CMSC 106
Introduction to C Programming

Instructor: Jan Plane
Fall, 2010
Sections 0101 and 0102

The Course Logistics

- Course Syllabus
  - check webpage
    http://www.cs.umd.edu/class/fall2010/cmsc106
- Tips for Success
  - Attend all classes and lab sections
  - Start assignments early
  - Get help early if you are having trouble
  - Study every day
    - it doesn’t work to cram for these exams
    - ask questions as soon as you realize you are confused
    - Study Groups – but not on most projects
  - Check announcements on course web-page every day

Important things to learn:

- The C programming language:
  - Types of data and ways of storing data.
  - C language constructs used to perform calculations and manipulate data.
- Problem–solving
- Program debugging
Computer Organization

- **Hardware**: physical parts of computer
  - Monitor, mouse, keyboard
  - Chips, boards
  - Cables, cards
  - etc.
- **Software**: non-physical ("logical") parts of computer
  - Programs = instructions for computer to perform

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 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 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*: Smallest unit of addressable memory (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 1 1 0 1</td>
</tr>
<tr>
<td>2</td>
<td>0 0 1 1 0 1</td>
</tr>
<tr>
<td>3</td>
<td>1 1 1 1 0 1</td>
</tr>
<tr>
<td>4</td>
<td>1 1 0 0 1 0</td>
</tr>
</tbody>
</table>
How Many Different Values in a...

- Bit?  
  2
- Two bits?  
  4 = 2 \times 2
- Byte?  
  \[ 256 = 2 \times 2 \times 2 \times 2 \times 2 \times 2 = 2^8 \]
- Word?  
  \[ 4,294,967,296 = 2^{32} \]

How Are Characters, Etc., Represented?

Via **encoding schemes**

- Example: ASCII (American Standard Code for Information Interchange)
  - Standard for encoding character values as bytes
  - In ASCII:
    - 'A' 01000001
    - 'a' 01100001
    - ',' 00101100
    - etc.

There are other character encoding schemes also: Shift-JIS, Unicode, etc.

Other Standard Terminology

- 1 KB = 1 "kilobyte" = \(2^{10}\) bytes = 1,024 bytes
- 1 MB = 1 "megabyte" = \(2^{10}\) KB = 1,024 KB
- 1 GB = 1 "gigabyte" = \(2^{10}\) MB = 1,024 MB
Software Overview

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

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

How Programs Are Executed

- Program "foo" initially stored in secondary storage
- Program copied into main memory
- CPU executes program instruction-by-instruction

Two Levels of Software

- **System Software**
  - controls hardware
  - user can give commands
  - UNIX, Windows, OS-X

- **Applications Software**
  - does something specific to accomplish a task
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

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": 000000000000100010001000000000000000000??

```
 000000 0000000000100010001000000000000000
```
- opcode
- 1st address
- 2nd address
- shift amount
- destination address
- function specifier
Programming in 1GLs

Problem with 1GLs: Who can remember those opcodes, addresses, etc. as 0’s, 1’s?

Solution (1950s): assembly language
- Use *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 (60s -- now): procedural languages
  - Higher-level, "universal" constructs
  - Examples: Fortran, Cobol, Pascal, C, C++, Java, C#

Compilers

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

Algorithms

- An algorithm is a set of ordered steps solving a problem
  - Steps – tell what needs to be done
  - Order – tells which step gets done when
- A program implements an algorithm in a particular programming language.
- Pseudo code = used to describe an algorithm independent of a programming language
  - Enough detail to tell exactly what needs to be done
  - No detail about the specific programming language that would be used for the implementation
Software Development Process

- Understand the problem and design a solution
- type in some code
- compile it
- run it
- compare it to expected results

Programming Errors

- Types of Errors
  - Syntax Errors
    - violates languages grammar
    - compiler warns about these
    - Eclipse puts red squiggles under the offending code
  - Semantic/Logic Errors
    - program doesn't work properly
    - run-time errors = crash or hang
    - can be more subtle (harder to find)
- Debugging
  - process of finding and fixing problems
  - to minimize debugging frustration – use “unit” testing
  - write a small part, thoroughly test it, cycle back