Overview: Introductory Topics

- **What is a computer?**
- **What is a program? What is programming?**
- **An Engineering Problem-Solving Methodology**
- **General format and components of a C program**

What is a Computer?

An electronic **machine** which can be programmed to carry out routine mental tasks by performing **simple operations** at very **high speed**.

Simple operations:

add two numbers
compare two numbers
get the second letter of a word
…

High speed:

more than 10,000 million operations per second

Computer = hardware + software

Hardware:

physical equipment used to perform computations: chips & circuits inside the machine, plus external devices (screen, keyboard, mouse, printer, network cables, …)

Software:

programs which control the computer and allow the user to perform useful tasks: word processor, accounting, library catalog, web browser, email, ...

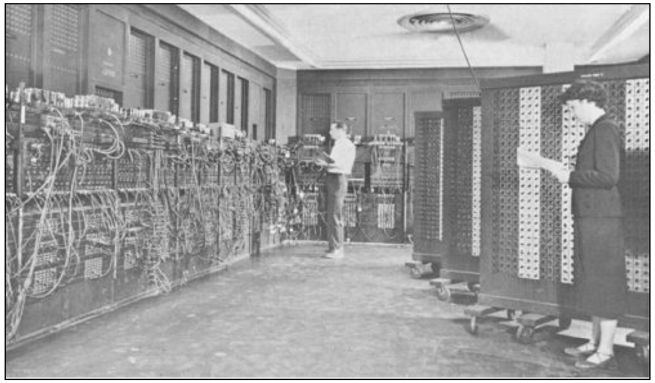
Hardware/Software distinction ⇒ computers are "reconfigurable"

computers do whatever their programs tell them to do.

Computers: history

First electronic computer was built in the 1930's by Dr. J. Atanasoff and C. Berry at Iowa State University.

1946 ENIAC: First large-scale general-purpose computer, University of Pennsylvania. 30 tons, 18000 tubes, 10×30 foot space.



Computers: today / future

Computers are everywhere!

Trends: smaller, cheaper, more powerful, moveable, more useful(?)

Computers can be categorised by their size and performance:

- supercomputers
- mainframes
- workstations / personal computers
- handheld computers

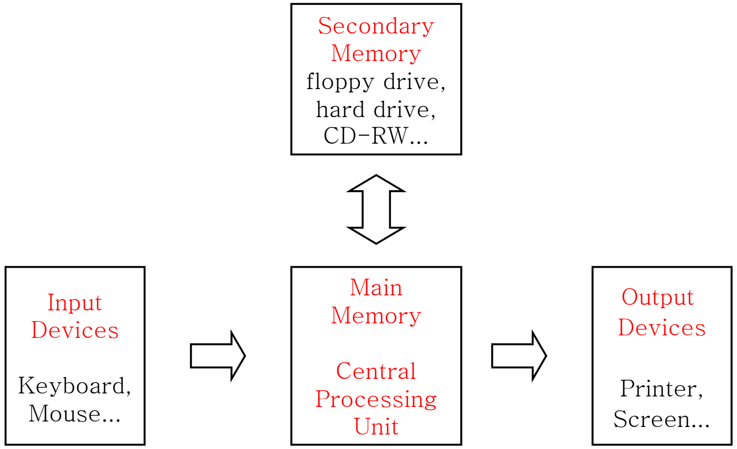
Also have computer networks: Local Area Network, the Internet...

Computers will play a key role in our futures:

- human genome project
- medical diagnosis and treatment
- automated vehicles (cars, planes, …)
- computer-based education
- and many others…

Computer hardware

Most computers have the same basic architecture:



Secondary Memory
floppy drive,
hard drive,
CD-RW...

Input Devices

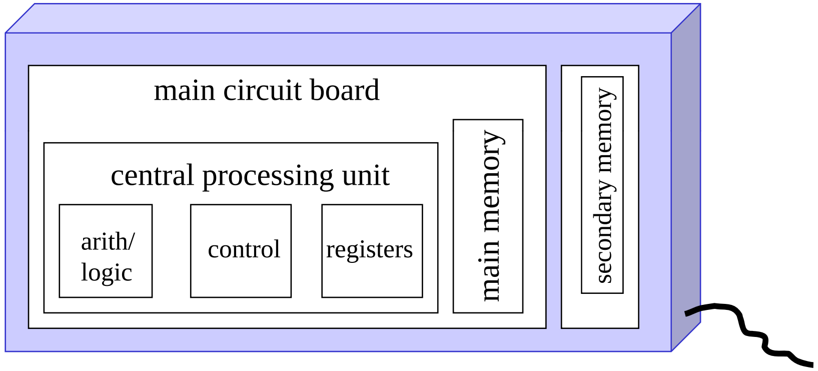
Keyboard,
Mouse...

