

Programming in C (++) and R, MATH10017

Song Liu (song.liu@bristol.ac.uk)

GA 18, Fry Building,

Microsoft Teams (search "song liu").

When and Where

Type	Time	Location
Lectures	10-11AM, Mondays	FRY BLDG LG.20
Labs*	9-11AM, Wednesdays	FRY BLDG LG.21 PC
Tutorials	5-6PM, Fridays	FRY BLDG LG.21 PC

[https://www.bristol.ac.uk/timetables/html.html?
unit=MATH10017](https://www.bristol.ac.uk/timetables/html.html?unit=MATH10017)

*Labs are two hours!

Key Objectives

Upon completion of this unit you should:

1. *Understand* the workflow of computer programming and appreciate computer as a data processing tool.
2. *Program, debug, document* and *test* basic algorithms in C(++) and R, with appropriate coding paradigms.
3. *Decide* which programming language to use when faced with a computing task.

How This Unit is Structured?

- TB1: C Programming Language
 - Some aspects of C++ Programming Language*.
- TB2: R Programming Language

*Some of C++'s features are designed for software engineering, not programming.

What are the Assessments?

- Written Exam (50%) at the end of TB2.
- Two Programming Coursework (25% + 25%).
 - One per TB.
 - Submit by the end of each TB.
- Non-assessed homework each week after lecture.
 - **Do not skip homework.**
 - These homework are designed to build up skills required for the final project.
 - Reuse some of the code you've written for these homework, to make your project work easier.

Your Week

- **Monday:** Attend lecture.
 - Pay attention to the homework released after lecture.
 - Write *pseudo code* of your homework.
- **Wednesday:** Go to lab.
 - Convert your *pseudo code* into actual C/R code.
 - Get help from TA/me on your homework if necessary.
 - Help each other*.
- **Thursday:** Submit your homework by 5pm.
 - Read pre-tutorial materials.
- **Friday:** Attend tutorial and engage in discussions.
 - Get feedback of your homework, ask questions.

