

# Programming in C (++) and R, MATH10017

Song Liu ([song.liu@bristol.ac.uk](mailto:song.liu@bristol.ac.uk))

GA 18, Fry Building,

Microsoft Teams (search "song liu").

Twitter: songandylIU

# When and Where

Type	Time	Location
Lectures	3PM - 5PM, Mon	G44, PHYS
Labs	Tue 1PM - 3PM	FRY BLDG LG.21 PC
Tutorials	Every week	FRY BLDG LG.21 PC

<https://www.bristol.ac.uk/timetables/html.html?unit=MATH10017>

\*Labs and tutorials and will have attendance checking.

\*For international students, these are your visa checkpoints.

# 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, first half: C Programming Language
- TB1, second half: R Programming Language.

# What are the Assessments?

- Written Exam (50%) at the end of TB1.
- Two Programming Coursework (25% X 4).
  - CW1, 3 NOV
  - CW2, 8 DEC
- Non-assessed homework each week after lecture.
  - **Do not skip homework.**
  - These homework are designed to build up skills required for the coursework.
  - Reuse some of the code you've written for these homework, to make your coursework easier.

# Plagiarism and Collusion

- Read [this guideline](#) on Plagiarism very carefully.
- Read [this guideline](#) on Collusion very carefully.
- They are serious academic offenses.
- **Tips:**
  - Never copy and paste from the internet/CHATGPT.
  - Learn from other people's idea, rather than copy it.
  - We encourage discussion but you should do your homework independently.
- We have many ways to detect Plagiarism and we caught several incidences in the past.

# Policy of AI Use

Read the [Policy of AI Use](#).

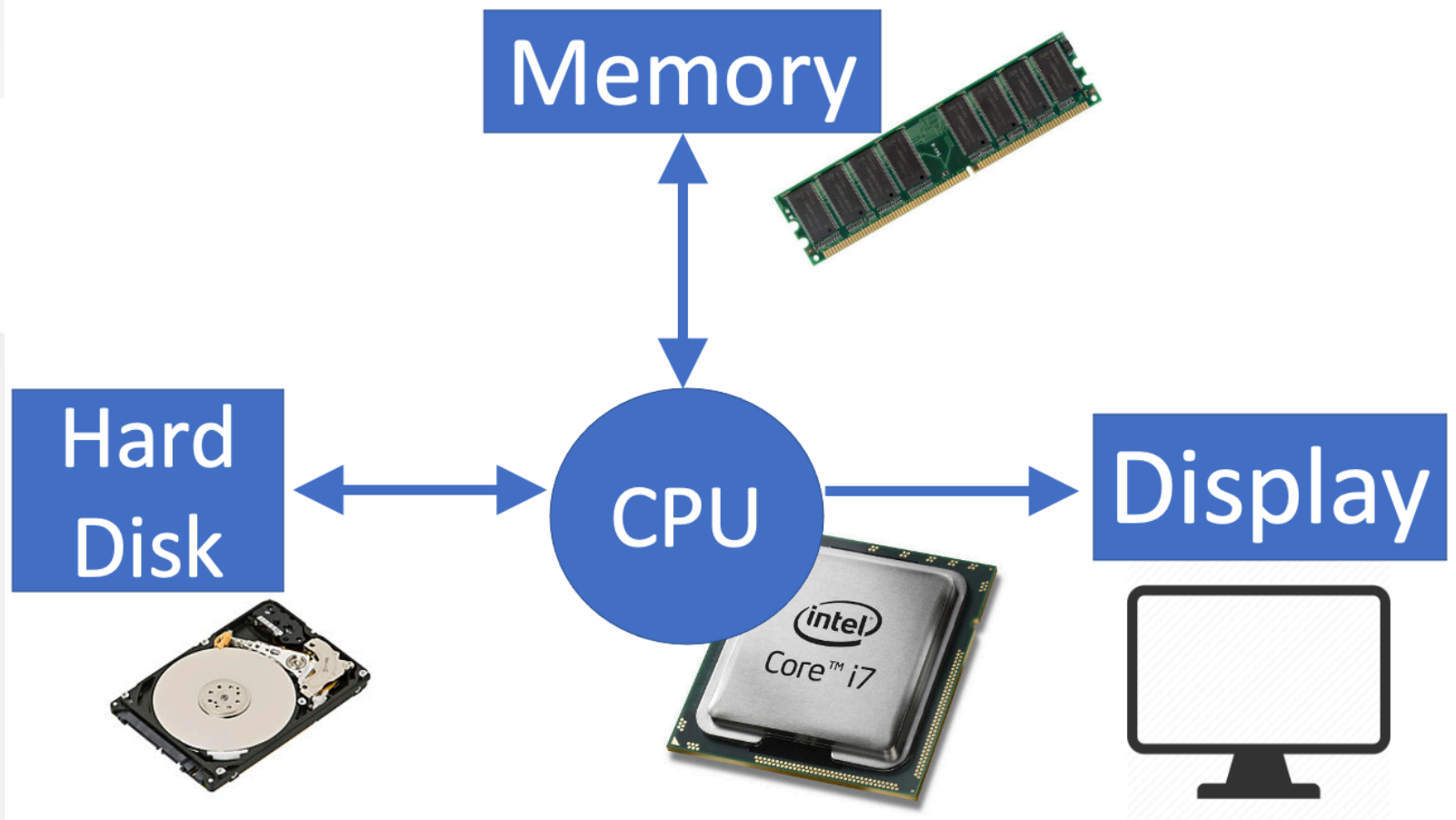
In short, in the assessment, the use of AI is prohibited.

