Chapter 1: Introduction

1 Compiler

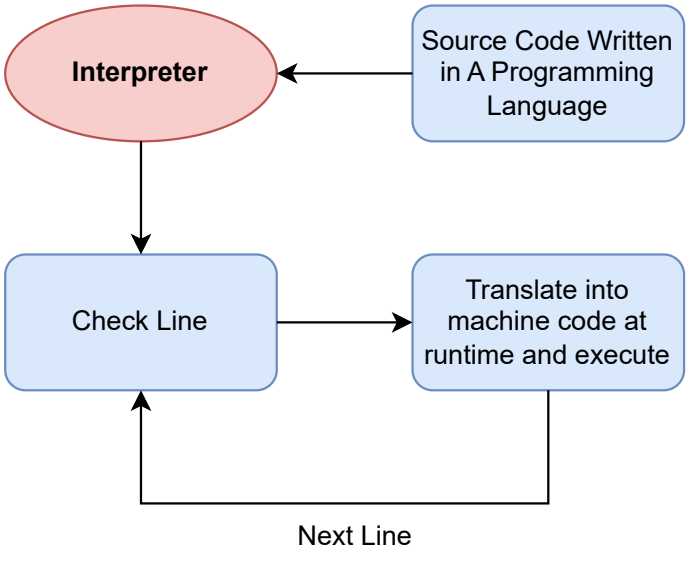
Definition

A compiler is a software program that takes a piece of code as input, analyzes it for errors, and generates output in the form of messages or logs. Additionally, it produces an executable file that can be run on a computer.

2 Interpreter

Definition

An interpreter follows the same steps as a compiler but differs in execution. Instead of analyzing the entire source code and generating an object file, it processes the code line by line then execute it immediately without producing an output file.



Note

Pros of an Interpreter:

- **Easier Debugging**: Errors are detected as the program runs, making it easier to pinpoint and fix issues.
- **Platform Independence at Source Level**: The same source code can be executed on different systems without recompilation, as long as an interpreter is available.
- **Smaller Program Size**: Interpreted programs do not require large binary files, reducing storage needs.

Cons of an Interpreter:

- **Slower Execution**: Since interpretation and execution happen simultaneously, it is slower compared to compiled programs.
- **Runtime Errors**: Errors only appear when execution reaches the faulty line, which can lead to unexpected crashes.

3 Steps of Compilation

Steps

The process of compilation is done following these steps chronologically:

1. Lexical Analysis
2. Syntactic Analysis (Parsing)
3. Semantic Analysis
    - Generation of Intermediate Code
    - Syntax-Directed Translation
4. Generation of Object Code

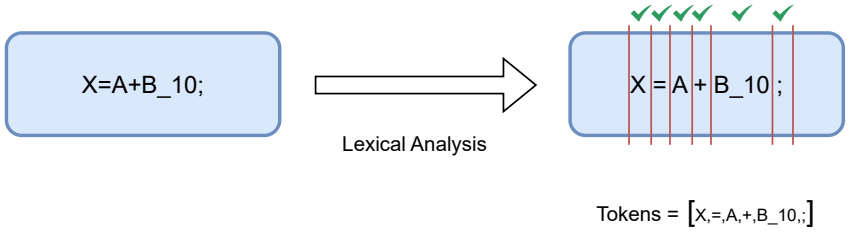
3.1 Lexical Analysis

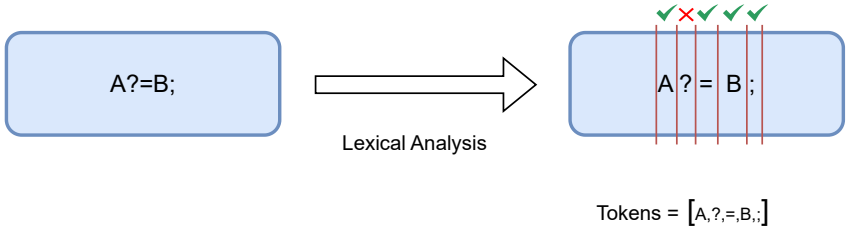
Lexical Analysis

Lexical analysis consists of dividing the entire code into lexical entities (tokens) storing them in a dictionnary and evaluating each of them independently throught automata or regex. Tokens can be divided based on:

- Spaces
- Logical operators (AND, OR, NOT, >=, >, <=, <, =)
- Arithmetic operators (+, -, *, /)
- Separators ([ ], { }, ( ))

