How to write and run a C code?

1. We write our C code using a code editor and save it in a file in the hard disk.
2. We use a C compiler to translate the human understandable (not all humans understand C, so we need to learn C first) C code into machine understandable machine code. The translated machine code is then saved in the form of an executable file in the hard disk.
3. We execute the machine code. The processor copies the instructions in the executable file into the main memory (RAM) and then starts to execute each instruction line by line.

   A key thing to remember is that memory is linear. All the instructions are stored one after the other, and all the data used in the code is also stored one after the other. It will play a very important role in understanding structs, unions and pointers later.

What are the components required to write and run a C code?

1. A computer?
2. A text editor / IDE of your choice
   Vim is the best, notepad is the simplest, VS code lies somewhere in between.
3. A C compiler
   We will use gcc, uninstall turbo C++ if you installed it by watching old youtube videos. Clang is OK but it requires the standard library (we will discuss about that later) of gcc in windows anyway. MinGW is the windows version of gcc, so install it if you are in windows, alternatively you can make your life a bit easier (or more difficult?) by using linux or WSL.
4. Terminal / command prompt
   We will be using the gcc command in the terminal to compile our C code, and we will also use it to run the executable and view the output.

How to write the simplest C code?

1. The main() function is the starting point of a C program. If the compiler does not find it, it's not going to generate the machine code. So, we need to define it at least. We will understand the code better when we learn about functions, but for now just learn the syntax.
   ```
   int main() {
         return 0;
   }
   ```
2. It is the simplest code but it does nothing, so to print hello world in the terminal screen, we add 2 extra lines.
   ```
   #include <stdio.h>

   int main() {
         printf("hello world\n");
         return 0;
   }
   ```
3. This code also does nothing much, but at least it prints hello world and a newline character (\n, so that the next line does not start immediately after hello world).

4. printf is another function, but unlike main() which we created, printf has been created by other people. It takes a text (which it will print) as an argument inside the round brackets, and the text must be enclosed within double quotes (we will learn more about it when we learn about identifiers and strings).
5. Since printf has been created by other people, we need to include the file in which it has been created so that our program can use it. We do that using the #include preprocessor (we will learn a little more about preprocessors in a while). #include is the preprocessor and within the angular brackets (<>) is the filename where the printf function exists (not really but we understand that completely when we learn about declarations and definitions).

What are preprocessors?

1. Preprocessors are something that are executed before gcc compiles the real code. The start with # and are more or less like a word processor which makes some changes in the code before the translation starts.
2. #include <x> tells gcc to copy-paste the contents of the file x in place of the preprocessor. How does the compiler know where x is? It searches for the file in the installation directory of gcc, if it is not found, it will throw an error.
3. If you are using VS code and the C/C++ extension pack, you can hold Ctrl and click on the filename inside the angular brackets to go to the file itself, but make sure not to change anything there, you won't want to fix a very complicated code written by other people! This same feature works for any other identifier. If you Ctrl click on printf for example, you will be taken to the printf function itself.
4. There are many other preprocessors, but for now we will only learn about #include (we will be learning about #define someday, because it is very important too).

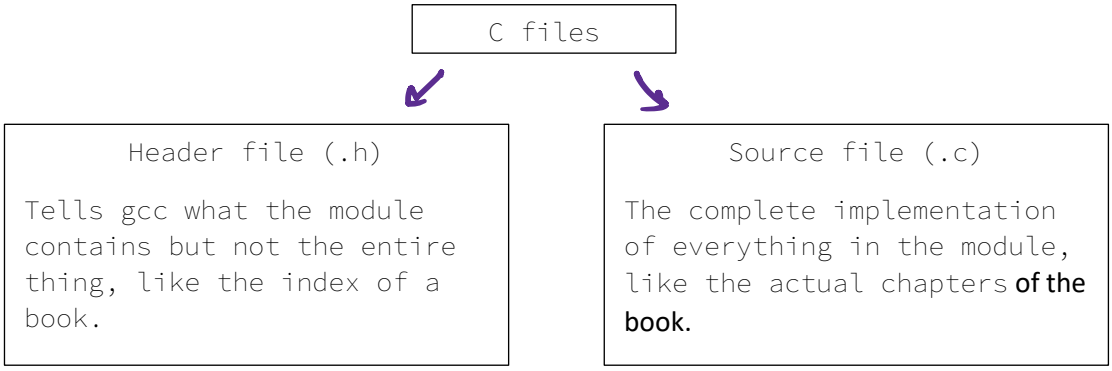
How to compile and run our C code?

1. Compiling our C code won't be as simple as clicking a button (we won't be using code runner in VS code, however tempting it may look).
2. Open a terminal window and go to the folder where your code is stored. Alternatively, you can browse to the folder where your code is stored in the file explorer, then right click and select the open in terminal (or something similar) option in the context menu that opens up.
3. Then type gcc followed by the filename you want to compile. Additional tip: turn on extension names from the view tab in windows file explorer, it is very helpful to see the file extensions, the file extension must be .c file for gcc to treat it as a C file (if you don't want to add a couple of complicated flags to tell gcc manually that is).
4. Then you hit enter and wait for a second or two, you will be showered with a lot of error messages or nothing if everything went fine. Another tip: use Tab key to auto complete filenames instead of writing big names manually.
5. If everything went fine, you will find an a.exe (if windows) or an a.out file. These are the executable files, translated by the compiler.
6. You can execute it by using the command ./a.exe or ./a.out in the terminal.

What are header files and source files?

1. The stdio.h file in the #include preprocessor is a header file (.h means header). On the contrary, the file we are writing (say hello.c) is a source file (.c means C source file).

2. For now, we may think that the header files contain the starting points of the functions and other stuffs created by other programmers. The complete thing is again stored in another source file or in some other form.



3. No one really learns this at the first day of learning C but I thought it is important if we want to ever split our code into multiple files or just to know why we are including .h files in a .c file.
4. It seems like pretty redundant because we never used different types of files for modules and code in python. Everything was .py, but this distinction is needed because of the primitive nature of C. Unlike the import statement in python, the #include only copy-pastes the content of a file. So, if the complete source file is a thousand lines of code, it is better to just copy only a hundred line of header file and then link the real implementation later (we will discuss linking a little in the next session).
5. The minGW version of stdio.h contains 2000+ lines of code in the version of gcc I was using during the session. 2000+ just for the header file, the real implementation will contain a lot lot lot more code than 2000, and that much copying would definitely take a huge toll on the processor.
6. There are more advantages of this distinction, but we will discuss it some other day.

What to do after going back home?

1. There is a gcc flag which can be used to generate the preprocessed file. Use that flag to get the preprocessed C file and note the differences. Don't just copy paste the contents of stdio.h in place of the preprocessor, the actual output will be different from a simple copy-paste due to the gcc being smarter than us.