CS 1023 Intro to Engineering Computing 3:
Computer Programming
LM1 – Simple C Programming

Mark Kerstetter
Department of Computer Science
Western Michigan University

© Spring 2012
## Computing Tools
### Spreadsheets vs. Mathematical Software vs. Programming

<table>
<thead>
<tr>
<th>Spreadsheets</th>
<th>Math Software</th>
<th>Programming</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>most widely available</strong></td>
<td>restricted availability</td>
<td>moderate availability</td>
</tr>
<tr>
<td><strong>widest range of applications</strong></td>
<td>applications restricted to engineering, math, &amp; science</td>
<td>unique applications or applications that cross boundaries</td>
</tr>
<tr>
<td><strong>college-level problem solving abilities</strong></td>
<td>knowledge of math notation</td>
<td>requires patience and logical skills</td>
</tr>
<tr>
<td><strong>rapid learning curve</strong></td>
<td>slower learning curve</td>
<td>slowest learning curve</td>
</tr>
<tr>
<td><strong>high-level abstraction</strong></td>
<td>moderate abstraction</td>
<td>low-level details</td>
</tr>
<tr>
<td><strong>least complex requires little set up</strong></td>
<td>more complex requires set up &amp; multiple windows</td>
<td>most complex requires set up, multiple windows &amp; separate steps</td>
</tr>
<tr>
<td><strong>interpreted execution</strong></td>
<td>interpreted execution</td>
<td>compiled execution</td>
</tr>
<tr>
<td><strong>instant</strong> intermediate results</td>
<td><strong>instant</strong> intermediate results</td>
<td><strong>must wait</strong> for final results</td>
</tr>
<tr>
<td><strong>solves certain kinds of problems</strong></td>
<td>solves restricted kinds of problems</td>
<td>solves all kinds of problems</td>
</tr>
</tbody>
</table>
Why Programming is Tough!

The Problem

Link with Libraries

Runtime

Debug Program

Tools

IDE

Learn Programming

Test Program

Make It

Fail

Compile Program

Sketch Solution

Learn Language Semantics

Run Program

Load Program

Learn Language Syntax

Write Program

Learn

Debug Program

The Problem

Link with Libraries

Runtime

Debug Program

Tools

IDE

Learn Programming

Test Program

Make It

Fail

Compile Program

Sketch Solution

Learn Language Semantics

Run Program

Load Program

Learn Language Syntax

Write Program

Learn
Why do engineers have to learn how to program?

- **Computer programming teaches you ...**
  - to think precisely
  - patience
  - humility
    - anticipate errors, failures, etc.
    - identify/locate them
    - fix/correct them
  - practicality
  - problem solving
    - to analyze problems
    - to synthesize/design solutions

  Engineers are careful, systematic, & very structured people.
  Engineers cannot afford to be hasty. They need to explore alternatives.
  Engineers are human, they make mistakes. They work hard to eliminate mistakes. They are ultimately responsible for mistakes remaining in the systems they design.

  Engineers solve real-world problems.
  Scientist solve theoretical problems.

  Engineers are all about design, Design, DESIGN!!!

- **We don’t expect you to become expert programmers!**
  - Expect you to read and understand macros (smaller programs) in spreadsheets and in mathematical software
  - You may probably write or change macros in other applications
  - You may write one-time use data conversion programs
  - You probably will not write programs from scratch
Learning to Program

Problem Solving Process Using Computer Programming

- Understand the Problem
- Design an Algorithm & Organize Data
- Write the Source Code
- Compile the Source Code
  - Remove syntax errors
- Link and Load the Object Code
  - Remove link-load errors
- Execute the Program
- Debug the Program
  - Remove run-time and logic errors
- Check the Program Activity/Output vs. the Problem Requirements

If you don’t understand the problem, you can’t solve it.

Know your programming language so you can use its tools, data types, and data structuring abilities.

Follow language syntax rules. Use simple text editor or a special editor designed to help write source code.

Step often hidden to programmer.

Make sure $2 + 2 \neq 5$ !!!

Engineers are interested in both product & process!
Most Important Advice

- Always start with a **small** C program that **works**!!!
  - Hello World Program
  - Hello World Program that inputs data
- Add **new instructions in very small increments**!!!
- Do not proceed until the previous program step **works**!!!
- Find prototypes...reuse code (cut and paste and change)
Program Compilation/Link-Load/Execution

C Language Program
(source code)

Compile

Link/
Load

Execute

Machine Language
Program
(object code)

Program Input

Program Output

Each box represents a system program that executes in your behalf.

Each arrow represents input or output within the process.
Learning to Program

- Can’t learn it all from a book. Must practice, practice, practice!
- Learn to read and understand first. Write your own programs second.
- Learn to use simple instructions first. Learn more complicated ones later.
- Problem Solving
  - Think Top Down – Break the problem into parts. Solve each part.
  - Remember the handful of instructions & data types you will use.
- Know when to ask for help.
- Random Trial-and-Error vs. Controlled Experimentation
  - Work from examples.
  - Modify in increments.
  - Think small. Think simple. There’s less to remember.
  - Work in pairs or small groups.
  - Random Trial-and-Error wastes time and is very frustrating.
  - Controlled Experiments confirms what you know and adds to your knowledge and self-confidence.
Why is Programming Difficult?