In the exam, you will write code on the paper.

# The foundation of computing



# An Idealized Computer



# von Neumann Architecture

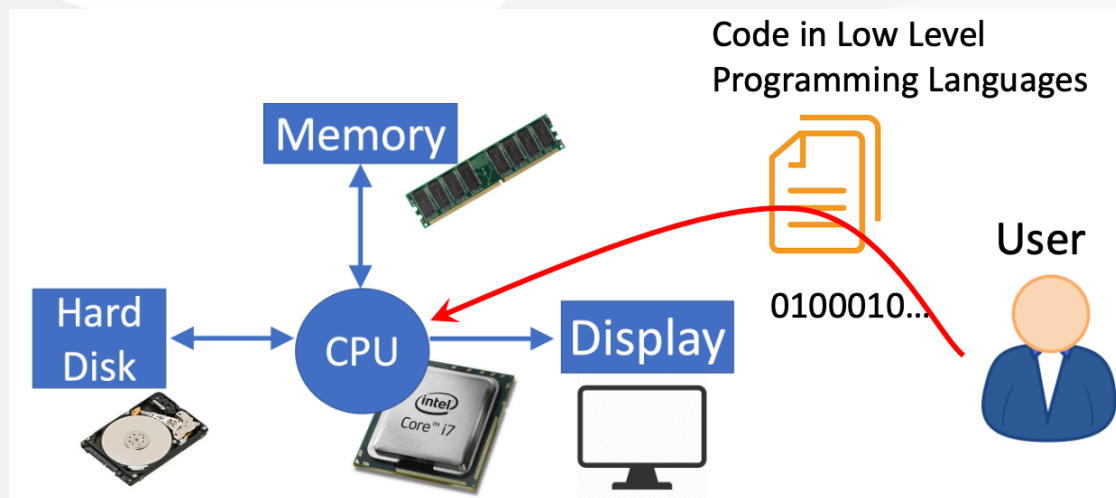
- **Central Processing Unit (CPU)**
  - Performs computational tasks.
  - Controls Input/Output (IO) devices.
  - Maintains data stored in the memory.
- **Memory**
  - Stores program/data being used by CPU temporarily.
- **IO Devices**
  - Hard disk
  - Display
  - Camera
  - Touch Screen, etc.

# 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 cpu architecture (e.g. x86 or ARM). Thus the code is not "portable".

