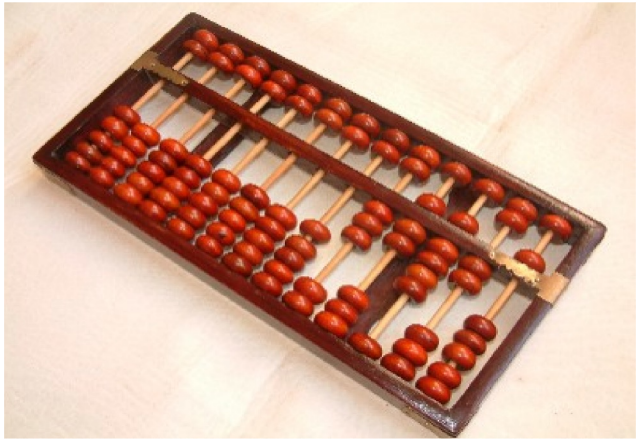
COMP1911: Introduction To Computers and C

Term 2, 2023

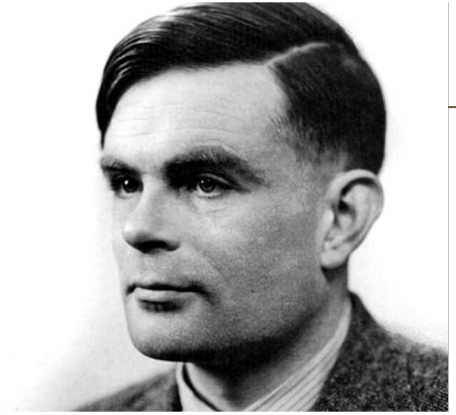
Computers

- “Computers” have existed for 1000’s of years
- For example, Abacus invented Sumeria c. 2500 BC,



- But, until 20th century, were specialised/simple devices

Computers (cont)



Modern computers are

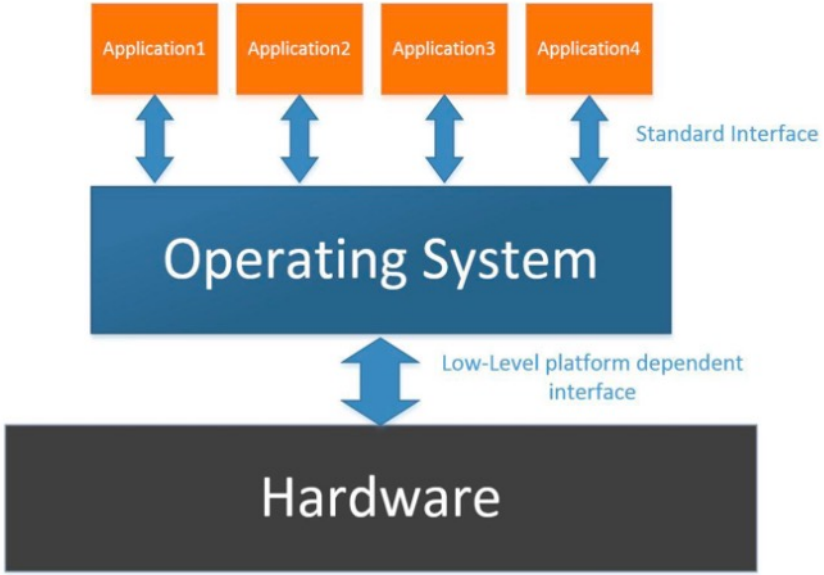
- electronic, digital, **stored-program**
- able to realise **any computable function**
  - demonstrated by Alan Turing in the 1940's

- **Alan Turing** is widely considered to be the **father** of theoretical **computer science** and **artificial intelligence**.
- During the Second World War, Turing worked at Britain's codebreaking centre that produced **Ultra intelligence**.