Main Memory

Central Processing Unit

Output Devices

Printer,
Screen...



main circuit board

central processing unit

arith/
logic

control

registers

main memory

secondary memory

Processor (all in one “chip”)

Central Processing Unit (CPU)

Arithmetic/Logic Unit (ALU) - performs arithmetic

Control Unit (CU) - sequences operations

Registers - temporary data storage

Speed now measured in GHz (billions of cycles per second)

Single Board Computer & Tiny Computer

Main Memory (separate chips on same "circuit board")

- Temporarily holds programs (while being executed) and data.
- RAM (Random Access Memory) or ROM (Read-Only Memory)
- **Capacity** measured in GB (gigabytes) typically.
- Information stored in **binary -** sequence of **bits** (1's & 0's)
  8 bits = 1 **byte**, 2^{10} bytes = 1 **kilobyte (KB)**
  2^{20} bytes = 1 **megabyte (MB)**, 2^{30} bytes = 1 **gigabyte (GB)**

Secondary Memory (flash, magnetic, optical)

- Main Memory RAM loses its contents when switched off.
- Need to store programs and data on a more permanent basis
  ⇒ use secondary memory.
- Size is usually measured in TB (2^{40} bytes = 1 **terabyte (TB)**).

Peripheral Devices

Devices through which the computer communicates with the outside world e.g. keyboard, screen, mouse, printer, scanner, memory stick, ...

Growth of Hardware

Moore's Law (1965):

> "The number of transistors that can be inexpensively placed on an integrated circuit increases exponentially, **doubling** approximately every **two** years."

Based on Moore's law, every two years, the following approximately double:

- CPU speed at which computers execute their programs
- The amount of main memory
- The amount of secondary memory

Computer software

There are different kinds of software:

• ***Operating system:*** acts as interface between user programs or applications, and the hardware

- Examples: Windows, DOS, Unix, Linux, …
- Example programs: copy file, save file, delete file, …

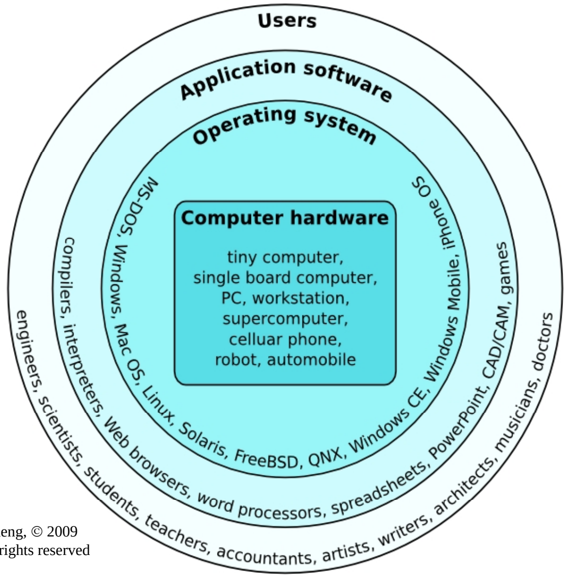
• ***Application software:*** someone else wrote it for you

- Examples: word processor, spreadsheet, web browser, …

• ***User programs:*** what you write

- e.g. write a C program to calculate the area of a circle with a given diameter
- you'll see plenty of examples in the Practicals!

Layers of Software



Users

Application software

Operating system

Computer hardware

tiny computer,
single board computer,
PC, workstation,
supercomputer,
celluar phone,
robot, automobile

MS-DOS, Windows, Mac OS, Linux, Solaris, FreeBSD, QNX, Windows CE, Windows Mobile, iPhone OS

compilers, interpreters, Web browsers, word processors, spreadsheets, PowerPoint, CAD/CAM, games

engineers, scientists, students, teachers, accountants, artists, writers, architects, musicians, doctors

Created by Harry H. Cheng, © 2009
McGraw-Hill, Inc. All rights reserved

Computer software: Programming

To carry out some task, the computer must be told exactly **what to do** and **how to do it**.

An **algorithm** is the series of steps involved in carrying out a particular task. To carry out these steps, the algorithm must be expressed in a form that the computer can understand.

An algorithm expressed in such a form is called a **program**. The user can then tell the computer to **execute** or **run** this program.

A computer program is written in a **programming language**.
“Natural” languages are for communicating with people.
Programming languages are for communicating with computers.