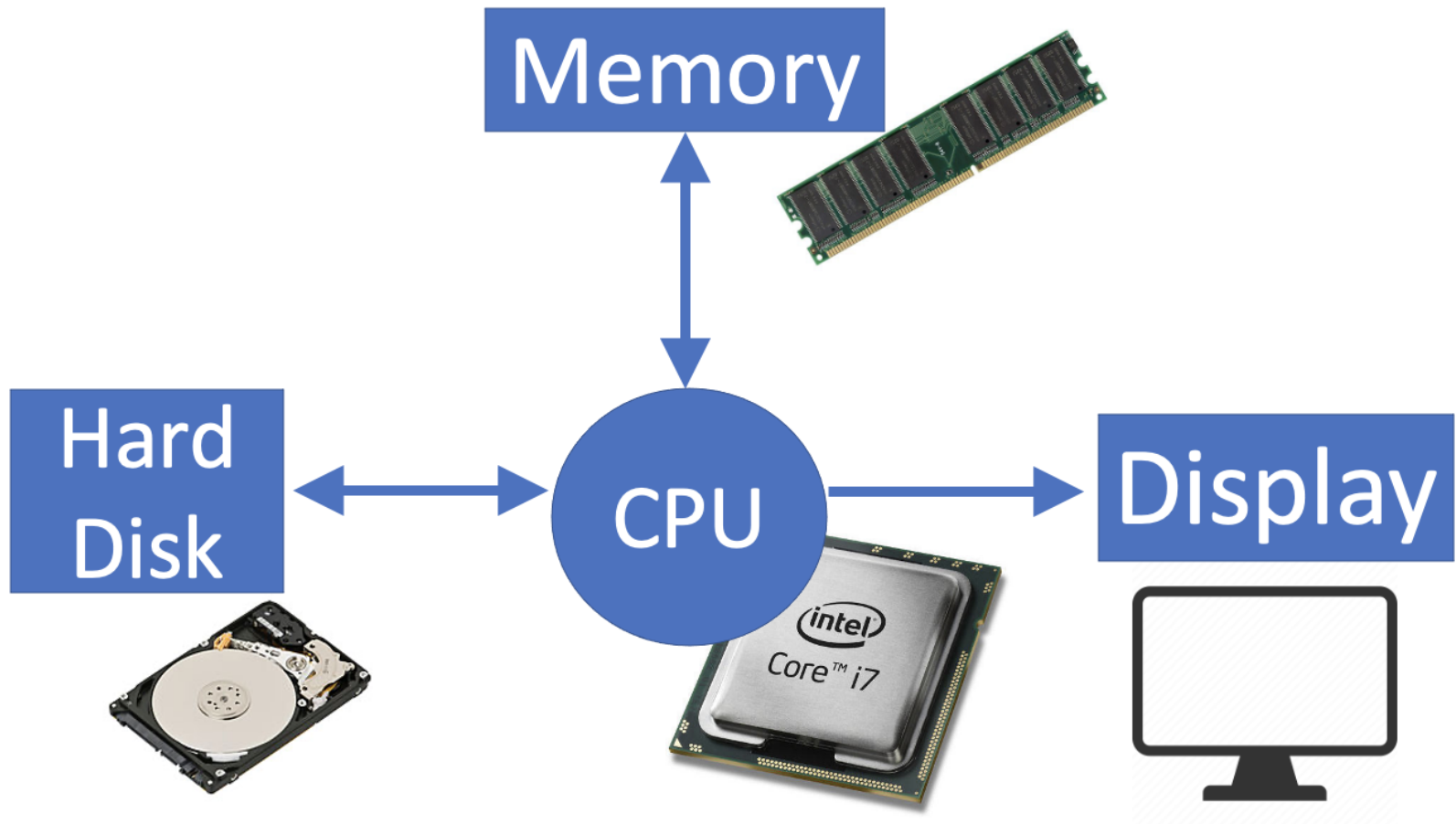
Mental Health

1. Get help early!
2. Live a structured live.

Plagiarism

- Read [this guideline](#) on Plagiarism very carefully.
- Plagiarism is a serious academic offence.
- Tips:
 - Never copy and past from the internet.
 - Learn from other people's idea, rather than copy it.
 - Do NOT ask other people to test your code. It is your own duty to make sure your code works.
