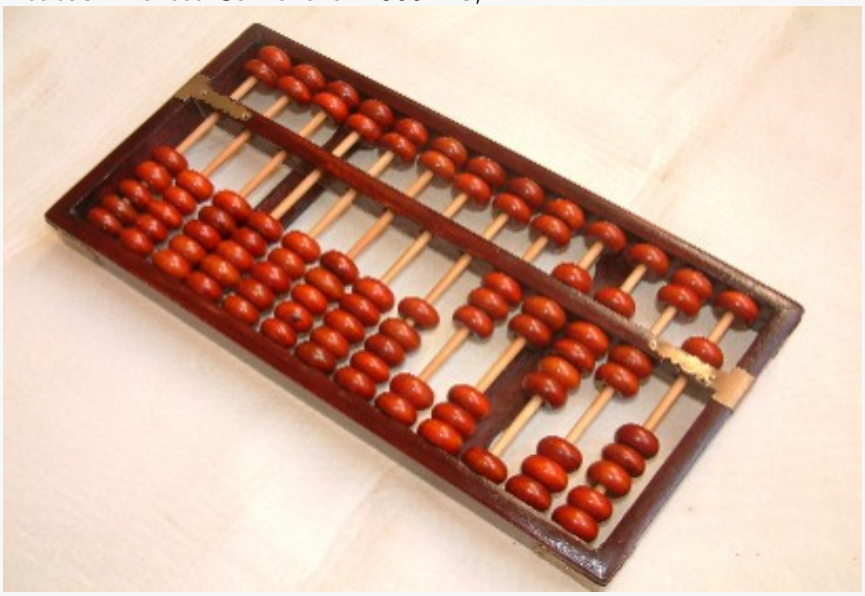
Computer Hardware: 2500 BC

Abacus invented Sumeria c. 2500 BC,

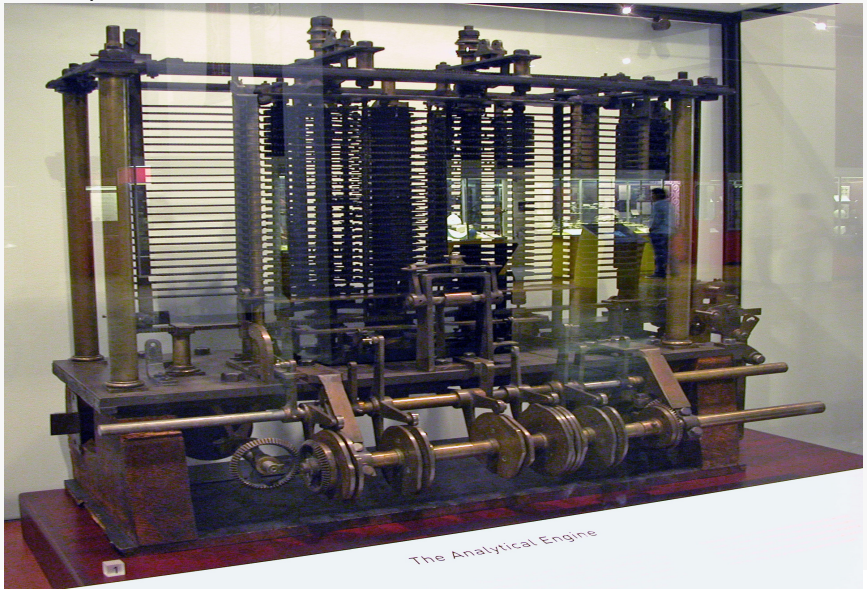


Computer Hardware: 100 BC

Antikythera mechanism - used to predict astronomical positions and eclipses

Computer Hardware: 1835

Analytical Engine invented by Charles Babbage 1835,
General purpose programmable computer uses punch cards and steam power



The first Coder: 1835

Ada Lovelace - mathematician who wrote the first program

Computer Hardware: 1890

Hollerith tabulating machine used for calculations in the US census, company eventually becomes IBM

Computer Hardware: the Electronic Era

- 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

Computer Hardware: the Integrated Circuit Era

- 1960's, integrated circuits start to be used in computers
- 1963, Apollo Guidance Computer, uses IC's, 1 MHz clock
- 1966, HP-2116, general purpose computer, 10 MHz clock
- 1971, Intel releases first commercial microprocessor: 4004
- 1976, Cray-1 supercomputer, 250 megaflops, 5.5 t, $5,000,000USD each
- 1980, Sinclair Z80, 50,000 sold, £99.5, 3.25 MHz, 1KB RAM
- 1981, IBM-PC, floppy (160KB), 10MB HDD, 256 KB RAM
- 1984 Apple Macintosh 128 KB RAM
- 1999, Intel Pentium III, 9.5 million transistors, 450 MHz
- 2008, the number of personal computers worldwide reaches 1 billion!

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 (Linux, Windows, etc.)
- software (games, utilities, etc.)

The Operating System

Operating system (OS) is a piece of complex software layer that manages a computer's hardware.
Allows you to program without knowing (independant) of hardware details.

- 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

Linux

Linux 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.

- 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

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:

1. when a computer program is required, first there is informal, natural language discussion to determine its scope and nature—Requirements
2. this is then translated into a more formal, specification of the program—Specification & Analysis
3. the specification is then translated into program code in some programming language, this is now a formal specification of the program—Implementation
4. the program code is then translated into machine language by a program called a *compiler* (we use `dcc`)

The output of the compiler (executable) is a program in a form that the computer hardware, can execute 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
- standardised under ANSI X3.159-1989, ISO/IEC 9899:1999 and ISO/IEC 9899:2011

Why C?

- classic example of an imperative language
- many libraries and learning resources
- widely used for writing operating systems and compilers as well as industrial and scientifc applications
- provides low level access to machine
- language you must know if you want to work with hardware

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

Hello World

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;
}
```

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 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

The C Compiler

A C program must be translated into machine code to be run.
This process is known as compilation.
It is performed by a compiler.
We will use a compiler named dcc for COMP1511
dcc is actually a custom wrapper aroung a compiler named clang.
Another widely used compiler is called (`gcc`).

Compiling A Program

- To create a C program in the file `hello.c` from the terminal:
  `gedit hello.c`
- Once the code is written and saved, compile it:
  `dcc hello.c`
- Run the program:
  `./a.out`

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

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:

```
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.

Variables

Consider this C variable definition:

```
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 a single integer value (from a limited range)
You must assign a variable a value before using it.
When a variable is created, its value is undefined.
If you do not initialize its value, you get strange non-deterministic behavior.

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`.
C identifiers must begin with a letter or underscore and may contain letters, digits and underscore

NB

identifiers are case sensitive (common source of errors)

Types

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 two important C primitive types:

`int` used to represent positive and negative integers

`double` used to approximate real (fractional) numbers values

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 $(+2^{31}-1)$.
Integer overflow errors and floating point precision errors often produce unexpected results without causing the program to crash. They can also be exploited by malicious users as security hole.

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 file, 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