

Stage	What Happens	Cooking Analogy	Example (C++)
1. Preprocessing	Handles directives that start with # (like <code>#include <iostream></code>). These bring in external code libraries before compilation starts.	Gathering all the ingredients before cooking.	Expands all <code>#include</code> files.
2. Compilation (Translation to Assembly)	The compiler translates your human-readable code into an intermediate, lower-level assembly code. Errors in syntax are caught here.	The chef writes a precise recipe in a special cooking shorthand.	Produces <code>.s</code> or <code>.asm</code> files.
3. Assembly	The assembler converts the assembly code into object code (binary). This is not yet a full program.	The shorthand recipe is translated into coded cooking steps.	Produces <code>.obj</code> or <code>.o</code> files.
4. Linking	The linker combines your object file with any libraries (like I/O, math) to form a complete executable program.	Combining your dish with side ingredients before serving.	Produces <code>.exe</code> or <code>.out</code> .
5. Execution	The operating system loads the executable into memory and runs it as machine code.	The final meal is cooked and served — ready to eat!	The program runs and prints output.

The Compilation Process

Stage	What Happens	Cooking Analogy	Example (Python)
1. Source Code Loaded	You write a script in a high-level language like Python (.py file).	You open the recipe.	<pre>print("Hello, World!")</pre>
2. Lexical Analysis (Tokenisation)	The interpreter reads the text line by line, breaking it into meaningful tokens (keywords, variables, symbols).	The chef reads one sentence and picks out the key actions (“add”, “mix”, “heat”).	Recognises <code>print</code> , <code>(</code> , <code>"Hello, World!"</code> , <code>)</code>
3. Parsing	It analyses how the tokens fit together (grammar and syntax rules) to form valid statements.	Checks the recipe makes grammatical sense — “Add milk to sugar” not “Sugar add milk”.	Builds a syntax tree (a structural map of your code).
4. Interpretation / Execution	The interpreter immediately translates and executes each valid line into machine instructions.	The chef performs each instruction as they read it.	The CPU prints <code>Hello, World!</code> to the screen.
5. Error Handling	If a line causes an error, execution stops and an error message is displayed.	If the chef hits a confusing step, they stop cooking until it’s fixed.	<code>SyntaxError: unexpected EOF while parsing</code>

The Interpretation Process

Feature	Compiled Language (C++, Java, Go)	Interpreted Language (Python, JavaScript)
Translation	Entire program translated before running	Code read and executed line-by-line
Error Detection	Caught at compile-time	Found during execution
Speed	Usually faster	Slower (runtime translation)
Portability	Needs recompiling for each platform	Usually more portable
Example Output	.exe, .out, .class files	No separate file, runs directly in interpreter

Comparison of
Compilation
and
Interpretation