**AIM:** Understand modules of the compilation process with the help of a program. (Pre-processor, Compiler, Assembler, Linker/Loader)

**Theory**: Explain Language Processing System with System Software Pre-processor, Compiler, Assembler and Linker/Loader.

# Pre-Processor

**Role:** The pre-processor handles initial code transformations before the actual compilation begins. It processes directives (e.g., #include, #define) and performs tasks like macro substitution and file inclusion.

**Example**: In C/C++, if you have #include <stdio.h>, the pre-processor includes the contents of stdio.h into your source file.

# Compiler

**Role:** The compiler translates the pre-processed source code into intermediate code or assembly code. It performs syntax and semantic analysis to ensure the code adheres to the language's rules and generates an intermediate representation (IR) of the program.

**Example**: The compiler translates C code into assembly code, such as converting int x = 5; into assembly instructions.

# Assembler

**Role:** The assembler converts the assembly code generated by the compiler into machine code (binary code). This machine code is a low-level code that the computer’s processor can execute directly.

**Example:** The assembler takes the assembly instructions and generates a corresponding object file containing machine code.

# Linker/Loader

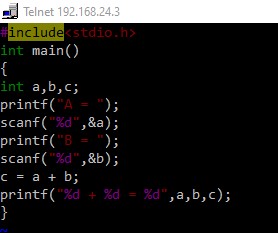
**Role:** The linker combines object files generated by the assembler into a single executable file, resolving external references between them. The loader then loads the executable into memory and prepares it for execution.

**Example**: If your program consists of multiple source files, the linker combines the object files from each source file and resolves references like function calls. The loader places the executable into memory and sets it up for execution.

* **Understand modules of the compilation process with the help of a program. (Pre-processor,**

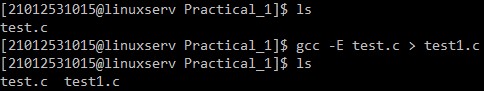
**Compiler, Assembler, Linker/Loader)**

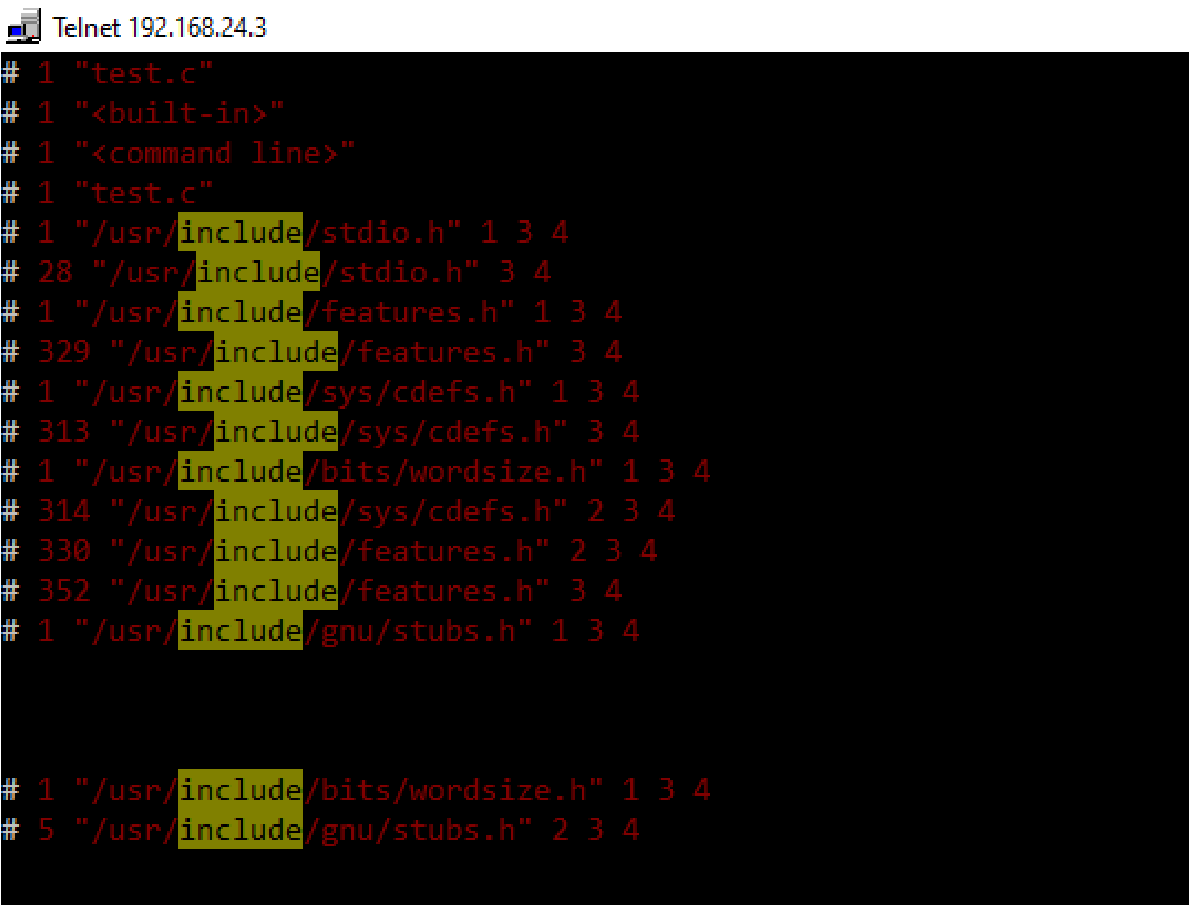
Execution steps to see the output of each module:



1. **Preprocessor**

gcc –E test.c > test1.c E option is used to perform preprocessor it performs file inclusion and macro expansion

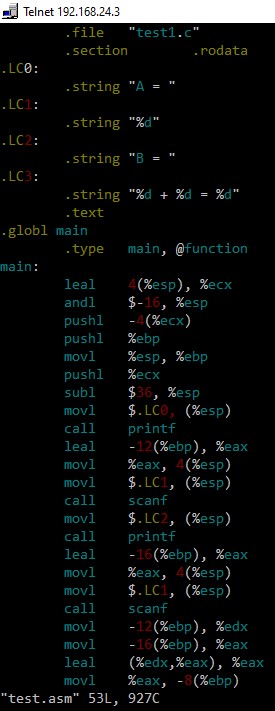
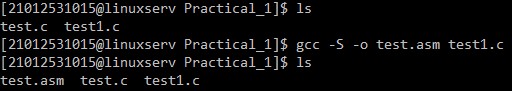




1. **Compiler**

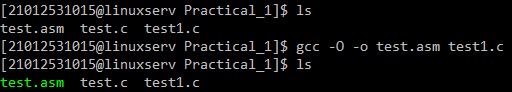
gcc –S test1.c S option generate assembly code file(test1.s) OR gcc –S –o test.asm test1.c – this generates assembly code file with the name given by the user.





1. **To optimize the assembly code**

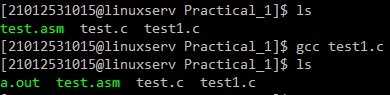
code gcc –S –O –o test.asm test1.c





1. **To compile the code**

gcc –C test1.s --- generate executable code



# Run the code

./a.out