# Low-level Programming Language

Example code printing a message "Hello, World" on display on x86 CPUs

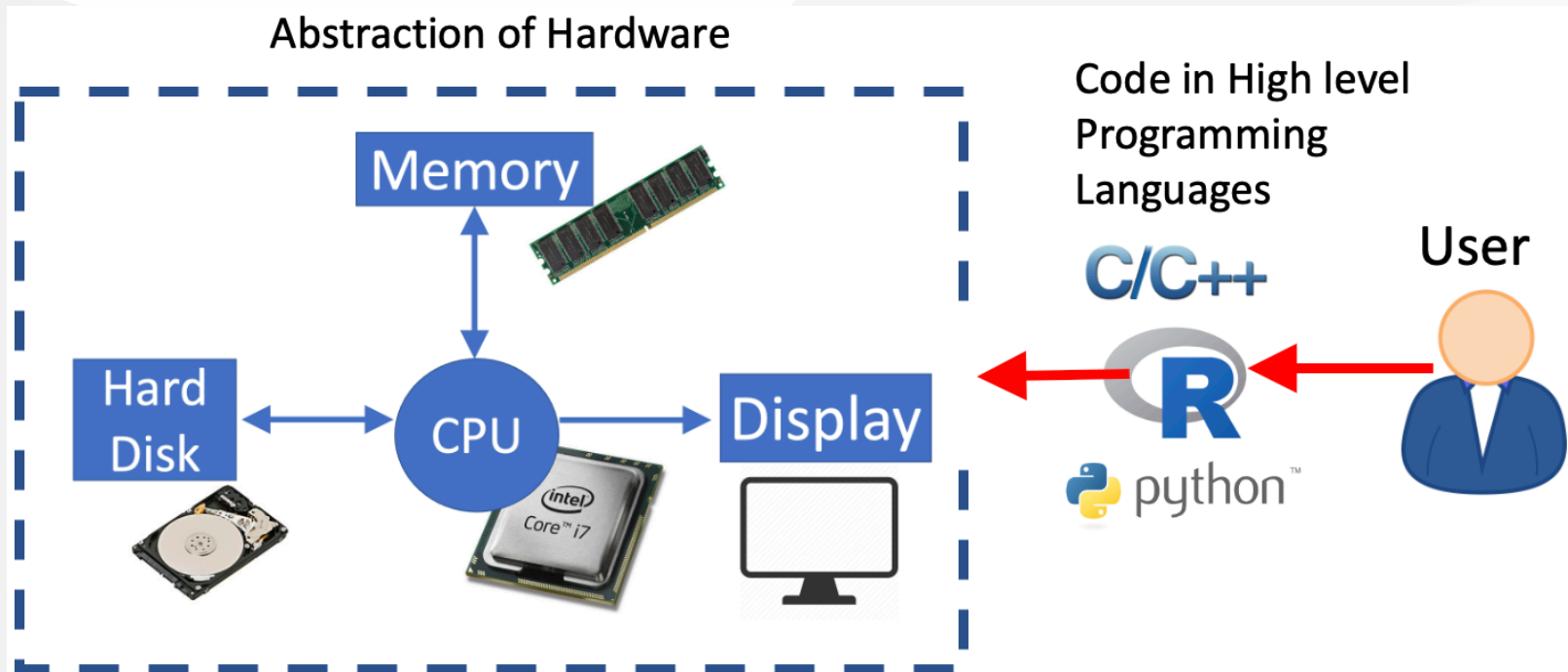
```
global start
section .text
start:
    mov     rax, 0x02000004
    mov     rdi, 1
    mov     rsi, message
    mov     rdx, 13
    syscall
    mov     rax, 0x02000001
    xor     rdi, rdi
    syscall
section .data
message:
    db      "Hello, World", 10
```

# 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 your screen.

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

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

Compilation (in command line):

