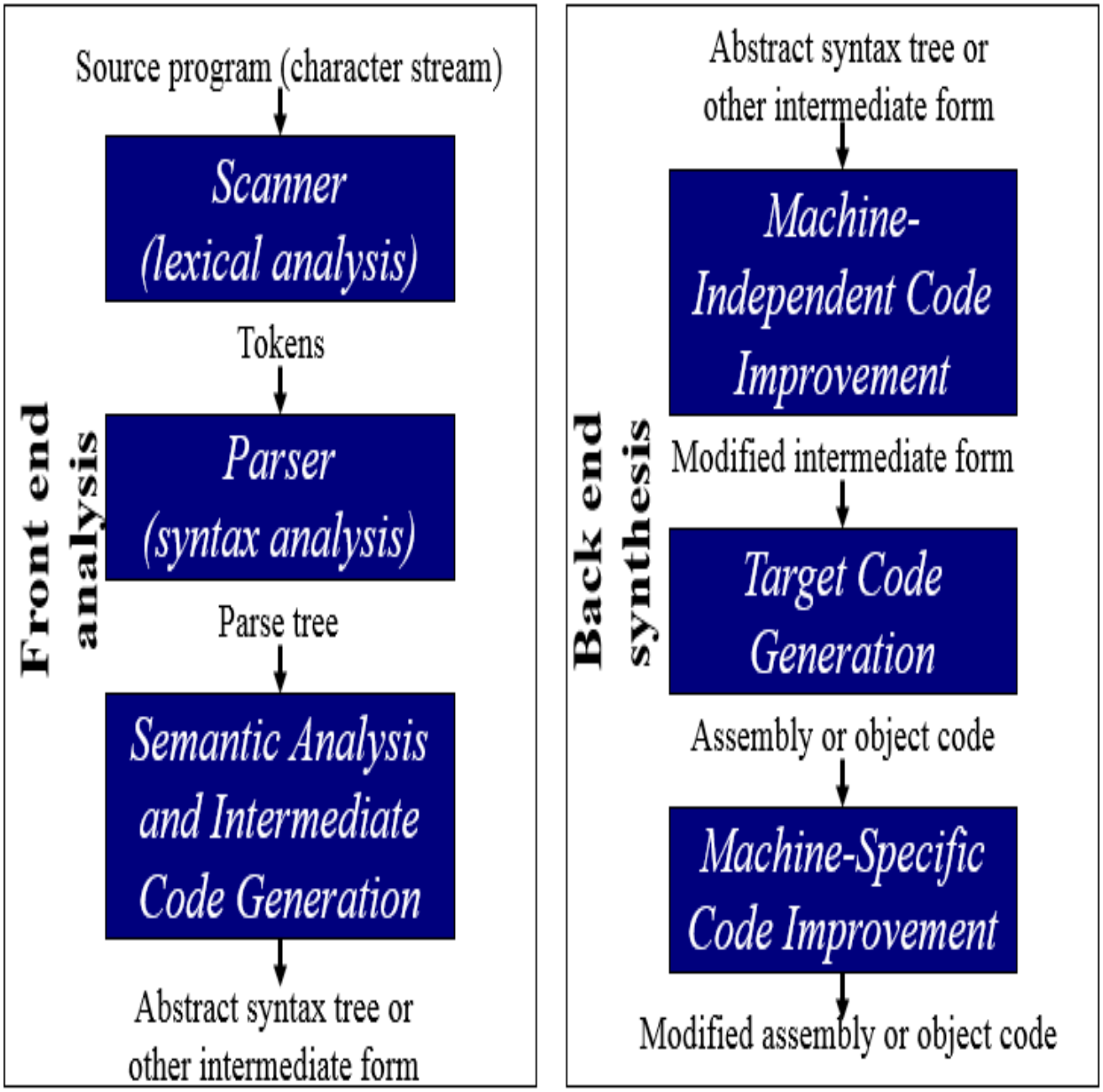
Lab Compiler Design

- A compiler is a software tool that converts high-level programming code into machine code that a computer can understand and execute.

- It acts as a bridge between human-readable code and machine-level instructions, enabling efficient program execution. The process of compilation is divided into six phases:

Compiler Front- and Back-end



Lexical Analysis

It is the first step of compiler design, it takes the input as a stream of characters and gives the output as tokens also known as tokenization.

The tokens can be classified into identifiers, Sperators, Keywords, Operators, Constants and Special Characters.

It has three phases:

Tokenization: It takes the stream of characters and converts it into tokens.

Error Messages: It gives errors related to lexical analysis such as exceeding length, unmatched string, etc.

Eliminate Comments: Eliminates all the spaces, blank spaces, new lines, and indentations.