- We have many ways to detect Plagiarism.

How does Computers Work?



von Neumann architecture

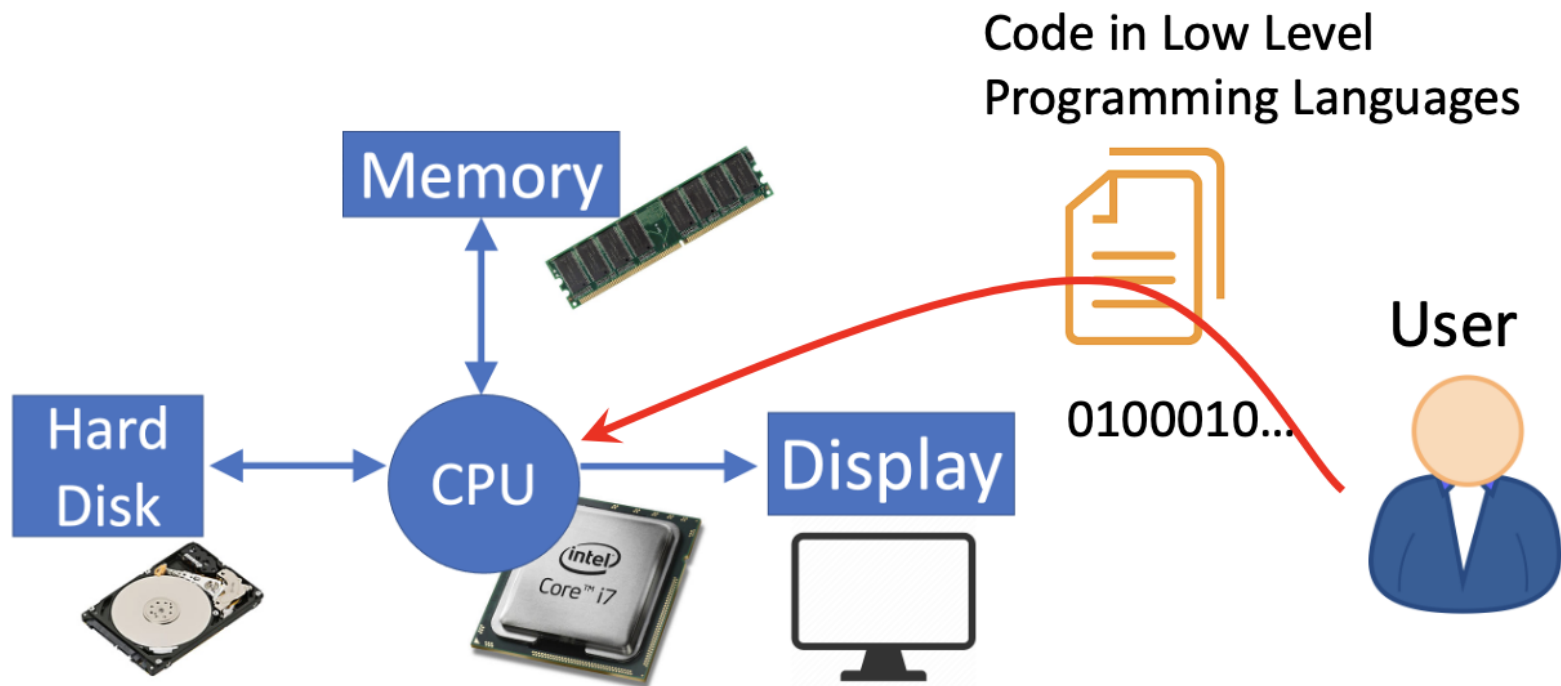
- **Central Processing Unit (CPU)**
 - Performs computational tasks.
 - Controls Input/Output (IO) devices.
 - Maintains data stored in the memory.
- **Memory**
 - Stores program/data being used by CPU temporarily.
- **IO Devices**
 - Hard disk
 - Display
 - Camera
 - Touch Screen, etc.

