12.010 Computational Methods of Scientific Programming
Fall 2008

12.010 Computational Methods of Scientific Programming

Lecturers

Thomas A Herring,

Chris Hill,

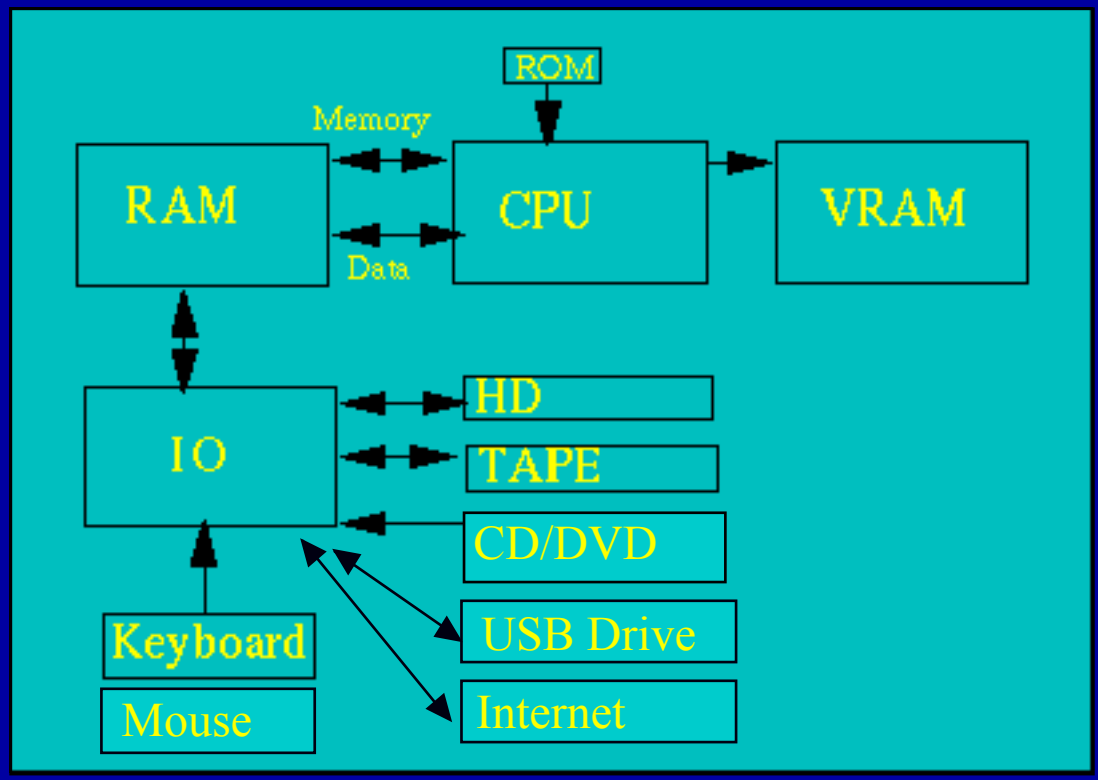
Review Lecture 01

- Language characteristics:
  - Compiled versus interactive
  - Numeric versus symbolic
- Algorithm development
  - Statement of problem
  - Algorithm design
  - Algorithm implementation
  - Verification

Today's Lecture

- Interface to computer hardware
- Computer Basics:
  - Memory,
  - data transfer,
  - number representation
- Example algorithm development

Computer components



CPU - Central Processor Unit (e.g., G3, Pentium III, RISC)

RAM - Random Access Memory (generally lost when power cycled)

VRAM - Video RAM (amount sets screen size and color depth)

ROM -Read Only Member (used for boot)

IO Input/Output. IO is through interfaces such as SCSI (Small Computer System Interface), USB (Universal Serial Bus), Firewire (video standard), Ethernet.