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

Execution (in command line):

```
./main.exe
Hello World!
```

# Dissecting C Code

```
//filename: main.c
#include <stdio.h>
void main(){
    printf("Hello World!\n");
}
```

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

# The `gcc` Compiler

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

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

# Example Code: R

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

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

To execute, in command line, run

```
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.

# Development Environment

Development environment refers to the collection of software 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

Download the slides from Github before clicking the links!

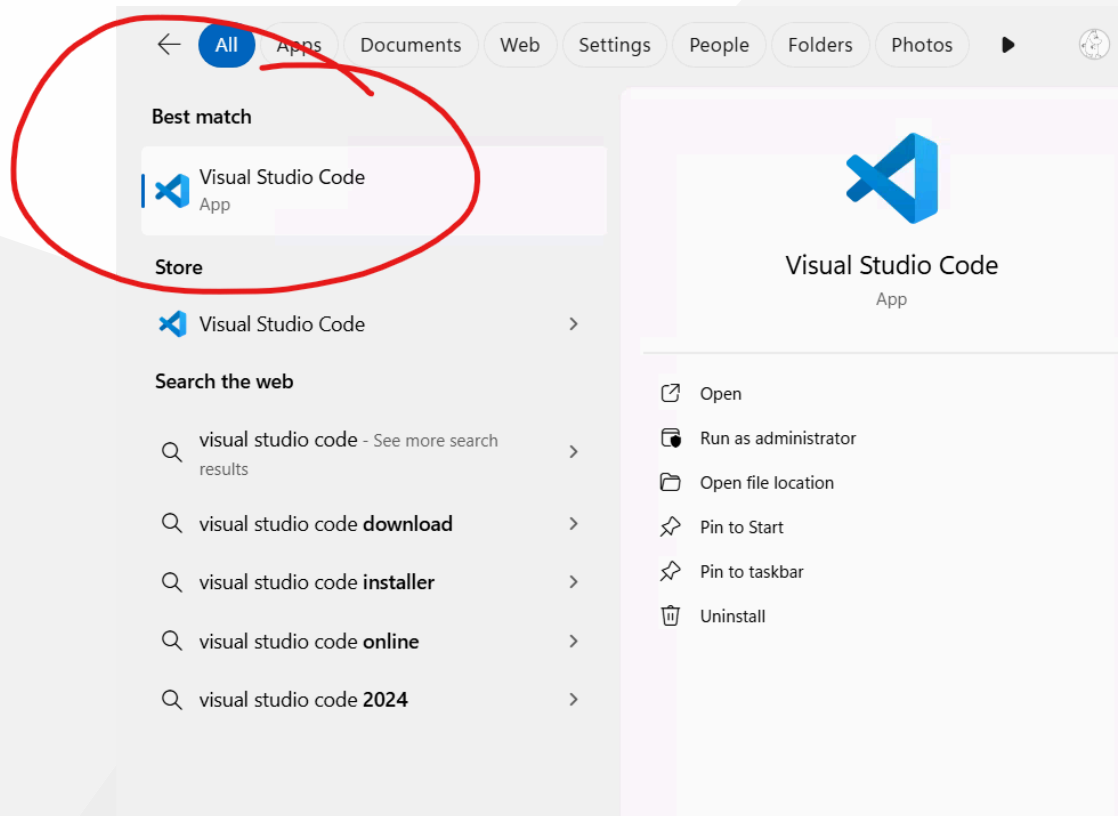
1. [The C Programming Language](#), Brian Kernighan and Dennis Ritchie (The creators of C). The Bible.
  - i. [An interview of Brian](#) on the history of C.
2. [Wikibooks: C Programming](#), for quick references.
3. [COMS10008: Imperative Programming](#)
  - i. An excellent unit use to be taught by 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 for C programming: Text editor + Compiler!

# A Pre-configured Development Environment for Fry PC Lab

- In our unit, we use "Visual Studio Code".



# A Pre-configured Development Environment for Fry PC Lab

- We cannot provide support for running VSCode on your own computer, due to limited time in lab.
  - If you want to use your own computer, [set up VScode](#) before the lab.
  - Note, this set up is a complicated procedure on Windows and Macs.

# Homework 0.1, Hello World

- Download the provided code for the lab (See the Github page) and save it to a personal folder.
- Open Visual Studio Code, "open folder", select the folder which contains the file you have just downloaded.
- **Compile** the provided "hello world" program.
  - Press `ctrl + `` to bring out the command line.
  - Use `gcc` to compile the code, as shown in the slides.
  - Hint: You can hide the command line by pressing `ctrl + `` again.
- **Execute** and Observe the program output.

# Homework 0.2, Hello World

- Change the C code in Visual Studio Code, so that it prints out your information, not mine.
  - Your student ID is the stuff before your Bristol email address, should be something like "ab231234".

For example,

```
My name is [YOUR NAME].  
I am from [YOUR HOMETOWN].  
My student ID is [STUDENT ID].  
My favourite food is [YOUR FAVOURITE FOOD].
```

- **Re-compile, execute** and observe the output.
- Does the program output change?

# Homework 0.3, Hello World

- Modify the C code in Visual Studio Code, so that it prints out your information in the following order:
  - NAME
  - STUDENT ID
  - FAVOURITE FOOD
  - HOMETOWN
- Make sure your code compiles, runs and produces intended output.

# Homework 0.4, Hello World

- Write a short comment in your code, listing
  - Pros and Cons of high level programming language
  - Three key components of von Neumann Architecture
  - Hint, you can use

```
/*  
to write multi-line  
comments.  
*/
```

- Make sure your code compiles, runs before submission!
- Submit the C code ( `YOUR_STUDENT_ID.c` ) [here](#).
  - Your student ID should be something like "ab231234".



# 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.