Example





3.1.1 Types Of Token

Types

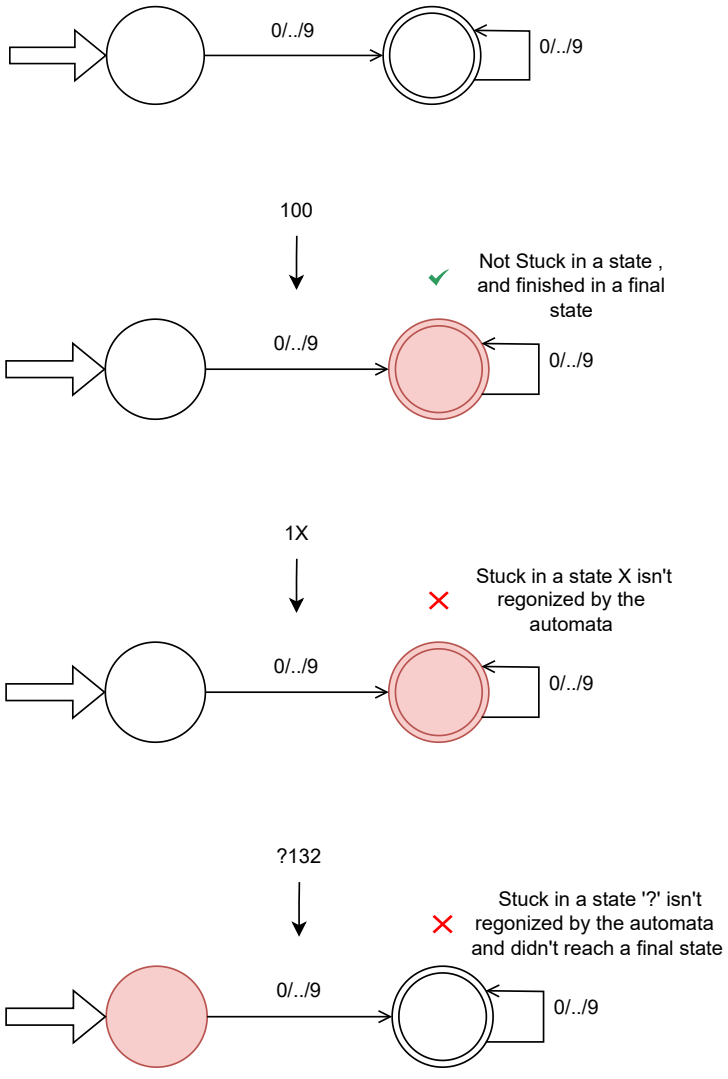
- **Constant**: Represent literal values like numbers : (3.14 , 1021 ,-120) , literal string : "hello world !" and character 'a'.
- **Keyword**: Reserved words that have a predefined meaning in the language, like 'if', 'while', and 'return'.
- **Identifier**: Names given to variables, functions, or objects.
- **Separator**: Symbols used to separate tokens (logical , arithmetic operators ...etc)

3.1.2 Automata

Automata

Lexical analysis uses a finit deterministc automata to verify wether a given token is correct or no , a token is considered correct if it reaches an end state with no blockage in the process

Example



Note

Never loop on the start state, as this will always cause undesired behavior, potentially leading to unexpected results or halting the automaton from transitioning to the next valid state.

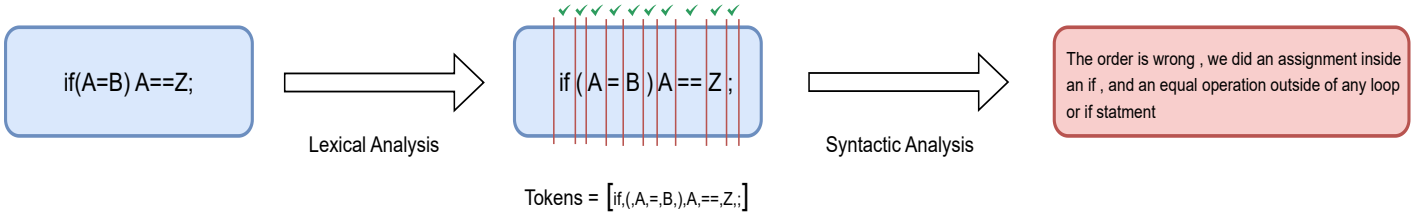
3.2 Syntactic Analysis (Parsing)

Syntactic Analysis

Syntactic analysis checks if the order of tokens is correct and adheres to the required format (using automata or regular expressions). There are two methods:

- Descending methods
- Ascending methods

Example



Note

If a code is lexically correct, it does not guarantee that it will be syntactically correct. However, the opposite is true:

$$\text{Correct Syntactically} \rightarrow \text{Correct Lexically}$$
$$\text{Correct Lexically} \nrightarrow \text{Correct Syntactically}$$

3.3 Semantic Analysis

Semantic Analysis

This phase ensures the code is logically correct and adheres to the language's rules. It verifies **type mismatches, variable declarations, function calls, and more**.

- **Generation of Intermediate Code**: Produces a simplified, platform-independent representation of the program. This step is crucial for optimizing the code and converting it into machine code later , it has many types :
  - **Quadruple Form**
  - **Postfix and Prefix Forms ... etc**
- **Syntax-Directed Translation**: Generates intermediate representations, ensuring that the semantics (meaning) of the program guide the creation of subsequent steps.

3.4 Object Code Generation

Object Code Generation

The final step of compilation, where the object code is generated to execute the program. This is the only step in the compilation process that we will not cover in this course.

4 Dictionary(Symboles Table)

Dictionary

The dictionary is an array that stores tokens and information about them. It is first initialized during lexical analysis and then progressively filled as the analysis continues.

Tokens	Token Type	Semantic Type	Memory address
X	Identificator	int	10021
Y	Identificator	float	9212

Note

Memory allocation occurs after semantic analysis to prevent allocation when the source code contains errors. This ensures that resources are only allocated once the code is verified as semantically correct.

5 Compilation Steps Diagram