HD Hard Disk (permanent storage); CDROM Compact Disk Read-only-memory; CD-RW; DVD Read/Wrire

Components: OS

- The Operating System (OS)
  - Controls everything in the way the computer works.
  - Not Specific to a CPU type but often some OS's are associated with specific CPU
    - G3/4/5 68x series MacOS
    - Pentium, x86 DOS (Windows)/Mac OSX
    - SPARC Solaris (Unix)
  - OS controls IO and memory management (latter is important in multi-tasking OS).
- Specific program implementations are often dependent on OS

Programming interface to OS

- Depending on language used, OS interface may or may not be important
- For Fortran, C, C++ when program is linked OS routines are needed
  - How to read from keyboard or file?
  - How to write to screen or disk?
- In your program you do not need to the details but some OS routines have more features (details later)

Hard Disks

- Hard disks: Contains the computer's "file system" (allows access through file names)
- Directory structure: Points to where files are located (reason less space than size of disk+some calibration tracks).
- Actual contents on HD and directories depend on OS used (different standards exist e.g, HF HF+ for Mac, FAT16, FAT32 Windows, EXT2 for Linux
- In general, OS can only use their own file-system.
- Network File System (NFS) allows connections between different computer systems

Other data sources

- CDROM: File system is standard so all computers can read all CDs. (May not be able to use the files from the CD but can read them)
- TAPE standards a very different systems but media can be used on different systems
- Correct IO port is needed to attach any device (for example need SCSI port for SCSI disk)
- Universal Serial Bus (USB Drives): Exchangeable between systems.
- Internet: Communication protocol is standard

Computer basics

- Smallest thing a computer knows is a bit 0 or 1 (false/true)
- CPU basically only knows how to perform and, or, xor (exclusive or) operations
  - And returns true if both same
  - Or returns true if either true
  - Xor returns true if different
- CPU is a massive collection of and and or gates
- A specific CPU has a set of instructions it can execute (usually small 50-100) (Machine language)

Basics

- The number of instructions per seconds is set by the “clock speed” e.g, 500 MHz Pentium III (MIPS)
- One clock tick is called a cycle and modern CPUs can often execute more than one instruction per cycle
- All programming ultimately ends up as a set of instructions to executed by the CPU (compiling/linking or interpreter do this for you).
- Floating point speed is measured in “floating point operations per seconds” flops.

Bits/Bytes and words

- To manage things, bits are grouped into larger units
  - 2 bits = 1 nibble (don't see much anymore)
  - 8 bits = 1 byte (still common)
  - 2/4/8 bytes = word (varies between CPU)
  - Most desktop machines are 32-bit words but 64 bits machines are becoming more common (set by data bus)
  - Why important? Sets minimum size unit you can access in program, and often precision

Grouping words and bytes

- Number of unique values that can be represented depends on number of bits
- For n bits: unique values are 2^n-1
- For n=8 (byte) = 255
- Grouped into larger units to represent different things
- ASCII (American Standard Code for Information Interchange)
  - Basic version is 7 bit (127 characters)
  - A-Z, a-z, 0-9 and special characters
  - Values <32 are “control characters”

Number types

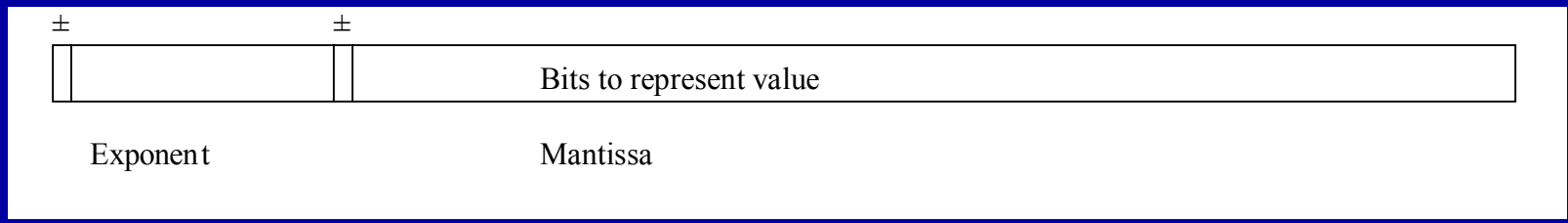
- Numbers represented in different base systems
  - Binary base 2 (0-1)
  - Octal base 8 (0-7)
  - Hexadecimal base 16 (0-15, with A-F representing 10-15)
  - E.g, $54_{10}=36_{16}=66_{8}=110110_{2}$
- Prefixes: kilo = 1024 (2^10); mega=1048576 (2^20); giga=1073741824 (2^30) (approximately 10^3,10^6,10^9)
- It is becoming more common now to use decimal values instead of binary versions (i.e., 1000 for kilo rather than 1024). There seem to be no standard rules for the usage especially for speed values. Memory usage still tends to be binary version.

