Compilação X Interpretação

Dentre as tantas diferenças entre os dois, no contexto de análise lexical, os compiladores tem como principal objetivo traduzir de uma só vez todo o código para a “”linguagem de máquina” e executar o mesmo. Enquanto o interpretador compila linha por linha do código e executando ele aos poucos focado em encontrar erros pontualmente.

Uma vez que na compilação a análise léxica é feita geralmente em uma fase distinta do processo, lendo os tokens todos de uma vez e com isso gerando a tabela de símbolos. Na interpretação os tokens são lidos conforme são criados e juntamente atualiza a tabela símbolos, o que por sua vez é muito conveniente em certos casos mas afeta consideravelmente o desempenho.

O que é a Análise Léxica?

A análise léxica é uma das principais no processo de compilação de linguagens de programação e tem aplicações na linguística computacional. Ela consiste em transformar uma sequência de caracteres de entrada em uma sequência de unidades significativas chamadas tokens, que são então usados na análise sintática e semântica subsequente.

Bases da Análise Léxica

Para tornar todo o processo de criação e estudo de caso, o uso de autômatos e expressões regulares se mostram importantes de maneira que com eles somos capazes de padronizar os diferentes tokens que podem aparecer no código aumentando desempenho e eficiência. E também simplificar processos como a criação da tabela de símbolos através do reconhecimento dos caminhos possíveis e seus resultados.

Tabelas de Símbolos

Através das tabelas de símbolos, o analisador identifica e classifica tokens a partir do código-fonte de um programa. Esses tokens podem caracterizar palavras-chave, identificadores, operadores, delimitadores e literais.

O analisador léxico transforma em tokens um código por exemplo:

x = 42 + y / <id, 1> <=> <id, 2> <+> <id, 3>

Sua estrutura é basicamente composta de:

1. Nome do Símbolo: Identificador da variável.
2. Tipo: Tipo de dado (inteiro, float, char, etc.)
3. Escopo: O alcance do símbolo, indicando onde ele é válido.
4. Endereço: Localização na memória onde o valor associado ao símbolo é armazenado.
5. Valor: O valor inicial do símbolo.

6. Atributos: Qualquer outro atributo relevante, como modificadores (const, static, etc.), parâmetros no caso de funções.