What is Programming?

- Programming = **writing a list of instructions to be executed on the CPU.**
- The list of instructions is called the "code".
- Programming is also called "coding".
- Programmer is also called "coder".
- The language used to write the code is called **programming language.**

Low-level Programming Language

- Coder can program in **machine code**.
- Then the code can be **directly executed** on the CPU requiring no (or very little) translation.
- Machine code (and its more human friendly variants) are referred to as "Low-Level Programming Languages".



Low-level Programming Language

- Advantages of Low-level Programming Language:
 - gives coder total control of hardware.
 - can be efficient since it needs no translation (You talk to the computer using its native language!).
- Disadvantages of Low-level Programming Language:
 - can damage the hardware if the coder is not careful.
 - is difficult to learn and read (machine instructions are usually very different from human languages).
 - only works on a specific computer architecture (e.g. x86 or ARM). Thus the code is not "portable".

Low-level Programming Language

Example code printing a message "Hello, World" using x86 CPUs

```
global start
section .text
start:
    mov     rax, 0x02000004
    mov     rdi, 1
    mov     rsi, message
    mov     rdx, 13
    syscall
    mov     rax, 0x02000001
    xor     rdi, rdi
    syscall
section .data
message:
    db      "Hello, World", 10
```

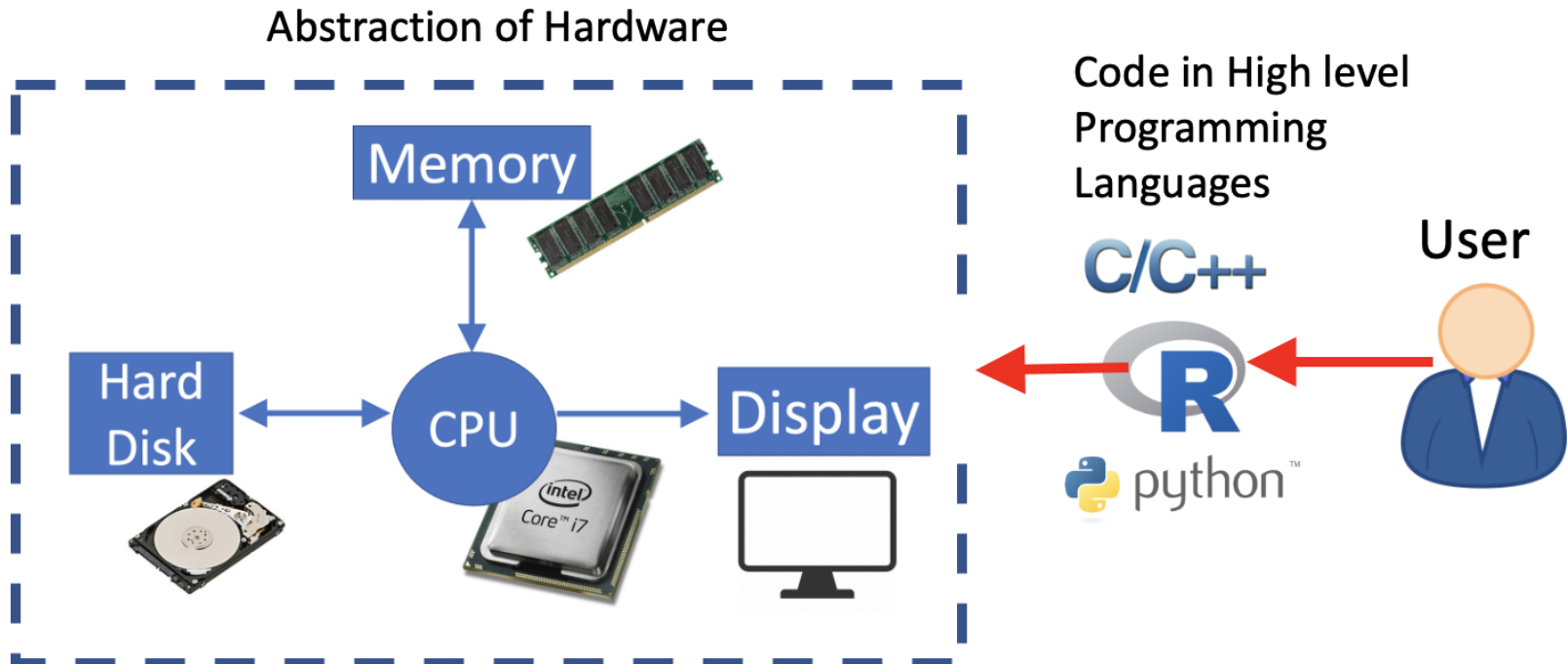
Source

High-level Programming Language

- Coder can program in a more natural language, which is then **translated** into machine code.
- This translation process is called "compilation" and the software that performs this translation is called "**complier**".
- This kind of languages are called "High-level Programming Languages". For example,
 - C, C++
 - Python
 - Java
 - MATLAB
 - R...

Abstraction of Hardware

- Since your code is not executed on the CPU "as is", high-level language provides **an abstraction of hardware**.
- Coder interfaces with this abstraction rather than directly program for the underlying CPU and hardware.



Abstraction of Hardware

- Not allowing the user to directly program for the hardware sounds restrictive but also has many advantages:
 - The code is "portable", i.e., the coder needs not to rewrite their code for different CPU architectures, as all CPU architectures share the same abstraction.
 - The abstraction hides many cumbersome details of hardware management, so the coder can focus on their computational task.
 - Not allowing the program to have direct access to the hardware would enhance the security of the system.

Abstraction of Hardware

- However, different programming languages may provide different **levels of abstraction**.
- Some high-level programming language allows you to directly manage computational resources to some extent.
- Therefore, you can again categorize high-level programming languages into "relatively high-level" and "relatively low-level".
 - C/C++ is regarded as "relatively low-level".
 - Python/R/MATLAB are regarded as "relative high-level".

Stack Exchange: Why do some programmers categorize C, Python, C++ differently?

High-level Programming Language

- Advantages of High-level Programming Languages
 - Close to human language, easy to learn/read.
 - CPU architecture independent, a.k.a., "Portable".
- Disadvantages of High-level Programming Languages
 - Less efficient as the code requires "translation" before it can be executed on CPU.
 - Cannot directly communicate with hardware: the coder has to interface with *the abstraction*.

Example Code: C

Example C code for printing "Hello World!" on almost all CPUs.

```
//filename: main.c
#include <stdio.h>

void main(){
    printf("Hello World!\n");
}
```

Compilation:

```
gcc main.c -o main.o
```

Execution:

```
./main.o
Hello World!
```

Dissecting C Code

```
//filename: main.c
#include <stdio.h>

void main(){ printf("Hello World!\n"); }
```

- `//filename: main.c` : **Comments**. Readable explanations or annotations in the source code. It is ignored by compiler.
- `#include <stdio.h>` : **Preprocessing command**. Instructs the compiler to perform pre-processing before the actual compilation.
- `void main(){...}` : **Function**. Contains list of statements.
- `printf("Hello World!\n");` : **Statement**. The actual command to be carried out by the computer.

The gcc Compiler

```
gcc main.c -o main.o
```

- `gcc` : [GNU C Compiler](#). An [open-source](#) C programming compiler, available on Linux and MacOS.
- `main.c` C code file, as the input.
- `-o main.o` "main.o" as the output executable file (a file contains machine code, ready to be executed by CPU). On windows, use `main.exe` instead of `main.o`.