Integer numbers

- Integer numbers can be represented exactly (up to the range allowed by the number of bytes)
- A 2-byte integer, unsigned 0-65535, signed ±32767 (sometimes called short)
- A 4-byte integer, unsigned 0-4294967295, signed ±2147483827
- (With a 32-bit address bus, can have 4Gbytes of memory—reason max memory is limited in computers)

Floating point

- Representations vary between machines (often reason binary files can not be shared).



  - Precise layout of bits depends on machine and format all formats are (mantissa)*$2^{(exponent)}$. (Above is not IEEE, exponent is 2s-complement in IEEE).
  - IEEE: 4-byte floating point is 8 bit exponent, 24 bit mantissa (1 sign bit for each), 7 significant digits, range $10^{\pm 38}$

Storage in memory

- Memory in a computer can be treated as a linear array of bytes from 1-<size of memory>. The OS should keep track of which parts of memory are being used and which parts are still free for use by programs and data.
- Some computers do “byte-swapping” i.e., the bytes are not counted linearly but rather are switched. The main (but not only) styles are Big Endian (HP, Sun, Macs) and Little Endian (PC).
- Affects ability to transfer binary data (TCP knows this and will switch a certain degree)

Other usage terms

- Most things are measured in bytes (memory size, data storage etc.)
- Transfer rates tend to be in “baud” which is “symbols-per-seconds”. Often, but not always, a symbol is a bit but with modern modems (phase shift keying, PSK) a symbol may has 4 or more states.
- Bits-per-second is a better measure and often the term used for transfer rates e.g, 56K modem is 56K bits per seconds.

Problem solving

- The clearer you can state the problem you are solving, the algorithms to used, and the possible problem areas, the easier your programming will be.
- Rough rule: 90% of time should be spent on design, only 10% on actually writing code and getting it to work.
- A good program is like good literature (it should logically flow)
- All your programs should be written in English before you start.

Program structure

- Basically all programs can be broken into three major parts:
  - **Input**: program collects the information it needs
  - **Computation:** Does the necessary evaluations to solve the problem
  - **Output:** Output its results for the user
- In an interactive program these parts may be looped over.

Language features

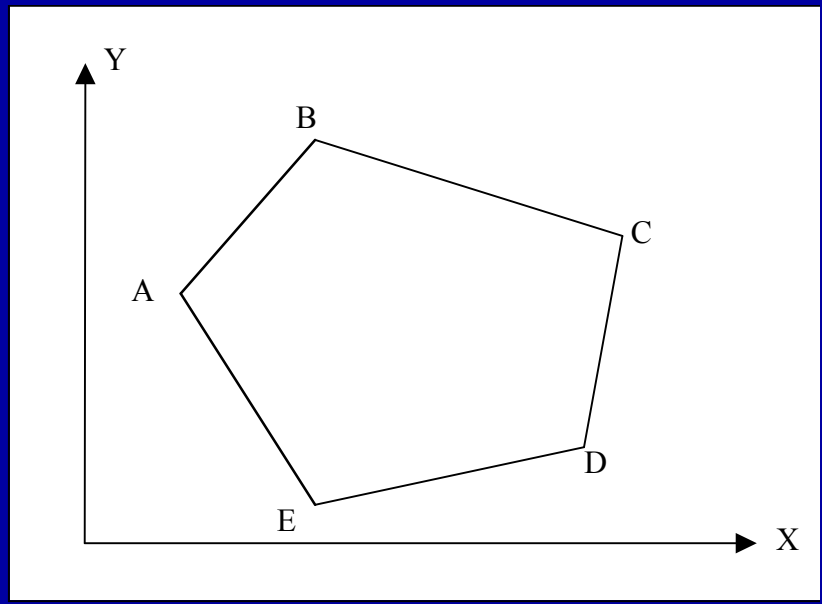
- All languages have the following basic features:
  - Start and end features
  - Input/output commands from and to a variety of sources
  - Decision structure (i.e., conditional branching and looping structures, error checking)
  - Assignment (setting variable to values, computing results)
  - Module structure that allows separation of functions.

Features 02

- A program is made up of the appropriate combinations of these features.
- Between languages the specifics of the features vary and some have more features than other.
- The syntax is different between all the languages although there are enough similarities to make it confusing (e.g., for versus do; end if versus end)
- So while learning the syntax you should keep careful note of how commands are structured in each language.