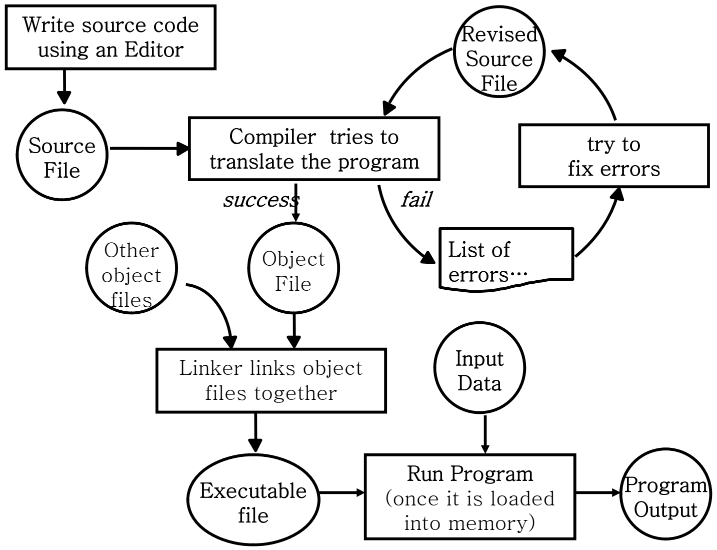
Programming Languages

- Computer only understands **machine language: strings of 1’s and 0’s**. Using machine language, a programmer can directly control the computer hardware.
    - machine language is specific to the type of computer.
    - machine language program is called **object code**.
    - ***extremely difficult:*** requires large amounts of object code to accomplish even simple tasks; requires a detailed knowledge of the hardware.

Programming Languages (contd.)

- **High-level languages** were developed to make programming easier:
    - precise enough so computer knows exactly what you mean, but don’t need to worry as much about low-level details (e.g. how many bits are used to store the number “12”)
    - high-level language program is called **source code** and is ***machine-independent*** and (ideally) ***portable*** from one type of computer to another
- However: computer doesn’t understand high-level language programs. They must be **compiled** (translated) into machine language, by a **compiler** for the high-level language being used.
    - compiler is another example of application software

Programming Flowchart


Write source code using an Editor
Source File
Compiler tries to translate the program
success
fail
Revised Source File
try to fix errors
List of errors…
Other object files
Object File
Linker links object files together
Input Data
Executable file
Run Program (once it is loaded into memory)
Program Output

Programming Languages: C

- There are hundreds of high-level programming languages:

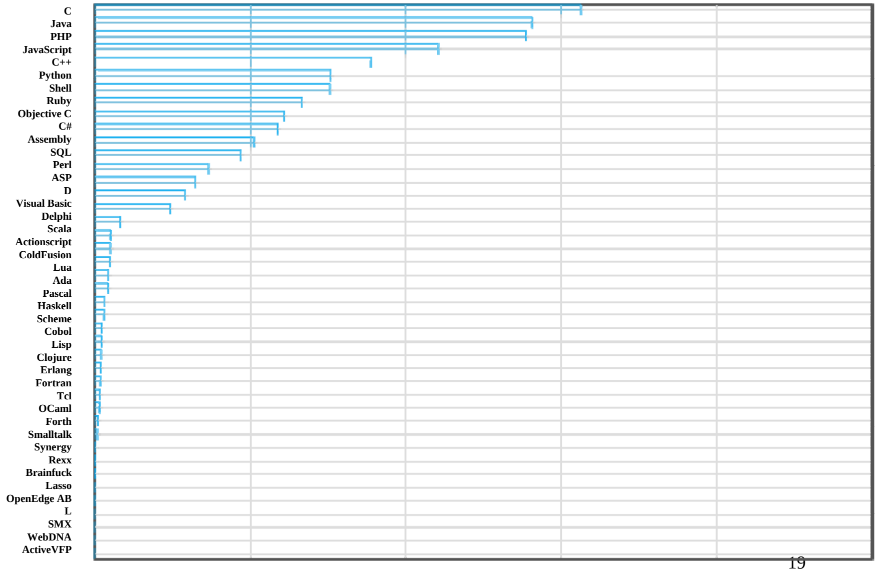
Pascal, Fortran, Basic, Ada, Lisp, Prolog, C, C++, Java, Cobol…

- All these languages are "equivalent"
  - but different languages are more/less convenient for particular tasks
- In this course, we'll learn just one: C
  - C developed in 1972 by Dennis Ritchie at AT&T Bell Labs
  - C was designed to write the UNIX operating system
  - over the years, the power and flexibility of C have made it a very popular general-purpose programming language

