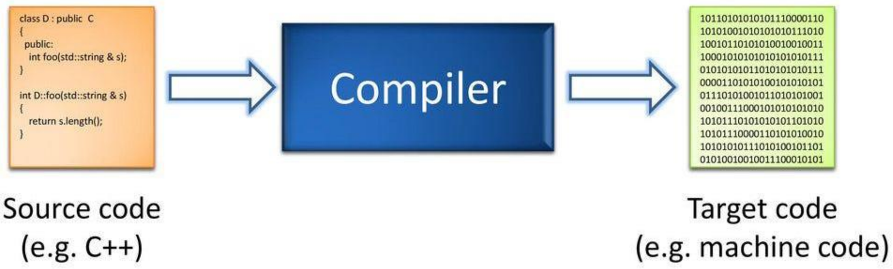
Section 1

Section contents:

- What is a Compiler?
- Difference between Compiler and Interpreter
- Compilation Process
- Phases of Compiler
- Types of Errors

What is a compiler?

A program that translates an executable program in one language into an executable program in another language.



```
class D : public  C
{
  public:
    int foo(std::string & s);
}

int D::foo(std::string & s)
{
    return s.length();
}
```

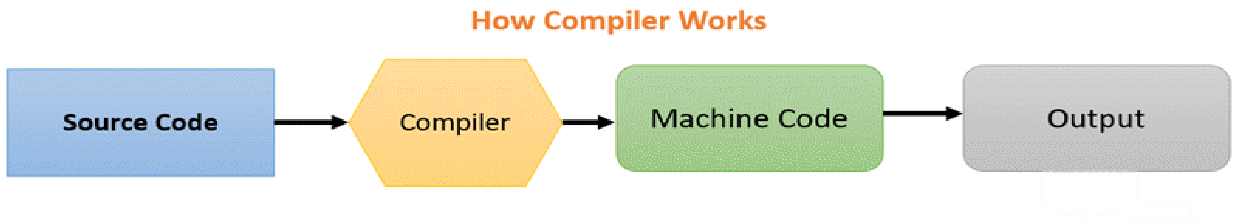
```
10110101010101110000110
10101001010101010111010
10010110101010010010011
10001010101010101010111
01010101011010101010111
00001101010100101010101
01110101001011010101001
00100111000101010101010
10101110101010101101010
10101110000110101010010
10101010111010100101101
01010010010011100010101
```

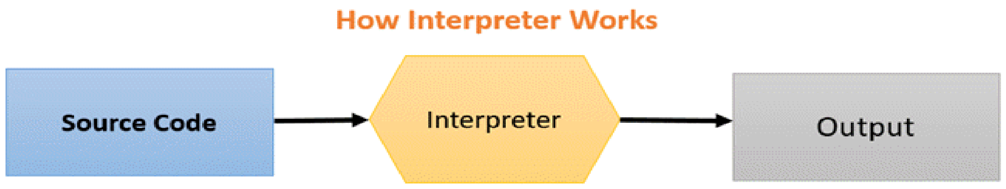
Features of compiler

- Correctness: preserve the meaning of the code
- Speed of target code (Translation)
- Speed of compilation
- Good error reporting/handling
- Cooperation with the debugger
- Support for separate compilation