- **Identifiers**
  - Names used for variables, functions, classes, etc.
  - Typically consist of letters, digits, and underscores, but cannot start with a digit.
  - Example: `myVariable`, `count123`, `_temp`
- **Keywords**
  - Reserved words that have a predefined meaning in the language.
  - Cannot be used as identifiers.
  - Examples (in C/C++/Java): `if`, `while`, `return`, `int`, `float`

- **Operators**
  - Symbols that represent operations like addition, subtraction, assignment, comparison, and logical operations.
  - Examples: `+`, `-`, `=`, `==`, `&&`, `||`

- **Constants**
  - Also known as literals.
  - Represent fixed values in the program, such as numeric constants (`123`), character

constants (`'a'`), or string constants (`"Hello"`).

- **Separators** (or Delimiters)
  - Punctuation symbols used to separate tokens.
  - Examples include commas (`,`), semicolons (`;`), parentheses (`( )`), braces (`{}`), and brackets (`[ ]`).

- **Special Characters**

  - Characters that may not fall neatly into other categories but serve a special purpose.

  - Could include symbols like `#` (in C/C++ for preprocessor directives), `@` in certain languages, or backticks in others.

int main() {
  int count = 10;
  count = count + 5;
  return count;
}
	• Identifiers: `main`, `count`
	• Keywords: `int`, `return`
	• Operators: `=`, `+`
	• Constants: `10`, `5`
	• Separators: `(`, `)`, `{`, `}`, `;`
	• Special Characters: (none in this snippet, but `#` would be if we had `#include <stdio.h>`)

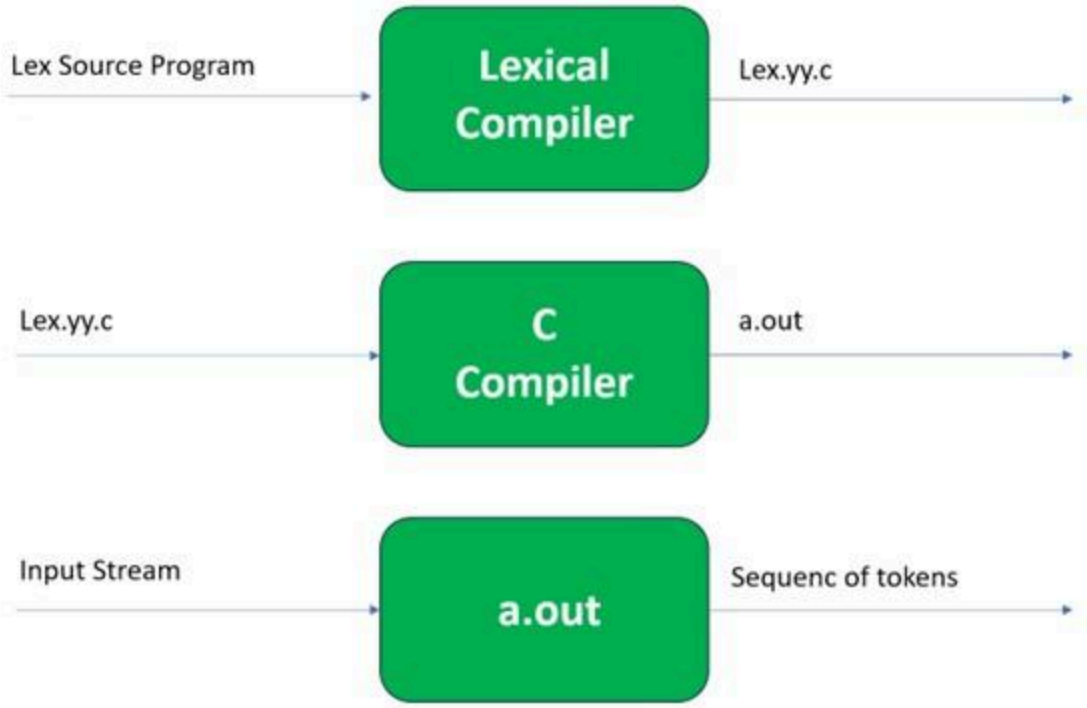
Lex Tool

- Lex is a tool or a computer program that generates Lexical Analyzers

(converts the stream of characters into tokens).

- The Lex tool itself is a compiler. The Lex compiler takes the input and transforms that input into input patterns.

- It is commonly used with YACC(Yet Another Compiler Compiler)



In the first step the source code which is in the Lex language having the file name 'File.l' gives as input to the Lex Compiler commonly known as Lex to get the output as lex.yy.c.

2. After that, the output lex.yy.c will be used as input to the C compiler which gives the output in the form of an 'a.out' file, and finally, the output file a.out will take the stream of character and generate tokens as output.