CSCI 1301 Book

May 18, 2021 (04:28:39 AM)

Contents

Introduction to Computers and Programming

Principles of Computer Programming

- Computer hardware changes frequently - from room-filling machines with punch cards and tapes to modern laptops and tablets
- Capabilities of computers have changed rapidly (storage, speed, graphics, etc.)
- Languages used to program computers have also changed over time
  - Older languages: Fortran, C, C++
  - Newer "compiled" languages: C#, Java, R
  - Newer "interpreted" languages: Python, JavaScript
- This class is about "principles" of computer programming
  - Common principles behind all languages won't change, even though hardware and languages do
  - How to organize and structure data
  - How to express logical conditions and relations
  - How to solve problems with programs

Programming Language Concepts

- Machine language
  - Computers are made of electronic circuits
  - Basic instructions are encoded by setting wires to "on" or "off"
    - Read data, write data, add, subtract, etc.

  - Binary digits represent on/off state of wires in a circuit
  - Machine language: which sequence of binary digits (circuit state) represents which computer instruction
    * Example instruction: 0010110010101101
  - Most CPUs use one of two languages: x86 or ARM
- Assembly language
  - Easier way for humans to write machine-language instructions
  - Use a sequence of letters/symbols to represent an instruction, instead of 1s and 0s.
    * Example x86 instruction: `movq %rdx, %rbx`
  - **Assembler**: Translates assembly language code to machine instructions
    * One assembly instruction = one machine-language instruction
    * x86 assembly produces x86 machine code
  - Computers can only execute the machine code
- High-level language
  - More human-readable than assembly language
  - Each statement does not need to correspond to a machine instruction
  - Statements represent more "high-level" concepts, such as storing a value in a variable, not "machine-level" concepts like "read these bits from this address"
  - Most languages we program in are high-level (C, C#, Python…)
  - **Compiler**: Translates high-level language to machine code
    * Small programs in high-level language might produce lots of machine code
    * Compiler is specific to both the source language and the target machine code
  - Compile then execute, since computers can only execute machine code
- Compiled vs. Interpreted languages
  - Not all high-level languages use a compiler - some use an interpreter
  - **Interpreter**: Lets a computer "execute" high-level code by translating one statement at a time to machine code
  - Advantage: Less waiting time before you can run the program (no separate "compile" step)
  - Disadvantage: Program runs slower, since you wait for each high-level statement to be translated before the program can continue
- Managed high-level languages (like C#)
  - Combine features of compiled and interpreted languages
  - Compiler translates high-level statements to **intermediate language** instructions, not machine code
    * Intermediate language: Looks like assembly language, but not specific to any CPU
  - **Runtime** executes compiled program by *interpreting* the intermediate language instructions - translates one at a time to machine code
  - Advantages of managed languages:

    - In a "non-managed" language, a compiled program only works on one OS + CPU combination (**platform**) because it is machine code
    - Managed-language programs can be reused on a different platform without recompiling - intermediate language is not machine code and not CPU-specific
    - Still need to write an intermediate language intepreter for each platform (so it produces the right machine code), but in a non-managed language you must write a compiler for each platform
    - Intermediate-language interpreter is much faster than a high-level language interpreter, so programs run faster than an "interpreted language" like Python

Software Concepts

- Flow of execution in a program
  - Program receives input from some source, e.g. keyboard, mouse, data in files
  - Program uses input to make decisions
  - Program produces output for the outside world to see, e.g. by displaying images on screen, writing text to console, or saving data in files
- Program interfaces
  - **GUI** or Graphical User Interface: Input is from clicking mouse in visual elements on screen (buttons, menus, etc.), output is by drawing onto the screen
  - **CLI** or Command Line Interface: Input is from text typed into "command prompt" or "terminal window," output is text printed at same terminal window
  - This class will use CLI because it's simple, portable, easy to work with – no need to learn how to draw images, just read and write text

Programming Concepts

- Programming workflow (see flowchart)
  - Writing down specifications
  - Creating the source code
  - Running the compiler
  - Reading the compiler's output, warning and error messages
  - Fixing compile errors, if necessary
  - Running and testing the program
  - Debugging the program, if necessary
- Intepreted language workflow (see flowchart)
  - Writing down specifications
  - Creating the source code
  - Running the program in the interpreter

  - Reading the interpreter's output, determining if there is a syntax (language) error or the program finished executing
  - Editing the program to fix syntax errors
  - Testing the program (once it can run with no errors)
  - Debugging the program, if necessary
  - **Advantages**: Fewer steps between writing and executing, can be a faster cycle
  - **Disadvantages**: All errors happen when you run the program, no distinction between syntax errors (compile errors) and logic errors (bugs in running program)
- Integrated Development Environment (IDE)
  - Combines a text editor, compiler, file browser, debugger, and other tools
  - Helps you organize a programming project
  - Helps you write, compile, and test code in one place
  - Visual Studio terms:
    * Solution: An entire software project, including source code, metadata, input data files, etc.
    * "Build solution": Compile all of your code
    * "Start without debugging": Run the compiled code
    * Solution location: The folder (on your computer's file system) that contains the solution, meaning all your code and the information needed to compile and run it # First Program

The students should understand all the components of a simple "Hello World" program:

```
using System;

class Program {
    static void Main() {
        Console.WriteLine("Hello, world!");
    }
}
```

- Comments (in line and block)
- `using` statements and namespace / API concepts
- blank lines and spacing
- indentation
- intro to classes and methods' structures (body / header)
- status of `main` method
- intro to Console's `Write` and `WriteLine`
- string literal

Rules and Conventions

- The difference between a "rule" (e.g. case-sensitivity) and a "convention" (commenting your code).
- Reserved words
- Identifiers and naming conventions
- That the distinction can vary with the programming language
- Importance and role of `{` and `}`