6,000 Years in 60 Seconds

- c. 2500 BC, abacus invented and used in Sumeria
- 1600’s, digital mechanical calculators invented by Schickard, Pascal, Leibniz
- 1835, “analytical engine” invented by Charles Babbage, general purpose programmable computer uses punch cards and steam power
- 1890, (Herman) Hollerith machines used for calculations in the US census, company eventually becomes IBM

- 1941, Zuse Z3, binary, electro-mechanical, programmable computer, Germany
- 1944, Colossus, binary, electronic, programmable, UK
- 1946, ENIAC, decimal, electronic, programmable, USA
- 1949, CSIRAC, binary, electronic, programmable, Australia
- 1951, UNIVAC, the first ‘mass produced’ computer, 46 units sold @ $1,000,000+USD each
- 1954, IBM 650, a small, affordable computer, 2250 kg, $500,000USD each
- 1956, IBM releases first magnetic disk, 5MB capacity, $50,000USD
- 1960, IBM 1401, uses transistors, up to 16KB memory

The Modern Computer

What is a computer?
A machine that manipulates data according to instructions.

Despite their apparent complexity, at the lowest level computers perform simple operations on binary data. Conceptually, all sufficiently complex computers are able to perform exactly the same tasks, only that some are faster than others.

What makes up a working computer?
- hardware (motherboard, CPU, RAM, HDD, etc.)
- bootstrapping code (BIOS)
- device drivers
- operating system (Unix, Windows, etc.)
- software (games, utilities, etc.)
Computers and Computation

The elementary operations are very simple, for example:
- Add this number to the one you already have
- Store the value you have to this memory location
- Skip the next instruction, if the number just computed is zero

The computation \(181,444 + 345,648 + 856,221\) would be carried out as a ‘load, add, add, store’ sequence.

A computer has a limited repertoire of operations, but carries them out extremely quickly and with astounding reliability and precision.

Computers and Computation

Computers are now a household device, but 30 years ago were much less powerful, and much more expensive. These days a home computer carries out more than 100,000,000 operations per second, where an operation is the equivalent of a multi-digit addition. There is no other aspect of life in which natural abilities are boosted by a factor of 100,000,000. Travelling 100,000,000 times faster than walking would take you to London within a tenth of a second!

A desktop computer today has more power than a million-dollar computer of twenty-five years ago.

The Operating System

What is an operating system (OS)?
A complex piece of software that manages a computer’s resources and provides an interface to users and programmers.

You are likely familiar with the Windows family of operating systems. You now get the chance to work with the Unix family of operating systems.
- GNU/Linux, Mac OS X, FreeBSD, and Solaris
- long history; many innovations come from Unix systems
- Unix is multi-user and multi-tasking
- reliable server and workstation operating system

The Operating System

Since Unix is a multi-user operating system, you will have your own account on the CSE machines, with a unique username and password. Logging in to your CSE account, either from a lab machine or from home, will give your access to your files and settings. These are not to be shared with anyone else.

Keep in mind:
- logging into a Unix system gives you access to a terminal window
- a terminal window is for text commands which the OS executes
- common commands: `ls`, `cd`, `mkdir`, `more`, etc.
- many tasks can be performed through the graphical user interface (GUI)
**Programming Languages**

Why don’t we program in English?
- it is too informal
- it is too big

What does “Time flies like an arrow” mean?
So we invent a programming language that:
- is small
- is formal (syntax and grammar)
- is still reasonably intuitive for humans

Does a computer then run program code?
No. Because programming language instructions are usually too complex to execute directly, they must be translated into an even simpler machine language.

**From Design to Execution**

There is a progression from high-level to low-level:
- when a computer program is required, first there is informal, natural language discussion to determine its scope and nature—Requirements
- this is then translated into a more formal, specification of the program—Specification & Analysis
- the specification is then translated into program code in some programming language, this is now a formal specification of the program—Implementation
- the program code is then translated into machine language by a program called a compiler (we use gcc)

The output of the compiler, often called an executable, is a program that the computer, i.e. the hardware, can understand and run directly.

**The C Programming Language**

Historical notes:
- created by Dennis Ritchie in the early 70’s at AT&T Bell Labs
- named so because it succeeded the B programming language
- designed as a high(er)-level language to replace assembler
- powerful enough to implement the Unix kernel
- in 1978 Dennis Ritchie and Brian Kernighan published “The C Programming Language”
- now considered low-level and widely used for system and application programming

**The C Programming Language**

What is C?
A general-purpose, imperative (procedural) programming language.

General-purpose means that the language does not include domain-specific constructs. It is often used for system development but also widely used for application development.

Imperative programming is a programming paradigm characterized by, as the name suggests, a prescriptive approach. The programmer exactly specifies the sequence of steps the program should perform. Other programming paradigms are described in your textbook.

C is a relatively small language, it requires minimal run-time support, affords the programmer great (low-level) control, and produces fast code.
The C Programming Language

Like most programming languages, C supports features such as:
- program comments
- declaring variables (data storage)
- assigning values to variables
- performing arithmetic operations
- performing comparison operations
- control structures, such as branching or looping
- performing input and output

In addition, C supports some low-level constructs, such as pointers, which are not directly available in many other (higher-level) programming languages.

Andrew Taylor
COMP1911 Computing IA

Hello World

The program is complete, it compiles and performs a task. Even in a few lines of code there are a lot of elements:
- a comment
- a #include directive
- the main function
- a call to a library function, printf
- a return statement
- semicolons, braces and string literals

A Doing Thing

Programming or coding, i.e., the activity of writing computer programs, is a practical skill, you can only get better at it if you practice continually.

