Computer Science I
CS 1511

Email: rmaclin
http://www.d.umn.edu/~rmaclin/cs1511/index.html

Outline

I. Introduction
- A. Computer Science History
- B. Computer Science as Science
- C. Computer Systems
  1. Hardware
  2. Software
  3. Computer Languages
  4. Compilation

Outline (cont)

I. Introduction (cont)
- D. Software Development
  1. Software Life Cycle
  2. Program Development
     a. Five Steps to Good Programming
     b. Top-Down Programming
  3. Software Engineering

Computer Science History

- Alan Turing
  - WW II
  - Enigma
  - “Computers”
- John von Neumann
  - Programs as data
- ENIAC

Hardware “Generations”

- Hardware
  - vacuum tubes
  - transistors
  - printed circuits
  - integrated circuits
- Moore’s law

Environment “Generations”

- Environments
  - single process
  - batch process
  - time-shared
    - one powerful computer serving multiple users
  - personal computer
    - multiple individual computers
  - client/server
    - individual computers (clients) interacting with powerful computer providing services to multiple users (server)

What is Computer Science?

Computer Science is *not*

- Computer Literacy
- *just* Computer Programming

Computer Science has aspects of:

- Mathematics
- Science
- Engineering
- Applied Science
- Ethics

Computer Science is ...

- Mathematics and Logic
  - solutions must be precise
  - programs are written in formal programming languages
  - programmer must think logically and symbolically

Computer Science is ...

- Science

  Scientific method:
  - formulate *hypothesis* to explain phenomenon
  - *test* hypothesis by conducting *experiment*

  Computer Science Method:
  - formulate *algorithm* to solve problem
  - *test* algorithm by writing *program*

Computer Science is ...

Engineering

- modeling
- modularity
- maintainability

Interdisciplinary

- a tool

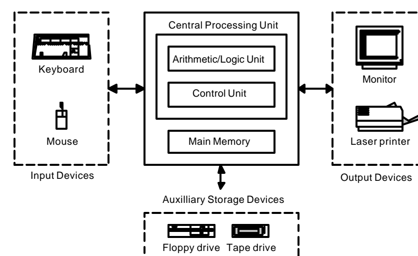
Ethics

- privacy
- security
- liability

Computer System

- Hardware
  - electronic parts connected together
  - examples: keyboard, CPU, memory, printer, …
- Software
  - “programs” that are executed on the hardware
  - examples: operating system, word processor, database program, ...

Computer Hardware



Central Processing Unit (CPU)

- Control Unit
  - decides instruction to execute
  - retrieves instruction from main memory
- Arithmetic/Logic Unit
  - performs instructions

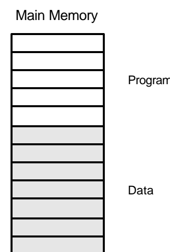
Memory (Storage)

- Main memory
  - memory unit is 1 or 0 (binary digit or "bit")
  - bits grouped into 8-bit "bytes"
- Auxilliary storage
  - magnetic/optical disks or tapes permanently store programs in "files"

Computer Software

- Systems software
  - operating system
  - system support software
  - system development software
- Application software
  - general-purpose software (word processing, database, etc.)
  - application specific (JPL image analysis, etc.)

Program in Memory



- No distinction between program and data for program
- CPU executes instructions from program
- Program may access data in memory

Computer Languages

- Machine language
- Assembly language
- High-level languages (including C)
- "Natural" language (the eventual goal)

Machine Language

- Instructions composed of 0's and 1's
- Instructions are groups (8, 16, 32, 64) of bits that mean something to the computer

  Example: 1011000100000001
- Difficult to understand (for humans)

Assembly Language

- Symbolic language
- Closely corresponding to machine language
  LOAD R1,#1
  corresponds to
  1011000100000001
- Translated using assembler to machine language

High-level Language

- Composed of English characters and words
- Example: printf("Welcome to CS 1511.");
- Does not correspond directly to a machine language instruction (generally corresponds to a series of instructions plus data)
- Translated into machine language (using a *compiler*)

"Natural" Language

- A goal of computer language developers
- Example: "Computer, please calculate the total number of people taking CS 1511."
- A lot more difficult than it seems
  total number
  people
  taking
  CS 1511

Creating a Program

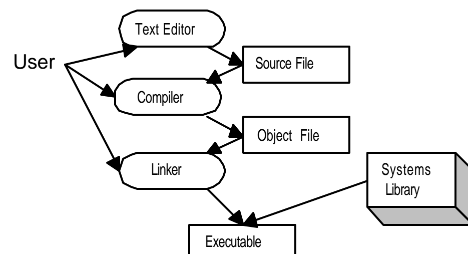
- Write/edit a source file (*name*.c)
- Compile source file to create object file (*name*.o, *name*.obj)
- Link object file(s) with system utilities to create executable file (*name*.exe, *name*)

- Compiling and linking sometimes done in one step

Compilers

- Translate source code in high-level language to *object* code
- Object code may contain references to:
  – source code from other source files
  – code for system utilities
- Linker assembles all of these pieces together into one machine language file (an executable file)

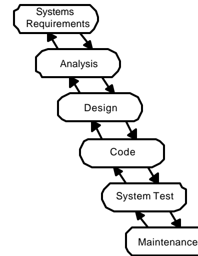
Creating a Program



Program Execution

- Program loaded into main memory by a program called a *loader*
- Resulting program may access input/output devices using system utilities (find out what the user typed at the keyboard, print a line to the screen, etc.)