Timeline for Major Programming Languages

FORTRAN	John W. Backus, 1954
BASIC	George Kemeny and Tom Kurtz, 1964
Pascal	Nicolas Wirth, 1969
C	Dennis M. Ritchie, 1972
C++	Bjarne Stroustrup, 1979 (1983)
Java	Patrick Naughton, Mike Sheridan, and James Gosling of Sun, 1991
C#	Anders Hejlsberg, 2000

2014 Language Popularity http://www.langpop.com



Importance of C

C/C++ are dominant languages in industry

- Most large-scale projects are written in C/C++
- Most off-the-shelf software packages are written in C/C++
- The language of choice for real-time and embedded computing

TIOBE Programming Community Index for January 2014

http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

(Over 70% of languages in top 10 here)

Jan-14	Jan-13	Programming Language	Ratings	Change
1	1	C	17.87%	0.02%
2	2	Java	16.50%	-0.92%
3	3	Objective-C	11.10%	0.82%
4	4	C++	7.55%	-1.59%
5	5	C#	5.86%	-0.34%
6	6	PHP	4.63%	-0.92%
7	7	(Visual) Basic	2.99%	-1.76%
8	8	Python	2.40%	-1.77%
9	10	JavaScript	1.57%	-0.41%
10	22	Transact-SQL	1.56%	0.98%

What will you learn about C?

- with any programming language, there are 3 basic issues:
- ***how can I write a correct program in this language?***
  - meaning: the compiler will generate an executable file from my source code, rather than a list of errors
  - you will learn how to do this for simple C programs
- ***how can I solve problems using this programming language?***
  - meaning: how can I come up with a solution algorithm, and then accurately translate it into a correct program?
  - you will learn how to do this for small-scale problems
- ***how can I solve problems using this programming language in a good/efficient/elegant/cheap/better-than-the-competition way?***
  - you will not learn much, if anything, about this here…

An Engineering Problem-Solving Methodology

- before starting on the details of C programming, it is important to realise where programming “fits” into problem-solving in general
  - a big (and very common) mistake is to try to solve a problem by directly writing a program – only works (if at all) for experienced programmers on simple problems
- you should follow this procedure on every problem:

1. ***State the problem clearly.***
2. ***Describe the inputs and outputs.***
3. ***Work a simple example (mentally, by hand, calculator, …)***
4. ***Develop an algorithm: a step-by-step outline of a solution.*** ***Then implement your algorithm as a C program.***
5. ***Compile & test your program (use more than 1 test case!)***

Engineering Problem-Solving Methodology: Example

Write a program to convert a price from Sterling to Euro.

1. Problem statement:
   Write a program that will convert a number of Pounds Sterling to the equivalent number of Euro.
2. Inputs and outputs:
   - Input? value_sterling
   - Output? value_euro
   - What else? Conversion rate: 1 Euro=0.8277 Pounds Sterling
3. Simple example:
   - calculator: 10 Pounds Sterling converts to 12.08 Euro

Engineering Problem-Solving Methodology: Example (contd.)

4. Algorithm Design:
   - Get value in Pounds Sterling
   - Convert value to Euro
   - Display value in Euro

   Refine steps?
   “Convert value to Euro” becomes
   - the value in Euro is equal to the value in Pounds Sterling divided by the conversion rate 0.8277

   Implementation as a C program:
   - Consider data requirements
   - Convert each algorithm step into one or more C statements

Engineering Problem-Solving Methodology: Example (contd.)

```
/* Date: 16/01/2014      Version: 1.0      */
/* Program to convert Pounds Sterling to Euro */

#include <stdio.h>                                /* definition of printf and scanf */
#define CONVERSION_RATE 0.8277                    /* Conversion constant */

main()
{
   float value_sterling,                          /* input:  value in Pounds Sterling */
         value_euro;                              /* output: value in Euro       */

   /* Get the value in Pounds Sterling */
   printf("Enter the value of the product in Pounds Sterling: ");
   scanf("%f",&value_sterling);

   /* Convert the amount of Pounds Sterling to Euro */
   value_euro=value_sterling/CONVERSION_RATE;

   /* Display the converted Euro value */
   printf("That equals %.2f Euro.\n",value_euro);
}
```

Sample Run:

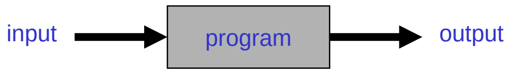
```
Enter the value of the product in Pounds Sterling: 10.00
That equals 12.08 Euro.
```

