What is computer science?
The study of data and how it is represented, manipulated and transformed into useful information.
What is a computer?

A machine that manipulates data according to instructions.
History of Computers

Computer Hardware: 1835
Analytical Engine designed by Charles Babbage.

The first Coder: 1835
Ada Lovelace - mathematician who wrote the first programs.
What makes a working computer today?

- Hardware
  - Processor (CPU), Memory (RAM), Storage (HDD)
  - Input Devices (Keyboard, Mouse etc)
  - Output Devices (Screen, USB Ports etc)
- Operating System (Linux, Windows)
- Software (games, utilities, applications etc)
• A computer has a limited repertoire of operations, but carries them out extremely quickly and with astounding reliability and precision.

• A home computer carries out more than 100,000,000 operations per second

• 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.
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.
What is an operating system (OS)?
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. Allows you to program without knowing of hardware details.
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.

Allows you to program without knowing of hardware details

- e.g., Windows, OS X, Unix, Linux, IOS
- long history; many innovations come from Unix
- Unix first widely used multi-user and multi-tasking OS
- Linux, Android, FreeBSD - descendents of Unix
Using CSE’s Computing Resources

Our labs are running Linux, a multi-user operating system, with the basic tools necessary to get started and you will have your own account on the cse machines, with your

- unique username (zid) and
- password (zpass).

Logging in to your cse account, from a lab machine or home (using VLAB) will give you access to your files and settings. These are not to be shared with anyone else.

For COMP1511, we also need a text editor (e.g., gedit)
Why don’t we program in languages humans use, like English?
Why don’t we program in languages humans use, like English?

- it is too informal
Programming Languages

Why don’t we program in languages humans use, like English?

• it is too informal
• it is too big
Why don’t we program in Machine Code that computers understand?

- We can! But it looks like this:
  0100101001000010001111100011000011101

Trivia: Many years ago, programs were written by punching holes into cards to represent binary codes.
Programming Languages

Why don’t we program in Machine Code that computers understand?

• We can! But it looks like this:
  010010101001000010001111100011000011101

• Error prone for humans but still used occasionally
Programming Languages

Why don’t we program in Machine Code that computers understand?

- We can! But it looks like this:
  0100101001000010001111100011000011101
- Error prone for humans but still used occasionally
- Not portable from one machine to another
Programming Languages

Why don’t we program in Machine Code that computers understand?

- We can! But it looks like this:
  0100101001000010001111100011000011101
- Error prone for humans but still used occasionally
- Not portable from one machine to another
- Trivia: Many years ago, programs were written by punching holes into cards to represent binary codes.
Programming Languages

So we invent a programming language that:

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

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, the hardware, can execute directly.

Warning: When we program we need to make sure we give exact instructions. Website: https://www.youtube.com/watch?v=cDA3_5982h8
So we invent a programming language that:

- is small

Warning: When we program we need to make sure we give exact instructions. Website: https://www.youtube.com/watch?v=cDA3_5982h8
So we invent a programming language that:

- is small
- is formal (syntax and grammar)

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 (the hardware) can execute directly.

Warning: When we program we need to make sure we give exact instructions.

Website: https://www.youtube.com/watch?v=cDA3_5982h8
So we invent a programming language that:

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

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.
Programming Languages

So we invent a programming language that:

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

How does a computer then run the program code?

Website: https://www.youtube.com/watch?v=cDA3_5982h8
Programming Languages

So we invent a programming language that:

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

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 ie the hardware can execute directly.

Warning: When we program we need to make sure we give exact instructions.

Website: [https://www.youtube.com/watch?v=cDA3_5982h8](https://www.youtube.com/watch?v=cDA3_5982h8)
Programming Languages

So we invent a programming language that:

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

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.

Warning: When we program we need to make sure we give exact instructions. Website:

https://www.youtube.com/watch?v=cDA3_5982h8
Why C?

- C is a relatively small language
- Gives the programmer great (low-level) control
- Produces fast code.
- Many libraries and resources
- Widely used for writing operating systems and compilers as well as industrial and scientific applications
Navigating Linux

The main interface to Linux is 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`
• **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)
  ▶ `ls -h` (shows file sizes in human readable format eg 3K 234M 2G)

  ▶ Can combine options. For example, `ls -la ls -lh`
- `pwd` *directoryName*
- `print` **working** directory
- a directory is like a folder in windows
- the working directory is the directory that you are currently in
mkdir

- `mkdir directoryName`
- **make** (create) a new directory called `directoryName` in the current working directory
- 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
Linux 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.
Creating and Editing a File

We recommend using **gedit** as your editor for your c programs

To use gedit to create or open a c file for editing type the following into your linux terminal

```
gecit myFile.c &
```

If you leave out the & your terminal will just sit there waiting for you to close gedit before you can type other commands. This is annoying, so usually we add the &
Hello World

Open a text editor to create a c file name helloWorld.c by typing

gedit helloWorld.c &
Hello World

Type the following program into it and press save

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

#include <stdio.h>

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

    return 0;
}

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

If you are not logged into your cse account, you may need to use gcc (GNU Compiler Collection, formerly GNU C Compiler) for this task.

If we are logged into our cse account we can use dcc 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 -o helloWorld helloWorld.c

We could then run the program by typing

./helloWorld

You could do the same thing with dcc instead by typing.
dcc -o helloWorld helloWorld.c
To get more help from gcc we want to turn on extra tough error checking so we would compile using something like:
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`
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
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
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
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
Dissecting 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 header 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
What does it all mean?

- `//`, a single line comment, use `/* */` for block comments
- `#include <stdio.h>`, import the standard I/O library
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
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`
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
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
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
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
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");
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.
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 Again

How do you think we could print out \\

We would need to use \ and if we also wanted a newline character we could do the following
Escaping Again

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

printf("\\\\n");
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
What is good 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
The Task of Programming

Programming is a construction exercise.

- Think about the problem
- Write down a proposed solution
The Task of Programming

Programming is a construction exercise.

- Think about the problem
- Write down a proposed solution
- Break each step into smaller steps
The Task of Programming

Programming is a construction exercise.

- Think about the problem
- Write down a proposed solution
- Break each step into smaller steps
- Convert the basic steps into instructions in the programming language
The Task of Programming

Programming is a construction exercise.

- Think about the problem
- Write down a proposed solution
- 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
The Task of Programming

Programming is a construction exercise.

- Think about the problem
- Write down a proposed solution
- 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
The Task of Programming

Programming is a construction exercise.

- Think about the problem
- Write down a proposed solution
- 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