System Development Life Cycle



- Waterfall model
- Creation of program/system a series of interacting steps
- Steps may flow forward/backward to various steps

Program Development

1. Understand the problem
2. Develop a solution
3. Write code for the solution
4. Run the program
5. Test the program

- Not always linear, may back up (debug program, solution)

1. Understand the Problem

- State what is to be done
- Specify data and assumptions
- Specify output and its form

Example problem:
Going from UMD to Metrodome
Assumption(s):
"UMD" = 10 University Drive
Output:
List of actions

2. Develop a Solution

Algorithm:
Finite sequence of statements that solve problem
Possible statements are:
1. Clear
"Get in the car" vs. "Get in the car in the parking lot outside the Med School Building"
2. Unambiguous
"Eat the bread at the bakery"

3. Write Code

- Translate algorithm into high-level language (C)
- Check program -- "walk through it"
- May need to redesign solution (if difficult or impossible to code)

4. Run the Program

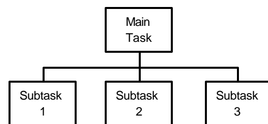
- Compile the program
  - if compilation errors, modify program using editor
- Run the program
  - if run-time errors, modify program

5. Test the Program

- A program that runs is not necessarily a *correct* program
- Once program runs without errors, test results for correctness
- If output is not what problem statement specifies, the program *logic* is incorrect

Developing an Algorithm

- Strategy: divide problem into subtasks
  "divide and conquer"
- Structure Chart:



Structure Chart Boxes

- Example: getting from UMD to MetroDome
- Algorithm:

  Subtask 1: Get in car at parking lot …

  Subtask 2: Drive to Minneapolis

  Subtask 3: Find MetroDome

  Subtask 4: Get out of car

Developing an Algorithm (cont)

- Strategy (cont): repetitively divide tasks until each task is easily solved
- Example: Subtask 2 - Drive to Minneapolis

  2.1 Drive east to I35

  2.2 Drive south to I35W

  2.3 Drive south to Minneapolis
- Each division of a task is a "stepwise refinement"

Stepwise Refinement

- Do stepwise refinement until all tasks easy
- Example: 2.1 - Drive east to I35

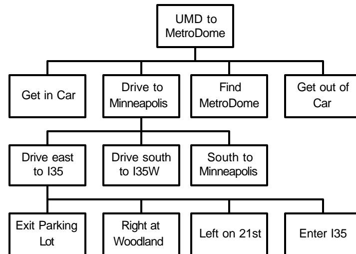
  2.1.1 Exit parking lot to East

  2.1.2 Turn right on Woodland

  2.1.3 Turn left on 21st

  2.1.4 Enter I35
- This process is know as *Top-Down Design*

Multi-layer Structure Chart

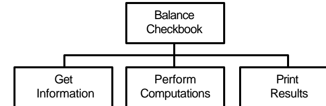


Another Example

Problem: Balance checkbook

Top-level tasks

1. Get information
2. Perform computations
3. Print results



Module Specification

- Top-level task is called a *Module*
- Module specification includes:
  1. Data received (input)
  2. Information returned (output)
  3. Logic used (how)
- Modular programming

Module Example

2. Perform Computations Module

Data:

Starting balance

Transaction type

Transaction amount

Information returned:

Ending balance

Logic:

If transaction a deposit, add to balance, else subtract

Pseudo-code

- Precise algorithmic description of program logic
- Can be used instead of (or with) structure charts
- Solutions for modules written in precise language

Pseudo-code Example

1. Get information
   - 1.1 Get starting balance
   - 1.2 Get transaction type
   - 1.3 Get transaction amount
2. Perform computations
   - 2.1 IF deposit THEN add to balance ELSE subtract from balance
3. Print results
   - 3.1 Print starting balance
   - 3.2. Print transaction
     - 3.2.1 Print transaction type
     - 3.2.2 Print transaction amount
   - 3.3 Print ending balance

Code Testing

- Blackbox Testing - test program without knowing how it works (look at specifications, design tests)
- Whitebox Testing - test program knowing how it works (look at code and try to design tests to "exercise" all aspects of the code)

Software Engineering

- "Use of sound engineering methods and principles to obtain software that is reliable and works on real machines"
- Guiding principles concerning:
  - readability
  - modularity
  - maintainability
  - etc. (lots more)