// Author: Kernighan and Ritchie
// Date created: 1978
// A very simple C program.

#include <stdio.h>

int main(void) {
    printf("Hello world!\n");
    return 0;
}

Andrew Taylor
COMP1911 Computing IA

A Closer Look

What does it all mean?
- //, a single line comment, use /* */ for block comments
- #include <stdio.h>, import the standard I/O library
- int main(...), the main function must appear in every C program and it is the start of execution point
- (void), indicating no arguments for main
- printf(...), the usual C output function, in stdio.h
- ("Hello world!\n"), argument supplied to printf, a string literal, i.e., a string constant
- \n, an escape sequence, special character combination that inserts a new line
- return 0, a code returned to the operating system, 0 means the program executed without error

Andrew Taylor
COMP1911 Computing IA
The C Compiler

As discussed, a C program must be processed and translated into machine code so it can be run by the computer. This process is known as compilation and is performed by a compiler. We use GCC (GNU Compiler Collection, formerly GNU C Compiler) for this task. The compiler (gcc) is itself a compiled program.

The simplest use of the compiler would be: gcc helloworld.c which produces the file a.out.

The command: gcc -Wall -Werror -O -o hello hello.c
- tells gcc to warn about all suspect code, -Wall
- tells gcc to treat warnings as errors, -Werror
- -O is typically used to optimise the executable code that is produced but when it is used in conjunction with -Wall and -Werror it can produce additional warnings that will help you
- tells gcc to call its output hello, -o hello
- tells gcc to work on the program in hello.c

The Task of Programming

Programming is a construction exercise.
- Think about the problem
- Write down a proposed solutions
- Break each step into smaller steps
- Convert the basic steps into instructions in the programming language
- Use an editor to create a file that contains the program
- Use the compiler to check the syntax of the program
- Test the program on a range of data

Think about eating a pizza - one bite at a time gets chewed and swallowed before the next bite is taken.

The task of programming

Everything comes down to careful advance planning, just like in any other construction project. Thoughtful design at the beginning always pays off in the end.

A day of debugging can save an hour of planning.

The best programs are simple, and include desired functionality and nothing else.

Keep it simple, stupid.

Coding Style

Code is like handwriting, in that everyone develops a unique style, but also in that there are certain conventions that must be followed, otherwise they both become illegible.

Style guides ensure that code:
- is uniform
- is easy to read (by you or others)
- is well documented
- is easy to debug (by you or others)
- conforms to good programming practice

Style Guide

The course style guide is available from the course website. You should always adhere to it lest you should lose style marks.
Variables

When we start writing non-trivial C programs we need to store and manipulate data values. Programs store and manipulate values using constructs called variables.

Here are some definitions of C variables:

```c
int i, j;
long int fibonacci;
float price;
double area;
char c;
```

What is a variable?

A variable is used to store a value. As its name suggests, the value a variable holds may change over the variable’s lifetime. A variable has a type, a name, a value and a memory address.

Consider this C variable definition:

```c
int length;
```

Here we define a variable called `length` of type `int`. It is automatically assigned a memory address (we won’t worry about this until later in the course) and it holds some random value (specifically, whatever value happens to be stored at the memory address it is assigned).

Remember to initialize!

When a variable is created, it is guaranteed to contain garbage. If you do not initialize it explicitly, you get strange non-deterministic behavior. So initialize it:

```c
int length = 0;
```

Identifiers

Assignment Operator (=)

It assigns the value on the right to the entity on the left.

Identifiers

The names, e.g., `length`, we give to program entities are known as identifiers. Identifiers should always be chosen to be as meaningful as possible, e.g., `length` is a lot more informative than `r7q34b`.

Identifiers must begin with a letter or underscore and may contain letters, digits and underscore, see Moffat, §2.1.

NB

- identifiers are case sensitive (common source of errors)
- check the COMP1911 Style Guide for naming conventions

Types

C is a typed language. Variables (and other entities) have types. Types specify what kind of values variables can hold. Types also tell us what operations can be performed on those variables. For example, we can multiply numbers but not strings.

Here are some useful C primitive types:

- `int` used to represent positive and negative integers
- `double` used to approximate real (fractional) numbers values
- `char` used to store single characters
Types

Limited Range and Precision

Each type is stored in a fixed amount of memory (more on this later). This means that integer types have limited range and floating point types have limited range and also limited precision. You must keep this in mind when writing your programs.

For example, on many machines the int type can hold values between $-2^{31}$ and $(+2^{31} - 1)$.

NB

Integer overflow errors and floating point precision errors often produce unexpected results without causing the program to crash.

ls

- Lists files in current directory (folder)
- Several useful switches can be applied to ls
  - ls -l (provide a long listing)
  - ls -a (list all files, i.e., show hidden files)
  - ls -t (list files by modification time)
  - Can combine options. For example, ls -la

mkdir

- mkdir directoryName
- Create (make) new directory called directoryName in the current working directory
- A directory is like a folder in windows
- To verify creation, type ls

cd

- cd directoryName
- Change directory
  - Change current directory to directoryName
  - directoryName must be in the current working directory
  - We will see how to use more complex names(paths) later
- Special directory names
  - cd ..
    - move up one directory (to parent directory)
  - cd ~
    - move to your home directory