The Operating System

- What is an operating system (OS)?
- A complex piece of software that manages and abstracts a computer's resources and provides an interface to users and programmers.
- You are likely familiar with the Windows family of operating systems.
- Most programmers work on Unix based operating system.
  - Linux (e.g., in the CSE labs)
  - Mac OS X
  - Note: Windows 10 now has a unix sub-system build in

The Operating System



Source: https://embeddedcentric.com/embedded-operating-systems/

The Operating System

- In CSE we allow you to log in to our linux system either remotely or on our lab machines.
  Your username and password for this system is your zid and zpass.
- Do not share your zId and zPass. You will be liable.
- Keep in mind:
- You can interact with the CSE unix system either via terminal window or graphical user interface
- Commands typed into a terminal window are instructions used to manipulate data
- In this course we will build applications by interacting with the GUI/terminal (which are also applications)

Navigating Unix

- Linux commands are typed into a terminal.
- To open a terminal on the default cse setup you can right click on the terminal icon at the bottom of the screen

- Some useful `linux` commands to get started are:
  - **`ls`**
  - **`pwd`**
  - **`mkdir`**
  - **`cd`**

Navigating Unix - 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

Navigating Unix - pwd

- pwd *directoryName*
- **p**rint **w**orking **d**irectory
- a directory is like a folder in windows
- the working directory is the directory that you are currently in

Navigating Unix - mkdir

- mkdir *directoryName*
- **ma**ke (create) a new **dir**ectory called *directoryName* in the current working directory
- To verify creation, type ls

Navigating Unix - 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

Navigating Unix | Tips and Tricks

- Tab Completion: start typing a command, then press tab and linux will try to complete the rest of it or suggest possibilities

- Up Arrow Key: Typing this will bring up the last command you typed in, typing it twice brings up the second last command you typed in etc.

How can computers understand us?

Why can't a computer execute a program written in languages humans use, like English?

- it is too informal (vague semantic)
- it is too big (massive scope)

Fundamentally, a computer's circuitary is determined by a series of zero or non-zero voltages across transistors, which we represent as 0's or 1's

- We refer to this binary that a computer can execute as Machine Code
- Machine code is able to be directly interpreted by a computer's CPU to execute many layers of very basic instructions (e.g. addition)

Machine Code

Cut out the middle man. Just program in Machine Code??

Well this is what programmers used to do! However, it was:

- Error prone for humans but still used occasionally
- Not portable from one machine to another
- Unbearable. Imagine writing a program that looks like this
  0100101001000010001111100011000011101

Real programmers code in binary.

Introducing Programming Languages

So we invent a programming language that:

- is small
- is formal (syntax and grammar)
- is still reasonably intuitive for humans

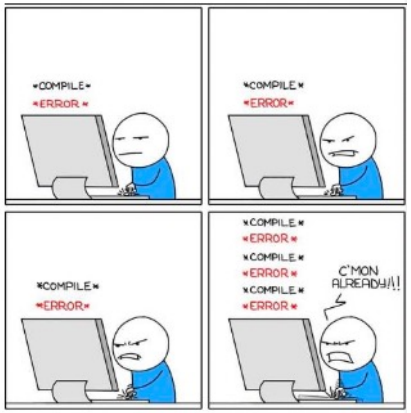
```
int length = 7;
int breadth = 4;
int area = length * breadth;
printf("Area: %d\n", area);
```

Programming Languages to Machine Code

How does a computer then run the program code?

- We use a program called a *compiler* to translate it into machine code (often called an executable) that the machine i.e. the hardware can execute directly.

- In this way a compiler allows us to write *abstracted* code without being concerned with many elements of the underlying computer.



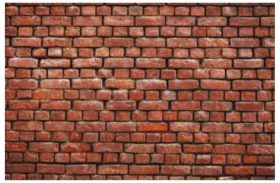
Algorithms and Programs

An algorithm is a set of (specific) instructions to accomplish a goal.

For example,

- make a cake
- build a wall
- sort a list of names

Similarly, a computer program is a set of instructions in a programming language (like C or Java or Python) that accomplish a goal.



Popular Baby Names

Top 10 Names for 2010

Rank	Male name	Female name
1	Jacob	Isabella
2	Ethan	Sophia
3	Michael	Emma
4	Jayden	Olivia
5	William	Ava
6	Alexander	Emily
7	Noah	Abigail
8	Daniel	Madison
9	Aiden	Chloe
10	Anthony	Mia