Specific problem example

- Problem: Find the area of an arbitrarily shaped plane figure.
- Figure defined by X,Y coordinates of vertices



How do we start solving problem?

- First: Ask questions and lots of them
- What basic algorithm should be used?
- Numerical integration by discretizing the shape? (i.e.. Make a fine grid over the shape and sum area of grid elements inside shape)
- Has problem that accuracy will be limited by element size in integration. Runtime will depend on size of grid.
- Break figure into triangles?
- Sounds OK. Can be made arbitrarily accurate
- Look for an analytic solution in a numerical algorithms book. Hint: Look at Green's Theorem which relates an area integral of the curl of a function to the line integral of the function (see: http://mathworld.wolfram.com/GreensTheorem.html)

Sample problem 02

- Let's say we decide that breaking the figure into triangles is the algorithm to be used.
- So what will we need to do this:

(a) How do we get the information about the shape into the program.

(b) A way of computing the area of a triangle

(c) A way of forming triangles from the coordinates.

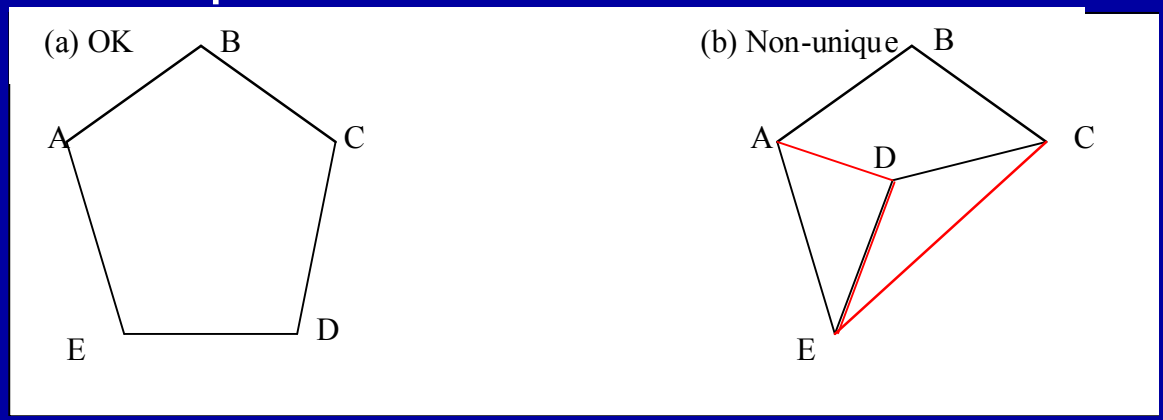
(d) how do we report the result.

- Algorithms (and routines) to compute areas of polygons can be found on the web (search "area of polygon"). In programming, we look at is how build these modules into a complete system that includes not just the algorithm but also the IO and logic needed.

Input options

(a.1) How do we get the information into the program?

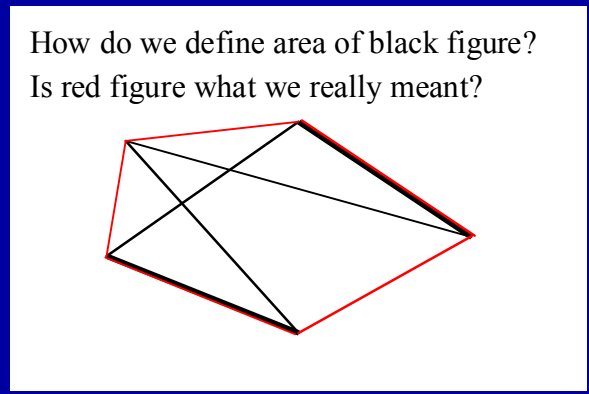
(a.2) Consider possible cases:



- (a.3) Input can not be completely arbitrary (although in some cases it can be)

Input options 02

- In some cases, for an arbitrary set of coordinates the figure is obvious, but in others it is not
- So how do we handle this?
- Force the user to input or read the values in the correct order?
- What if user makes a mistake and generates a figure with crossing lines?
- Warn user? Do nothing?



Final

- Finish up this section in Lecture 3.
- As the languages are covered we will explore more the concept on program development discussed here.
- Starting lecture 3, we will put this into practice with Fortran
- Homework #1 is on the web site. It is due Thursday September 25.