Example Code: R

Example R code for printing "Hello World!" on all platforms that run R programming language.

```
#filename: hello.R  
print("Hello World!")
```

```
RScript hello.R  
  
[1] "Hello World!"
```

There is no explicit complication. R command line (RScript) tool reads the R code line by line and automatically translates them into executable codes in real time.

More on R in the Next TB.

Development Environment

Development environment refers to the collection of software you need to write, debug (more on this later) and test your code.

To do C programming, you need two things at least:

- **Text Editor:** As C code are text files, you need a text editor to write and organize these code files.
- `gcc` : The compiler to translate your C code into machine code. It comes with most Linux installations.

This unit recommends using [Visual Studio Code](#) as your code editor. It comes with many useful features for beginners (such as syntax highlighting, code autocomplete, etc.).

Resources about C Programming

1. [The C Programming Language](#), Brian Kernighan and Dennis Ritchie.
 - i. [An interview of Brian](#) on the history of C.
2. [Wikibooks: C Programming](#).
3. [COMS10008: Imperative Programming](#)
 - i. An excellent unit used to be taught by Dr. Ian Holyer in Computer Science department.
4. [CS50](#) is Harvard University's introductory course to computer science and the art of programming.

Conclusion

1. Programming = **writing a list of instructions to be executed on the CPU.**
2. There are two types of programming language: High level programming and low level programming.
 - i. Pros and cons.
3. Two things you need to do C programming language: Text editor + Compiler!

A Pre-configured Development Environment for Fry PC Lab

- Download the [Lab Pack](#) and unzip.
- Double click "lab0.bat".
- Visual Studio Code should open. You should be presented with a "hello, world" C code example.
- Of course, you are free to configure your own development environment.

printf function

Builtin function `printf` displays "format string".

- `printf` is a shorthand for *print_formatted*.
- This function is provided by the OS and is declared in file `stdio.h`.
- To use this function, you need to add `#include <stdio.h>` at the beginning of your C code.

```
printf("FORMAT_STRING", VARIABLE1, VARIABLE2 ...);
```

- `FORMAT_STRING` can contain "format specifiers".

printf function

- `printf` will replace all "format specifiers" in the `FORMAT_STRING` with supplied `VARIABLE` before display.
 - `printf("My name is %s %s. \n", "Song", "Liu");`
 - Prints out `My name is Song Liu.`
 - `"%s"`: string specifier, tells computer to expect "string" type variable at this location.
 - `"\n"`: [ASCII code](#) for "new line".
- [Read this manual](#) for a list of all possible "format specifiers" and simple examples.

Homework 1, Hello World

Setting up your Development Environment.

Test your set up with the Hello World c example above.

Homework 2, `printf`

Now, try to modify the "Hello World" example code, and accomplish the following tasks:

1. Print out your UoB ID as a string.
2. Print out your UoB ID, treating the numeral part of your UoB ID as an integer.
3. Print out the outcome of $1/3*3$.
 - i. Does it work?
 - ii. Can you guess why it does not work?
 - iii. Hint, try printing out the outcome of "1/3" and infer what went wrong.

Homework 3, `printf`

4. (submit) Print out the following table using `printf`.

```
10 Fahrenheit = -12.22 Celsius,  
20 Fahrenheit = -6.66 Celsius,  
...  
100 Fahrenheit = 37.77 Celsius.
```

- Note: Fahrenheit temperatures are all integers, but Celsius are all decimal numbers, and only 2 characters should be displayed after the decimal point.
- You **must** make use of the conversion formula:
$$(F - 32) \times 5/9 = C$$
- Hint: `printf("%.2f", 1.0/3.0)` prints out `0.33`.

Homework 4, `printf`

5. (submit) Print out the following triangle using `printf`.



- Hint: `printf("\\")` will print out `\`.

Homework 5, `printf`

4. (Challenge) `%c` specifier is for printing a single character and single character is wrapped in `' '` in C Programming Language.
- i. `printf("%c\n", 'A')` prints `A`.
 - ii. try `printf("%c\n", 65)` and `printf("%d\n", 'A')`, what do you see? why does it work like this?
 - iii. Guess the output of `printf("%c\n", 'A'+1)`.
Validate your guess.
 - iv. Can you print out an emoji 🤔? Hint: the "numeral code" for 🤔 is four consecutive numbers, 240 159 152 137.

Feel free to look up on internet! (Do not copy and paste, of course.)