- Needed – a disciplined, highly focused, and logical mind!
- Lots of “picky” little rules.
- Various kinds of mistakes
  - syntax errors – found by compiler
  - link/load errors – found by link-loader
  - run-time errors – found by the software system (run-time programs, OS, etc.)
  - logic errors – obvious and non-obvious, found by programmer/tester
- Takes development time before seeing results.
  Not instant gratification! Not fancy GUI immediately!
- Frustration! Frustration! Frustration!
Honest Disclosure about Programming

- Programming is very rewarding, but very frustrating.
- What you learn and do in this course will not be as satisfying and enjoyable as solving problems with a spreadsheet or with mathematical software.
  - Programming in CS1023: teething ring
    :: Spreadsheets in CS1021: Leapfrog toy
  - Programming in CS1023: teething ring
    :: Mathematical Software in CS1022: Leapfrog toy
- However, both spreadsheets applications and mathematical software ARE computer programs!
- Learn to appreciate the complexity of computer programs
- Learn logic, problem solving, patience, & humility
Asking Questions
Expectations for Help

■ Don’t waste time trying to locate or fix an error or to find a solution or part of a solution.

1. Do prepare BEFORE tackling an error or solution. You must try!
   • Read the textbook and any notes.
   • Find a second or third source if you can’t understand the textbook or notes.
   • Write your own notes about the error or solution,
     e.g., what you know, what you don’t know, your ideas, etc.

2. Using your preparation try to eliminate the error or implement the solution or partial solution.

3. Iterate Step 1 and Step 2 several times.

4. Ask for help – roommate, classmate, TA, instructor, etc.
   1. Expect to explain what you have done and what you think.
Components of a Programming Language

- **Constants**
- **Variables**
  - Scalar
  - Structured
- **Expressions**
- **Statements**
  - simple statements
  - grouping of statements
  - flow control statements

**Analogy #1**
Building a house or a piece of furniture.
- materials used in constructing a building
- small useful components; e.g., door hinges
- tools for getting the construction done; hand tools, power tools, & specialized tools

**Analogy #2**
Writing an essay for an English course.
- vocabulary/words in a natural language
- phrases in a natural language; several words
- complete sentences in a natural language; made up of words and phrases
C Program Components

Goal: Be able to recognize the following in a program on a quiz.

- **Keywords – Cannot be changed**
  - int, float, double, const, if, else, while, do, return, …

- **Variables – Programmer-Defined**
  - values can change during execution
  - variables are named storage locations, e.g., index, x, distance, y1, lastName, ID_Nbr
  - What are name forming rules?

- **Constants**

- **Symbolic Constants – Programmer-Defined**
  - values cannot change during execution
  - symbolic constants have same name forming rules as variables
  - symbolic constant naming conventions are to use all upper case, e.g., PI, SIZE

- **Function Names**
  - built-in names should NOT be changed – printf, scanf, sqrt, min, etc.
  - programmer-defined function names – please wait until Chapter 4

Try to recognize keywords, variables, symbolic constants, constants and functions in the demo program.
C Program Components

Goal: Be able to recognize the following in a program on a quiz.

- **Comments – Info for Human Program Readers**
  - /* comment goes here */ -or- // comment goes here

- **Preprocessor Directives**
  - #include <header file> make sure supporting info is available

- **Statements (Program Instructions)**
  - Statements normally end with a semicolon ( ; )
  - Declaration Statements – organize data
    - int, short, long, float, double, long double, char
  - Executable Statements – control instruction flow, what to do next
    - assignment statements – sum = a + b + c ;
    - function call statements – printf(…) ;
    - return statements – return number ;

Try to recognize comments, preprocessor directives, declarations, assignment statements, function calls in the demo program.

We’ll add more statements as you learn more programming.
I/O Control String Specifiers

Goal: Be able to recognize the following in a program on a quiz.

- control strings – manage input and output formatting
- input – scanf function
  - Get data from keyboard or from a file
  - E.g., scanf ("%spec", &varName);
- output – printf function
  - Put data on a display screen or into a file
  - E.g., printf ("control string info %spec %spec %spec \n", varName, expression, constant);

- %Specifiers
  - int: %i or %d | long: %li or %ld
  - float: %f, %e, %g | double: %lf, %le, %lg
    - Can specify size of numbers input, displayed, or printed
e.g., %5i or %8.2f

- Escape Sequences
  - \n (new line), \t (horizontal tab), \a (sound alert)

Try to recognize control strings, input function calls, output function calls, and I/O specifiers in the demo program.
# Arithmetic Operator Precedence

<table>
<thead>
<tr>
<th>Precedence</th>
<th>Operator</th>
<th>Associativity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>parentheses</td>
<td>innermost first</td>
</tr>
<tr>
<td></td>
<td>( )</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>unary ops</td>
<td>right-to-left</td>
</tr>
<tr>
<td></td>
<td>+ -</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>multiplier binary ops</td>
<td>left-to-right</td>
</tr>
<tr>
<td></td>
<td>* / %</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>additive binary ops</td>
<td>left-to-right</td>
</tr>
<tr>
<td></td>
<td>+ -</td>
<td></td>
</tr>
</tbody>
</table>
## Arithmetic & Assignment Operator Precedence

<table>
<thead>
<tr>
<th>Precedence</th>
<th>Operator</th>
<th>Associativity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>parentheses (  )</td>
<td>innermost first</td>
</tr>
<tr>
<td>2</td>
<td>unary ops + - ++ - -</td>
<td>right-to-left</td>
</tr>
<tr>
<td>3</td>
<td>multiplier binary ops * / %</td>
<td>left-to-right</td>
</tr>
<tr>
<td>4</td>
<td>additive binary ops + -</td>
<td>left-to-right</td>
</tr>
<tr>
<td>5</td>
<td>assignment ops = += -= *= /= =</td>
<td>right-to-left</td>
</tr>
</tbody>
</table>