# INE5421 – LINGUAGENS FORMAIS E COMPILADORES

- **PLANO DE ENSINO**
  - **Objetivo geral**
    - **Conhecer a teoria das linguagens formais visando sua aplicação na especificação de linguagens de programação e na construção de compiladores.**
  - **Objetivos específicos**
    - **Adquirir uma visão geral do processo de compilação sob o ponto de vista de implementação.**
    - **Correlacionar a Teoria das Linguagens Formais com a Teoria da Computação e esta com a Ciência da Computação.**
    - **Adquirir sólidas noções de linguagens formais e suas representações.**
    - **Ser capaz de especificar linguagens através de autômatos e gramáticas.**
    - **Conhecer e saber usar as técnicas formais de análise sintática.**
  - **Programa**
  - **Avaliação**
  - **Bibliografia**

# Capítulo I – Introdução

## I.1 - Introducão a Compiladores

### I.1.1 - Definições preliminares

#### Tradutor

- É um programa que traduz um programa fonte escrito em uma linguagem qualquer (denominada linguagem fonte) para um **programa objeto equivalente** escrito em outra linguagem (denominada linguagem objeto)


#### Compilador

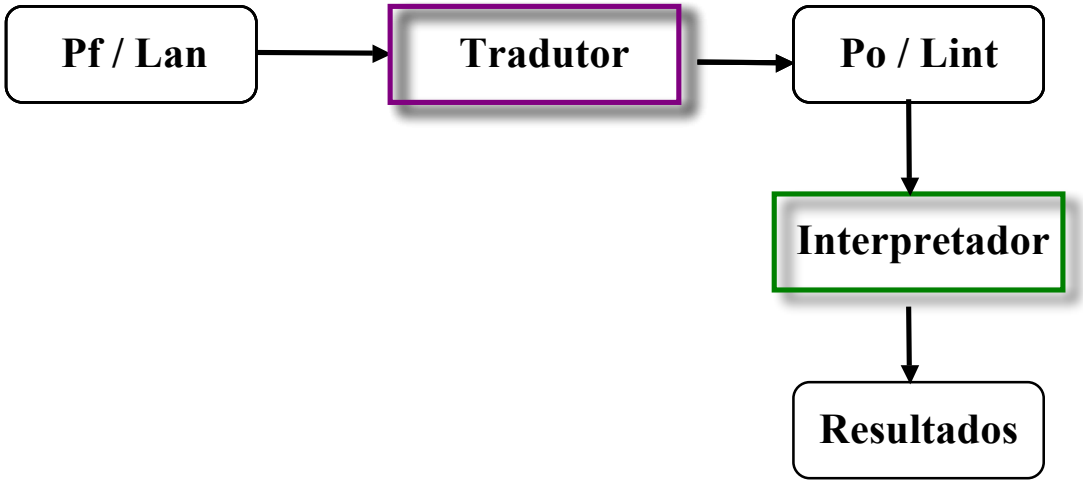
- É um Tradutor em que a linguagem fonte é uma linguagem de alto nível e a linguagem objeto é uma linguagem de baixo nível (assembly ou máquina)


#### Interpretador

- É um programa que interpreta diretamente as instruções do programa fonte, gerando o resultado.

## Tradutor / Interpretador

- Esquema híbrido para implementação de linguagens de programação




## Montador

- É um Tradutor em que o programa fonte está escrito em linguagem assembly e o programa objeto resultante está em linguagem de máquina

