**Compilador em Módulos**

**Scanner, Lexer (Analise Léxica)**: Analisa se aquelas palavras pertencem ao dicionário e retornam tokens;

T**okens:** Padrão reconhecido da linguagem;

**Parsing (Analisador sintático):** Recebe os tokens e verifica se a ordem das palavras está correta, (construção da árvore);

**Analisador semântico:** Verifica se não há erros de concordância no código, ex.: “int recebe string”;

* X, e y foram declarados?
* Existem outras variáveis com estes nomes?
* O tipo está correto?

Cada fase depende do resultado anterior;

Podem existir fases extras, existem diversas abordagens e elas podem ser combinadas;

**Código intermediário**

Gera instruções para a máquina abstrata e não é executada pela máquina alvo, ex.: “bytecode do java”;

**Otimização**

Analisa o código intermediário e tenta melhorar ele de forma que venha ser executado mais rápido, eliminando repetições e dados redundantes dentro do programa;

**Geração de código de maquina**

Gera o código objeto para a máquina(arquitetura) especifica;

Um código “construído” em uma maquina x, e “rodado” em uma maquina y, deve possuir ambos resultados idênticos. (hipoteticamente em um universo onde se é possível rodar a linguagem de alto nível em x e o código maquina em y). Resumindo, independente do compilador para quaisquer arquiteturas, se o código é de alto nível, o resultado sempre deve ser o mesmo;

**Interpretador**

Não gera código a partir da árvore sintática, ele realiza as fases no tempo de execução, assim, pulando algumas fases, resultando em um desempenho pior.

**Linker (Linkador)**

Pega trechos de um programa e vincula ao sistema operacional. Nesta etapa, o programa se “junta” ao sistema operacional, diferenciando o “produto” final no Windows e no Linux, por exemplo.

**Montador (Assembler)**

Efetua a montagem da linguagem de montagem para código de máquina.

**Implementação de um analisador léxico**

**Padrão:** Expressão regular, é a forma de reconhecer os lexemas em uma cadeia de caracteres, ou também identificadores;

**Lexema:** A sequência de caracteres reconhecidos pelo padrão.

**Token:** Um lexema com um valor atribuído, é representado da seguinte forma, no caso haja o código:

if a < 10 print “ok”

Teríamos os tokens (representados por <\_, \_>):

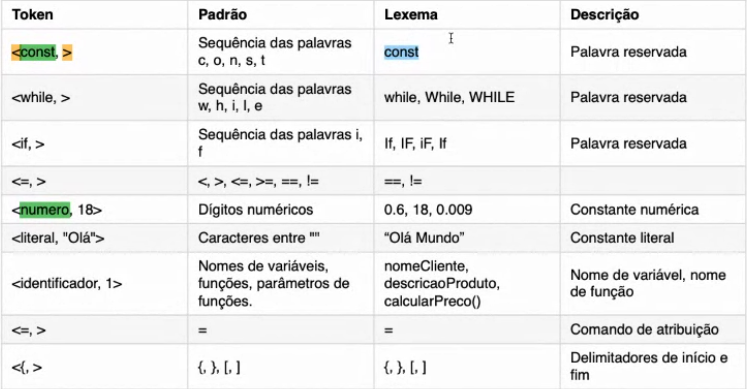
<if, >

<var, int>

< “>”, >

<10, >

**Termos**



Tokens simples, tokens que não possuem valor <if, >

Tokens com argumento, tokens que possuem valor <id, 10>

**Passos para reconhecer uma sequyencia de tokens:**

1. Reconhecer a substring relacionada através de um padrão;
2. Partir a string de entrada em lexemas, separando do restante do código;
3. Identificar o token de cada lexema.