CMSC131 Lecture Set 0: Course Introduction

Topics in this set:
1. Course information
2. Computer terminology basics
3. Tools needed for this course
Basic Info

❖ **Name:** “Object-Oriented Programming I”
❖ **Class meetings:** Lab and Lecture
❖ **Instructor:** Tom Reinhardt
❖ **4 TAs:** Ahmed Elgohary, Ujjwal Goel, Ugur Koc, Xuetong Sun
❖ **Office Hours**
  ❖ Will be Posted
  ❖ All in AVW building:
    ❖ 1112 (TA’s), 3239 (Tom Reinhardt)
What Is This Course?

- A fast-paced introduction to techniques for writing computer programs!
  - Skill Development in Programming
  - Conceptual Understanding of Programming
  - Beginning of “computer science”
- Intensive, but assumes you are starting at level 0.
- Keys to success
  - Attend all classes and lab sections
  - Start assignments early – and continue until you truly understand
  - Get help early if you are having trouble – instructor & TAs
  - Study every day
    - it doesn’t work to cram for these exams
    - ask questions as soon as you realize you are confused
  - Check announcements every day
Course Software

- **Eclipse**
  - An IDE (integrated development environment)
  - We will use it for writing Java™ programs
  - Access to Eclipse (it’s free!)
    - You can install it on your own machine: [http://www.cs.umd.edu/eclipse](http://www.cs.umd.edu/eclipse)
    - Also accessible in some labs around campus

- **CVS (Concurrent Versions System)**
  - A version-management system
  - You will use it for submitting your projects

Both of these – Demonstrations on Wednesday
Computer Organization

- **Hardware:**
  - physical parts of computer
  - examples
    - Monitor, mouse, keyboard
    - Chips, boards
    - Cables, cards
    - etc.

- **Software:**
  - non-physical (“logical”) parts of computer
  - Programs = instructions for computer to perform
How Programs Are Executed

Program “foo” initially stored in secondary storage

Program copied into main memory

CPU executes program instruction-by-instruction
Hardware Overview

❖ **CPU** = central processing unit
  ❖ Executes the "instructions" in programs

❖ **Main memory** = random-access memory = “RAM”
  ❖ Stores data that CPU accesses, including instructions
  ❖ FAST, but smaller and temporary; wiped out when computer is shut off!

❖ **Secondary memory**: Hard disks, CDs, DVDs, flash memory, etc.
  ❖ Stores data that can be loaded into main memory
  ❖ SLOWER, but larger and permanent

❖ **I/O devices**
  ❖ How you communicate with your machine
  ❖ Keyboard, monitor, mouse, speakers, etc.

❖ **Networking equipment**
  ❖ How others communicate with your machine
  ❖ Networking “cards”, cables, etc.
Main Memory

- Computer data consists of off and on pieces (often written as 0’s and 1’s)
- bit: A single cell in main memory that can hold either a 0 or 1
- byte: A sequence of 8 bits
- word: Unit of memory (size varies by computer - often a sequence of 4 bytes)
- Main memory: table of bytes indexed by “addresses”

<table>
<thead>
<tr>
<th>Address</th>
<th>Byte value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1 0 0 1 1 1 0 1</td>
</tr>
<tr>
<td>2</td>
<td>0 0 0 1 1 0 0 1</td>
</tr>
<tr>
<td>3</td>
<td>1 1 1 1 1 1 0 1</td>
</tr>
<tr>
<td>4</td>
<td>1 1 0 0 0 1 0 0</td>
</tr>
</tbody>
</table>
How Many Different Values can be stored in a...

- Bit?
  - 2
- Two bits?
  - \(4 = 2 \times 2\)
- Byte?
  - \(256 = 2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2 = 28\)
- Word?
  - \(4,294,967,296 = 2^{32}\)
Other Standard Terminology

One kilobyte is approximately one kibibyte which is approx 1000 bytes (actual 1024 bytes).

$2^{10} = 1024$

$2^{20} = 1024^2 = 1024 \times 1024 = 1,048,576$

$2^{30} = 1024^3 = 1,073,741,824$
How Are Characters, Etc., Represented?