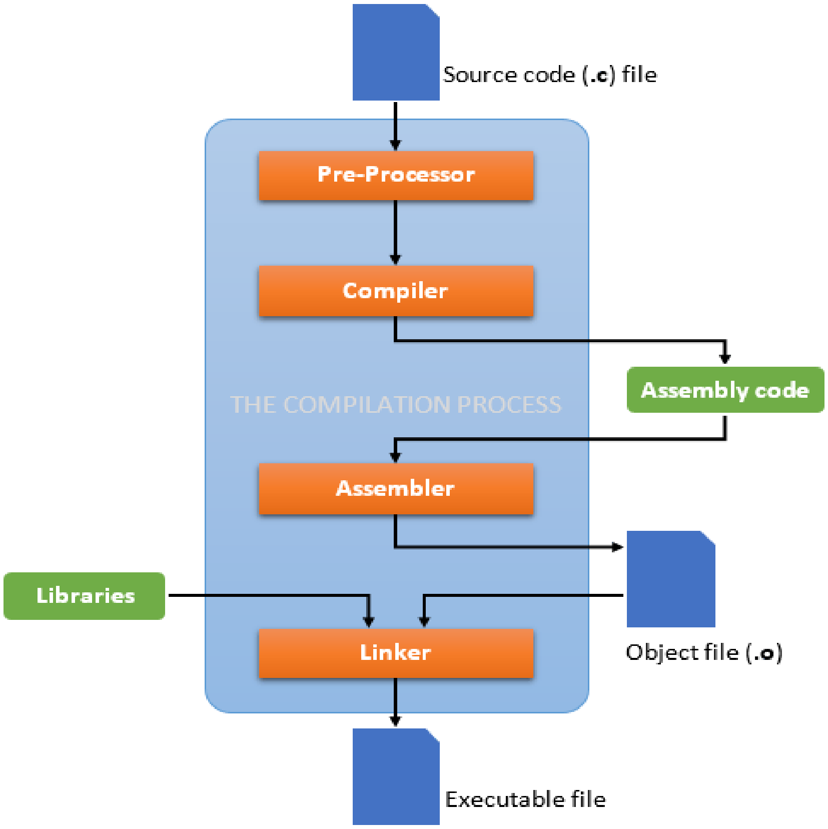
Difference between Compiler and Interpreter

Features	Compiler	Interpreter
Execution	Translate the entire code at once before execution	Translate and execute the entire code line by line
Output	Generate a separate executable file	No separate file, executed code directly
Error detection	Shows all errors at once after compilation	Stops at the first error it encounters
Speed	Faster (since it runs precompiled code if there are no changes)	Slower (because it translates every time)
Examples	C, C++	Python, JavaScript





Compilation Process



1) **Pre-processor**

   Prepares the source code before actual compilation by handling macros, including header files, and removing comments.

2) **Compiler**

   Translate the preprocessed code into assembly code, checking for syntax and semantic errors

