Q. Please write the execution of sample.c to a.out executable binary of c program in my own words.
->
Let's assume we have written the program in the text editor/ IDE (such as gedit, kwrite, VSCode, etc.). And saved it as sample.c

1. Preprocessor
During the compilation of a C program, the compilation is started off with preprocessing the directives (e.g., #include and #define).
The preprocessor (cpp - c preprocessor) is a separate program in reality, but it is invoked automatically by the compiler.
Preprocessor performs the following tasks as
• File Inclusion
• Macro expansion
• Conditional compilation
• Comment removal
• Extra Whitespace removal
For example, the #include <stdio.h> command in line 1 of sample.c tells the preprocessor to read the contents of the system header file stdio.h and insert it directly into the program text.
The result is another file typically with the .i suffix.
In practice, the preprocessed file is not saved to disk unless the -save-temps option is used.
This is the first stage of the compilation process where preprocessor directives (macros and header files are most common) are expanded.
The result is a file sample.i that contains the source code with all macros expanded.

2. Compiler
In this phase compilation of a program takes place.
Compiler converts human-understandable code to Machine dependent code ie in assembly Language. The compiler translates sample.i into sample.s or sample.asm depending upon the system architecture.
File sample.s contains assembly code.
We can explicitly tell gcc to translate sample.i to sample.s by executing the following Command.
gcc -S sample.s
The command line option -S tells the compiler to convert the preprocessed code to assembly language without creating an object file.
After having created sample.s we can see the content of this file. The instruction in the assembly code generally looks like this :

```
        .file   "Test2_Q1.c"
        .text
        .globl  newn
        .type   newn, @function
newn:
.LFB6:
        .cfi_startproc
        endbr64
```

```
        pushq  %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq   %rsp, %rbp
        .cfi_def_cfa_register 6
        subq   $32, %rsp
main:   movl   %edi, -20(%rbp)
        movl   $24, %edi
        call   malloc@PLT
        movq   %rax, -8(%rbp)
        movq   -8(%rbp), %rax
        movl   -20(%rbp), %edx
        movl   %edx, (%rax)
        movq   -8(%rbp), %rax
        movq   $0, 8(%rax)
```

3. Assembler

Here, the assembler translates sample.s into machine language instructions and generates an object file sample.o or sample.obj (depending upon the system architecture).
This Assembler step also contains several steps to generate this .o file. Such as Lexical analysis, Syntax analysis, Semantic analysis, Code optimisation, and Code generation. (learnt in CS_310 LLAP course).
And, the resulting file contains the machine instructions, machine code and some additional information about the program's symbols for our program.
The object file is not yet executable, as it needs to be linked with other object files and libraries to resolve external references and generate an address map.

4. Linker

This is the final stage in the compilation of our program.
This phase links object files to produce the final executable file.
Linker links our .o file with other .o files.
An executable file requires many external resources (system functions, C run-time libraries etc.).
In this step, functions like printf are contained in a separate pre-compiled object file printf.o, which must somehow be merged with our sample.o file.
The linker performs this task for us.
Eventually, the resulting file is produced, which is an executable .out or .exe file(depending upon the system architecture).
This is now ready to be loaded into memory and executed by the system.

5. Loader :

As the above executable file is on the hard disk but for execution purposes, it should be loaded in RAM.
The Loader is an Operating dependent entity which loads our executable file from the Hard disk into RAM. the program will start to execute after running the following command :
gcc sample.c

It will continue to run until it either completes its task or encounters an error.

To get the intermediate files which are created in the build process by using the following command :
gcc sample.c -save-temps

For better understanding :

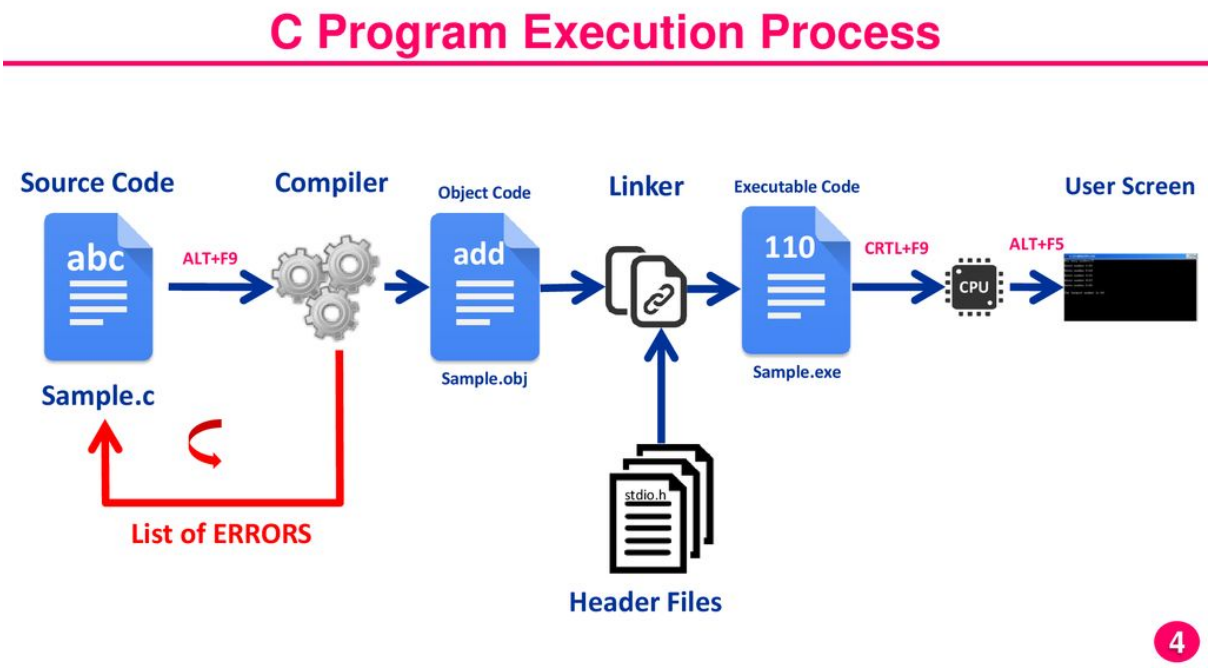

Image Source ~
https://www.google.com/url?sa=i&url=https%3A%2F%2Fslideplayer.com%2Fslide%2F16381723%2F&psig=AOvVaw33sJ5jKcshaTY6EbtNt2-S&ust=1680025170934000&source=images&cd=vfe&ved=0CBAQjRxqFwoTCKDf98_T_P0CFQAAAAAdAAAAABAp