Programs

A program needs to be

- sufficiently detailed
- unambiguous
- eventually leading to goal

So we don’t use English for programming

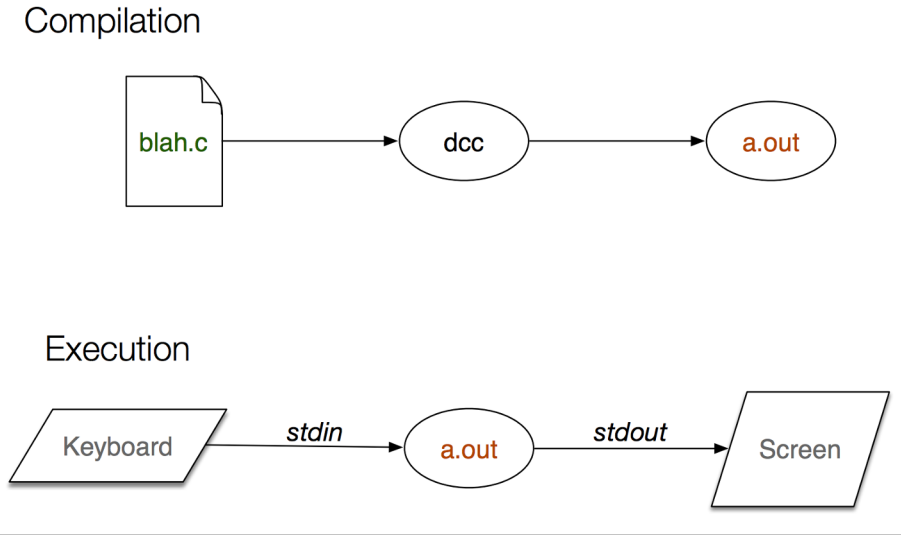
Programs

A program is a text document, containing

- a description of an algorithm
- expressed in a programming language (like C, Java, Python, etc.)

It cannot be directly executed on a computer

- need to translate to executable machine code



Programs

Typical program structure

- get input values
- process input to compute result
- display result

The C Programming Language

C is an important programming language

- relatively simple, widely used and forms the basis for many other languages
- venerable (developed in early 70's by Thompson & Ritchie)
- named so because it succeeded the B programming language
- widely used for system and application programming, powerful enough to implement the Unix kernel
- classic example of an imperative language
- widely used for writing operating systems and compilers as well as industrial and scientific 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

```
// 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` , 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

Escaping a Problem

I want a program that prints out the following

This is a "Hello World" demo.

What is wrong with the following line of code?

```
printf("This is a "Hello World" demo \n");
```

- The character " has special meanings to the compiler.
- We use the escape character \and type \" to escape the way the normal way it is interpreted by the compiler.

Escaping a Problem

- How do you think we would print out \
- We would need to use \\
- if we also wanted a newline character we could do the following

```
printf("\\\n");
```

Compiling and Running Hello World

To run our hello world program we need to compile it first.

At home you may need to use gcc (GNU Compiler Collection, formerly GNU C Compiler) for this task.

In our cse accounts we can use dcc (Direct C Compiler) instead, as dcc is more helpful with error messages.

Compiling and Running Hello World

The simplest use of the compiler would be:

```
gcc helloWorld.c
```

which produces the file a.out.

We could then run the program by typing

```
./a.out
```

Or if we were logged into a cse account we could use

```
dcc helloWorld.c
```

and run it in the same way with

```
./a.out
```

Compiling and Running Hello World

a.out is not a great name. It is a better idea to give your executables their own meaningful names.

To do so you need to use the -o flag, followed by the name you have chosen.

```
gcc helloWorld.c -o helloWorld
```

We could then run the program by typing
./helloWorld

You could do the same thing with dcc instead by typing.

```
dcc helloWorld.c -o helloWorld
```

Compiling and Running Hello World

To get more help from gcc we want to turn on extra tough error checking so we would compile using something like:

The command: gcc -Wall -Werror -O -o helloWorld helloWorld.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

There is no need to use these extra flags with dcc

Compiling A Program

- Create a file named `hello.c` containing the program
  `gedit hello.c`
- Once the code is written and saved, compile it:
  `dcc hello.c`
- Run the program:
  `./a.out`

```
$ gedit hello.c &
$ dcc hello.c
$ ./a.out
```

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

Coding Style

- The code examples we give you in this course are examples of good style.
- The course C style guide is available from the course website.
- You should always adhere to it otherwise you will lose style marks for your assignments.
- We will refer to it throughout the course when we learn new C constructs

The Task of Programming

Programming is a construction exercise.

- Think about the problem
- Write down a proposed solutions
- Write a scaffold of the program in plain English so you solve the problem logically before programmatically
- 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