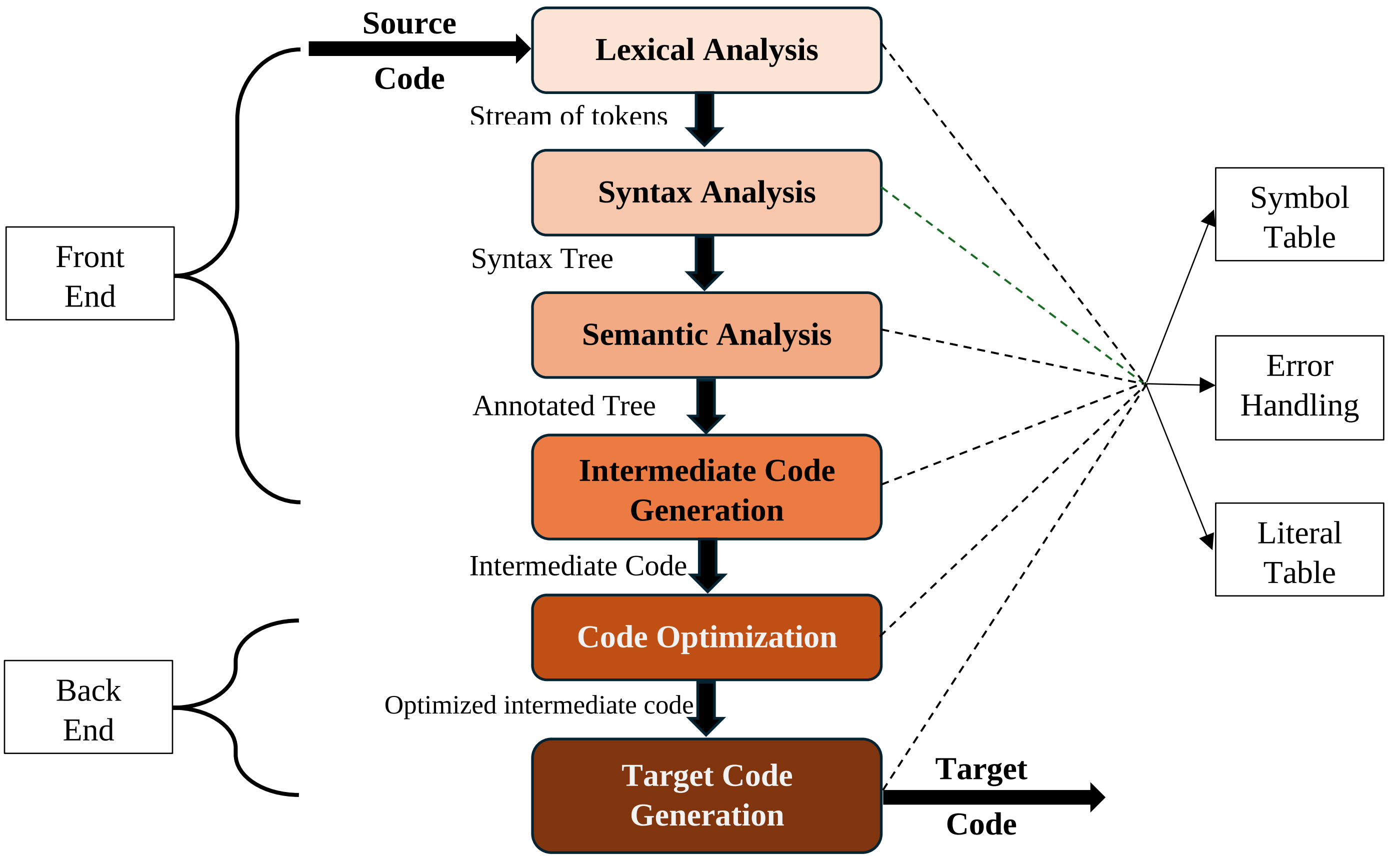
3) **Assembler**

   Converts assembly code into machine code, generating an object file (.o) that the computer can understand.

4) **Linker**

   Combines multiple object files and necessary libraries into a single executable file, resolving function and variable references.

Phases of Compiler



1) Lexical Analysis (Scanner)

Converts the source code into tokens (smallest units like keywords, identifiers, symbols).

Output: A stream of tokens.

Ex: **a[index]=4+2**

Lexemes	Token
a	Identifier/ ID
[	Left bracket
index	Identifier/ID
]	Right bracket
=	Assignment
4	Number/num
+	Operator
2	Number/num

2) **Syntax Analysis (Parser)**

Checks if the tokens follow the correct syntax (grammar rules of the language).

Output: A syntax tree, parse tree.

3) **Semantic Analysis**

Ensures that the code is meaningful and follows language rules.

Output: Annotated syntax tree

4) **Intermediate code generation**

Translates the syntax tree into an **intermediate representation (IR)** that is easier for optimization.

Output: Intermediate code (e.g., Bytecode).

5) **Code optimization**

Improves performance by reducing execution time and memory usage.

Output: Optimized intermediate code.

6) **Target code generation**

Converts the optimized intermediate code into machine code (binary instructions for the CPU).

Output: Executable machine code.

Auxiliary components interact with phases:

- **Literal table**

  A data structure that stores unique constants and string literals used in a program. It ensures efficient reuse, reducing redundancy and program size.

- **Symbol table**

  A structure used by compilers to store information about identifiers (variables, functions, labels, etc.), including their names, types, memory locations, and scopes.

- **Error handler**

  A component of a compiler or program that detects, reports, and sometimes corrects errors (syntax, semantic, etc.) to ensure smooth execution.

Types of Errors

Type	Description	Example
Syntax Error	Occurs when the code violates language rules.	Missing semicolon
Semantic Error	Code is syntactically correct but has incorrect meaning.	Assigning a string to an integer variable
Runtime Error	Happens during execution, causing the program to crash.	Division by zero or accessing an invalid memory location
Logical Error	The program runs but produces incorrect results.	Using + instead of * in a mathematical operation.