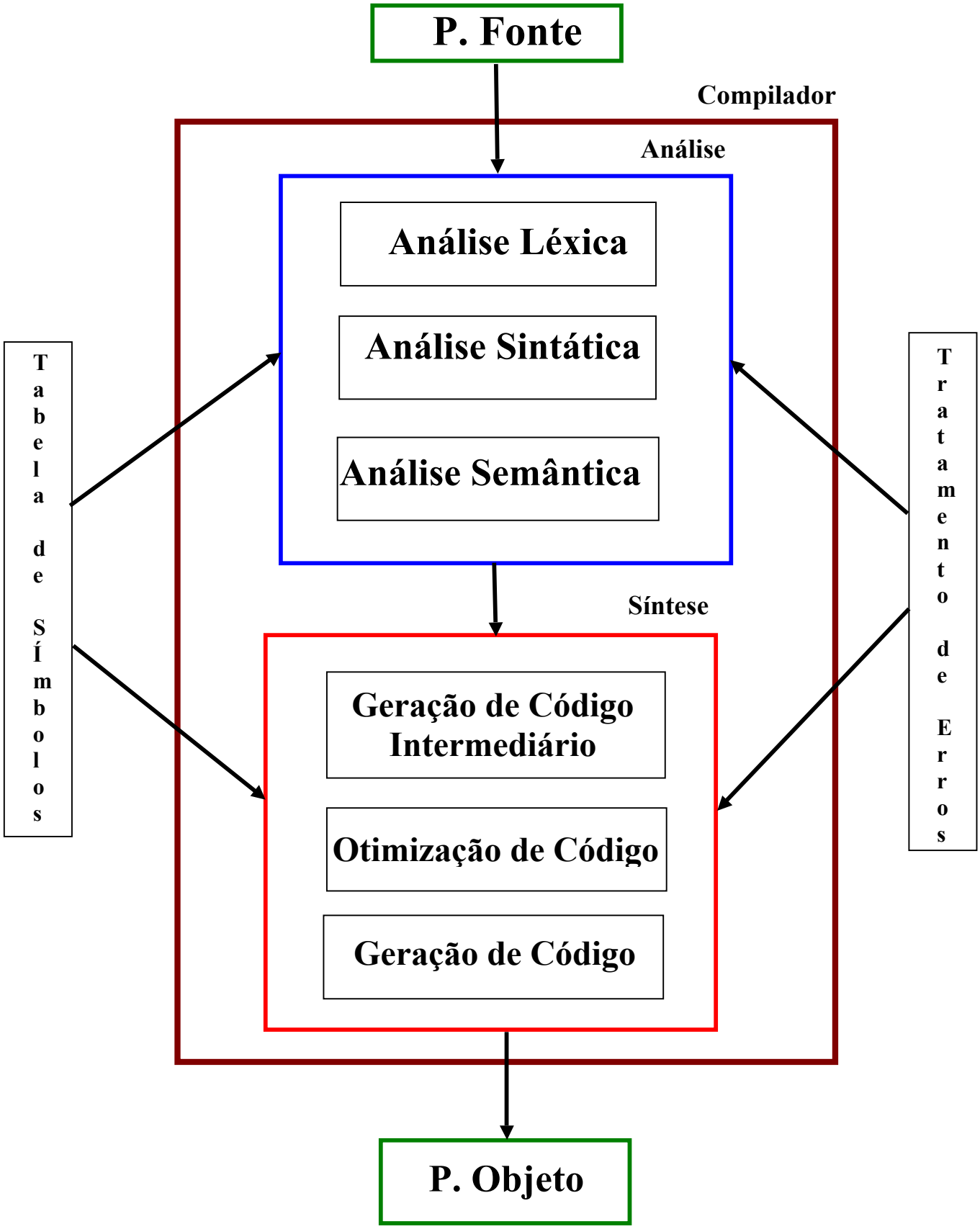

## Pré-processador

- É um Tradutor em que tanto o programa fonte quanto o programa objeto estão escritos em linguagens de alto nível


## Cross - Compiler

- Compilador que gera código para uma máquina diferente da utilizada na compilação.