- **Via encoding schemes**
- Example: ASCII
  - American Standard Code for Information Interchange
  - Early standard for encoding a single character in a bytes
  - In ASCII:
    - ‘A’ 01000001, ‘B’ 01000010, ‘C’ 01000011, …
    - ‘a’ 01100001, ‘b’ 01100010, ‘c’ 01100011,…
    - ’1’ 00110001, ‘2’ 00110010, ‘3’ 00110011, …
    - ‘,’ 00101100
    - etc.
Other Character Encodings

- International support?
  - Unicode
- Most common variation: UTF-8
  - Backwards compatible with ASCII

<table>
<thead>
<tr>
<th>Unicode</th>
<th>Byte1</th>
<th>Byte2</th>
<th>Byte3</th>
<th>Byte4</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>U+0000–U+007F (0 to 127)</td>
<td>0xxxxxx x</td>
<td></td>
<td></td>
<td></td>
<td>'$' U+0024</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>→ 00100100</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>→ 0x24</td>
</tr>
<tr>
<td>U+0080–U+07FF (128 to 2,047)</td>
<td>110yyyx 10xxxxx x x</td>
<td></td>
<td></td>
<td></td>
<td>'¢' U+00A2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>→ 11000010,10100010</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>→ 0xC2,0xA2</td>
</tr>
<tr>
<td>U+0800–U+FFFF (2,048 to 65,535)</td>
<td>1110yyy 10yyyyx 10xxxxx y x x</td>
<td></td>
<td></td>
<td></td>
<td>'€' U+20AC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>→ 11100010,10000010,10101100</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>→ 0xE2,0x82,0xAC</td>
</tr>
<tr>
<td>U+10000–U+10FFFF (65,536 to 1,114,111)</td>
<td>11110zz 10zzyyy 10yyyyx 10xxxxx z y x x</td>
<td></td>
<td></td>
<td></td>
<td>'⁠acji⁠' U+024B62</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>→ 11110000,10100100,10101101,10100010</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>→ 0xF0,0xA4,0xAD,0xA2</td>
</tr>
</tbody>
</table>
Software Overview

- **Operating system**: manages computer's resources; typically runs as soon as computer is turned on.
  - Typical responsibilities:
    - *Process management*
      - Determines when, how programs will run on CPU time
    - *Memory management*
      - Controls access to main memory
    - *I/O, window system, network control*
      - Performs low-level drawing, communication operations
    - *Security*
      - Manages user IDs, passwords, file protections, etc.

- **Applications**: programs users interact directly with; usually are explicitly run.
  - Examples:
    - Word processors
    - Games
    - Spreadsheets
    - Music software,
    - Etc
Programming Languages

- Used to write programs that run on computers
- Generations of programming languages
  - 1st (1GL): machine code
  - 2nd (2GL): assembly code
  - 3rd (3GL): procedural languages
  - 4th (4GL): application-specific languages
  - 5th (5GL): constraint languages
1st Generation: Machine Code

- Recall: computer data is 0’s and 1’s.
- In machine code, so are programs!
  - Program: sequence of instructions
  - Machine code: instructions consist of 0’s and 1’s
- Next slide: example machine code instruction from MIPS (= “Microprocessor without interlocked pipeline stages”) architecture
  - Popular in mid-, late 90s
  - Instructions are 4 bytes long
Example MIPS Instruction

- “Add data in addresses 1, 2, store result in address 6”:
  
  \[
  00000000010001000110000001000000
  \]

- broken into parts:

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>opcode</td>
<td>2nd address</td>
<td>shift amount</td>
<td>1st address</td>
<td>destination address</td>
</tr>
<tr>
<td>000000</td>
<td>000001</td>
<td>0010</td>
<td>00110</td>
<td>00000</td>
</tr>
</tbody>
</table>
Programming in 1GLs

Courtesy of Microsoft Encarta Encyclopedia Online. Copyright (c) Microsoft Encarta Online
2nd Generation: Assembly

- Problem with 1GLs: Who can remember those opcodes, addresses, etc. as 0’s, 1’s?
- Solution (1950s): *assembly language*
  - *mnemonics* = descriptive character strings for opcodes
  - Let programmers give descriptive names to addresses
- MIPS example revisited:
  ```
  add $1, $2, $6
  ```
  instead of
  ```
  00000000001000100011000000100000
  ```
  for “add contents of addresses 1, 2, store result in 6”
Assemblers

- Computers still only work on machine code (1GL)
- Assembly language is not machine code
- Assemblers are programs that convert assembly language to machine code (= “object code”)
3rd Generation: Procedural Languages

❖ Problems with 2GLs
  ❖ Platform dependency
    ❖ Different kinds (architectures) of computers use different instruction formats
    ❖ E.g. x86, Pentium, 68K, MIPS, SPARC, etc.
    ❖ 1GL / 2GL programs written for one kind of machine will not work on another
  ❖ Low level: programs difficult to understand

❖ Solution (1960s -- now): procedural languages
  ❖ Higher-level, “universal” constructs
  ❖ Examples: Cobol, Fortran, Algol, Pascal, C, C++, Java, C#
Compilers

- Computers can only execute machine code
- *Compilers* are programs for translating 3GL programs ("source code") into assembler / machine code