Lecture 2: Programming

Last time:
1. Course information
2. Computer basics

Today:
1. Programming languages
2. Eclipse and CVS
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
1\textsuperscript{st} 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”:
  
  \[ \text{0000000000100010011000000100000} \]

- ???

<table>
<thead>
<tr>
<th>opcode</th>
<th>2nd address</th>
<th>shift amount</th>
<th>function specifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>000000</td>
<td>00001</td>
<td>00110</td>
<td>00000</td>
</tr>
</tbody>
</table>

- First address
- Destination address

---

Bonnie Dorr (adapted from Rance Cleaveland)
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*
  - 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, 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
Interpreters

- Another way to execute 3GL programs
  - Interpreters take source code as input
  - Interpreters execute source directly
  - Much slower than compiled programs

- Debuggers are based on interpreters
  - Debuggers support step-by-step execution of source code
  - Internal behavior of program can be closely inspected
This Course

How to write programs in procedural languages

- Language is Java
- Principles are broadly applicable
Tools for Writing Programs

- The bad old days
  - Text editor: used to create files of source code
  - Compiler: generate executables from source
  - Debugger: trace programs to locate errors
- Today: IDEs (= “integrated development environment”)
  - Text editor / compiler / debugger rolled in one
  - Examples: Eclipse, Visual Studio, etc.
Basics of Eclipse


- Eclipse is used to:
  - Create
  - Edit
  - Compile
  - Run
  - Debug

programs (for this class, Java programs).
Basics of Eclipse-speak

● **Project**: collection of related source files
  To create a program in Eclipse:
  ● Create a new project
  ● Create files in the project

● **Perspective**: framework for manipulating programs

● Important perspectives in this class:
  ● **Java**: for creating, running programs
  ● **Debug**: for tracing, removing errors in programs
  ● **CVS repository**: for interacting with assignment-submission system
Class Projects with CVS

- You will use Eclipse for Java programming in this course
- How will you:
  - obtain
  - turn in
  class projects?
- CVS (= Concurrent Versions System)
  - Tool for project-file management
  - Maintains versions, etc.
  - Allows different sites to work on same project
CVS Worldview

Files = "repository"

Files (local copies)

"checkout"

"commit"

Server

Client 1

Client 2
CVS in More Detail

- CVS server maintains current versions of files in project (= “repository”)
- To access files from another machine (“client”), repository must be “checked out”
- Changes to files on client may be “committed” to server, with changed files becoming new version
- (Once a repository is checked out by a client, subsequent versions may be accessed via “update”)
What’s Needed for CVS?

- Server machine
  
  *For CMSC 131, CS linuxlab machines*

- User authentication
  
  *For CMSC 131, student linuxlab accounts*
How CMSC Project Submission Works

- Repository created for each student linuxlab account
- You check out repository to start work on project
- When you “save” changes in Eclipse, “commit” automatically invoked by plug-ins
- You “submit” when finished using Eclipse (UMD plug-in handles relevant CVS commands)
To Checkout a Project

1. Set repository location
   - Change to “CVS Repository Exploring” perspective in Eclipse (“Window -> Open Perspective” …)
   - Right-click in “CVS Repositories” panel and select “New -> Repository Location…”
Adding a CVS Repository

Common to everyone
Your linuxlab username
Your linuxlab password
Don’t forget to set this!
To Checkout a Project (cont.)

2. Open repository name, then “Head”
3. Right-click on project name to save
Working on Project

- When you switch back to “Java” perspective, your project is now there!
- When you save in “Java” perspective, changes are automatically committed to CVS repository.
Study Questions

- Login: study
- Password: daily