Engineering Problem-Solving Methodology

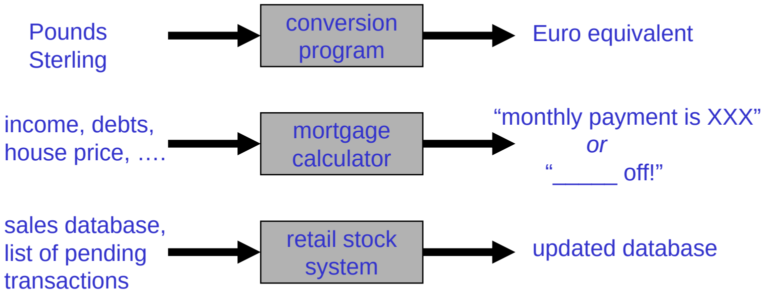
Program testing:

- first check you get correct answers in known, simple cases
- then check unusual (or "boundary") cases
  - what if you enter 0 Pounds Sterling? Or a very high number?
- should also check impossible cases (if any), to make sure your program can handle them appropriately: **error message**, rather than program crashing
  - what if the above example should not accept a negative number of Pounds Sterling? (currently, it does!)
- how do you know when you've fully tested a program?
  - **In general: never**. But you should try!

Programs process some **input** and generate some **output**:


input
program
output

- **Input:** data the program will process -- entered by the user from the keyboard, read from a file, interpreted from mouse clicks, etc.
- **Output:** results produced by running the program. This can be displayed on the screen, written to a file, printed, etc.


Pounds Sterling
conversion program
Euro equivalent
income, debts, house price, ….
mortgage calculator
"monthly payment is XXX"
or
"_____ off!"
sales database, list of pending transactions
retail stock system
updated database

General format of a C program:

```
/* initial comments */
preprocessor directives
main()
{
     declarations; /* comments */
     statements;   /* comments */
}
```

- C programs are ***free-format*** and ***case-sensitive***
- comment starts with `/*` and ends with `*/` (don't forget it!)
- execution of a C program ***always*** starts with `main()`
- declarations ***must*** go before statements
- all declarations and statements ***must*** end with a `;`

General format of a C program (contd.):

- C is free-format ⇒ comments, statements, etc. can begin anywhere on a line (e.g. doesn’t have to be in column 1).
- however, style guidelines have been developed, e.g.
  - Use **whitespace** (spaces, blank lines, …) to separate different components of your program.
  - Use indentation to convey information about relation of statements to each other (especially in **loops** – details later).
- C is case-sensitive: **`value_sterling`** , **`Value_sterling`** , **`value_STERLING`** are all different things.
  - by convention, use all-capitals for constants e.g. **`CONVERSION_RATE`**

Comments in C

- Comments are ignored by the C compiler – they are purely for the benefit of the programmer, and anyone else who may read the source code.
- Comments are optional and therefore don’t have to be included in a C program. However they can help to explain and document **what** the program does, and **why**.
- At a minimum, comments should explain the steps of the algorithm used to solve the problem being addressed.
- When evaluating your programs, we will take into account how appropriate (or otherwise) your commenting is…

Preprocessor directives in C

- these are instructions your program gives to the C compiler
  - your program can re-use existing pieces of code which are stored in a **library**
  - the ***standard C library*** provides many useful services, e.g. **`stdio.h`** provides input and output facilities and is used in nearly every C program
  - how do you know what to **`#include`** ?
    - examples, textbooks, view standard C library…
- can also be used to **`#define`** constants – makes your program more readable and understandable
- preprocessor directives have a different **syntax** to C declarations / statements. In particular, they don’t end in a **`;`**

Declarations in C

- before using any variables in your program, you have to tell the C compiler about them so that later statements can use them
  - example: **`float value_sterling`** -- tells the compiler that your program will make use of a variable called **`value_sterling`**, and that the type of this variable is **`float`** (meaning: it may have a fractional part)
- a variable in C has a name (called its **identifier**), a **memory location** where it is stored, a **type**, and a **value**
  - identifier should be meaningful e.g. **`x`** instead of **`value_sterling`**
  - identifier, type, and value -- you decide
  - memory location -- the computer decides

Statements in C

- statements are the detailed steps of what the program does
    - **`value_euro=value_sterling/CONVERSION_RATE`** tells the computer to take the current value of the variable **`value_sterling`** and divide it by the constant **`CONVERSION_RATE`**, and to store the result in the variable **`value_euro`** (over-writing the current value of **`value_euro`**)

- statements are not just for computations -- they also tell the computer when to ask the user for input, where to store the input data, what data to output and how, etc.

- good practice: one statement per line