculate the grade for each student and class average
  - Find the average test score
  - Determine the grade
- Data: students’ names; test scores
Algorithm

- Algorithm to determine the average test score:
  - Get the five test scores
  - Add the five test scores
    - Suppose \textit{sum} stands for the sum of the test scores
  - Suppose \textit{average} stands for the average test score:
    - \textit{average} = \textit{sum} / 5;
Algorithm

- **Algorithm to determine the grade:**
  
  ```
  if average is greater than or equal to 90
      grade = A
  else
      if average is greater than or equal to 80 and less than 90
          grade = B
      else
          if average is greater than or equal to 70 and less than 80
              grade = C
          else
              if average is greater than or equal to 60 and less than 70
                  grade = D
              else
                  grade = F
  ```
Algorithm

- Main algorithm is as follows:
  - totalAverage = 0;
  - Repeat the following for each student:
    - Get student’s data (name and test scores)
    - Use the algorithm to find the average test score
    - Use the algorithm to find the grade
    - Update totalAverage by adding current student’s average test score
  - Determine the class average as follows:
    - classAverage = totalAverage / 10
Here's an algorithm – follow exactly!!

1. Draw a diagonal line
2. Draw another diagonal line connected to the top of the first one
3. Draw a straight line from the point where the diagonal lines meet
4. Draw a horizontal line over the straight line
5. At the bottom of the straight line, draw a curvy line
6. Draw a diagonal line from the bottom of the first diagonal to the straight line
7. Draw a diagonal line from the bottom of the second diagonal to the straight line ...
How did the pictures turn out?

- Compare your picture with others' pictures...
  - Were they different?
  - Why?
  - What was difficult about following the instructions?
  - What was missing from the instructions?
It was meant to be a kite!!

1. Draw a diagonal line
2. Draw another diagonal line connected to the top of the first one
3. Draw a straight line from the point where the diagonal lines meet
4. Draw a horizontal line over the straight line
5. At the bottom of the straight line, draw a curvy line
6. Draw a diagonal line from the bottom of the first diagonal to the straight line
7. Draw a diagonal line from the bottom of the second diagonal to the straight line

...
It was meant to be a kite!!

- Now write a set of instructions that work!
- Ensure only one way to interpret each step
  - unambiguous
- ... and enough detail in each step
Putting all this together...

- This time:
  - write an Algorithm
  - test it yourself
  - get someone else to try it out...

- Can you be sure your algorithm will work ok?
(a) Write & test your algorithm

- The task/problem:
  - make a shape out of paper

- Write the algorithm
  - Write a set of instructions that explains how to make a paper shape from 1 sheet of paper

- Test it
  - Try out your algorithm – does it work?
  - Note: follow your instructions as closely as possible
  - Adjust the instructions if necessary
(b) Following an algorithm

- Hide your shape

- Get into pairs
  - by teaming up with someone **on the opposite side of the room**
  - move to sit together
  - *Do not* show them your paper shape – hide it!!

- Swap algorithm/instructions with your partner

- Follow your partner's instructions to create their paper shape

- Compare shapes
  - how similar is each 'pair' of shapes?
  - what advice can you give on how to improve the instructions?
What do we know about algorithms?

- What are the key characteristics of a “good” algorithm? Why are they hard to develop?
  - Must be unambiguous
  - Must be correct
  - Must be at the right level of detail

- Also, what did we learn about problems we pick?
  - too large sometimes?
10 Stacks of Coins

- You have 10 stacks with 10 coins each that look and feel identical.
- 9 of the 10 stacks contain coins that weigh 1 gram each while the 10th stack has coins that weigh 1.1 grams each.
  - Difference is undistinguishable by humans.
- You have a digital scale which measures the number of grams of the total items on the scale, accurate to 0.1 grams. You can only use the scale ONCE!
- Select a set of coins to put on the scale all at once in such a way that you can identify which stack has the heavy coins.
10 Stacks of Coins Solution

- Choose 1 coin from the 1\textsuperscript{st} stack, 2 coins from the 2\textsuperscript{nd} stack, etc. and weigh that collection
- The non-integer portion of the weight returned should indicate which stack
Algorithm Development: Fibonacci Number

- Consider the following sequence of numbers:
  - 1, 1, 2, 3, 5, 8, 13, 21, 34, ....
- Called the Fibonacci sequence
- Given the input \( n \)
  - The \( n^{th} \) number \( a_n \), \( n \geq 3 \), of this sequence is given by:
    \[
    a_n = a_{n-1} + a_{n-2}
    \]