# I.1.2 - Estrutura geral de um Compilador (Modelo Análise - Síntese)



# Modelo Front-end – Back-end

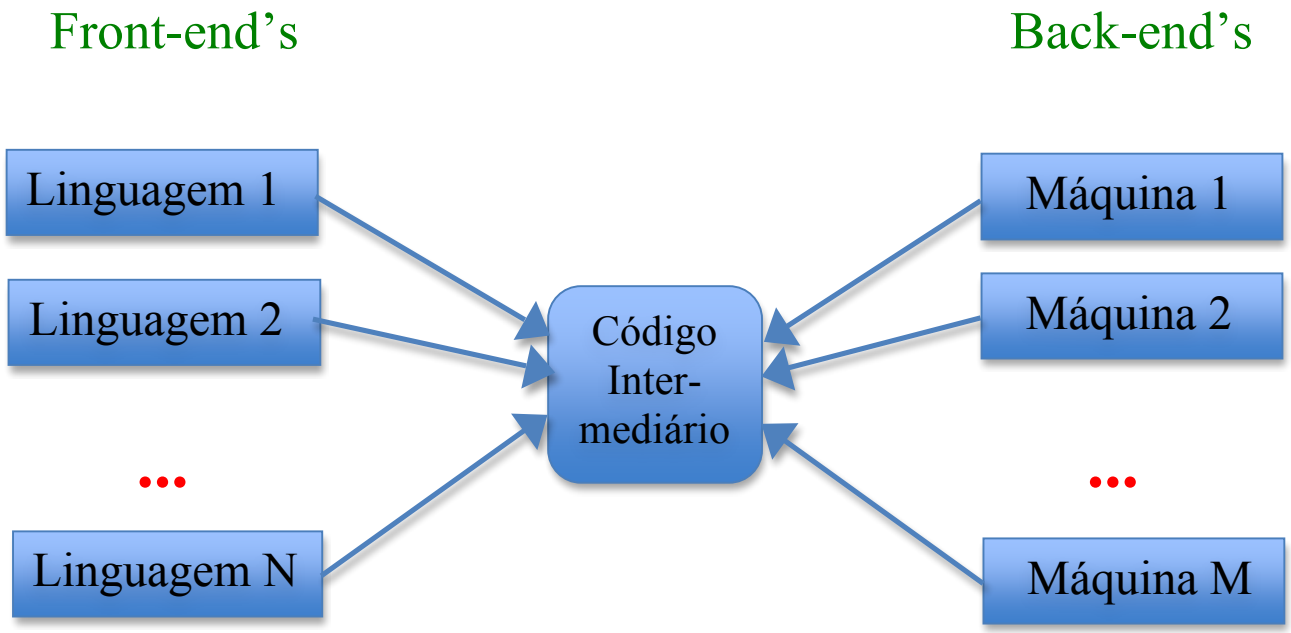
## Front-end

- Independente de máquina
- Compreende:
  - Análise (Léxica, Sintática e Semântica)
  - Geração de Código Intermediário

## Back-end

- Independente de linguagem
- Compreende:
  - Otimização de Código
  - Geração de Código

## Infraestrutura de desenvolvimento de Compiladores



# I.1.3 - Formas de Implementação de Compiladores

**Fase** - Procedimento que realiza uma função bem definida no processo de compilação.

**Passo** - Passagem completa do programa compilador sobre o programa fonte que está sendo compilado.

## Formas de Implementação

- **Compiladores de 1 passo**
  - **Todas as funcionalidades (fases) são executadas simultaneamente**
- **Compiladores de vários passos**
  - **Diferentes composições (agrupamento de fases)**
  - **Exemplos :**
- **Critérios para escolha**
  - Memória disponível
  - Tempo de Compilação
  - Tempo de execução
  - Características da Linguagem
    - **Referências futuras**
  - Características das Aplicações
    - **Necessidades de Otimização**
  - Tamanho / Experiência da Equipe
  - Disponibilidade de Ferramentas de Apoio
  - Prazo para desenvolvimento
- **Vantagens X Desvantagens**

# I.1.4 - Fases de um Compilador

## I.1.4.1 - Analisador Léxico

- Interface entre o programa fonte e o compilador
- Funções básicas:
  - Ler o programa fonte
  - Agrupar caracteres em itens léxicos (tokens)
    - * Identificadores
    - * Palavras Reservadas
    - * Constantes (numéricas e literais)
    - * Símbolos especiais (simples, duplos, ...)
  - Ignorar elementos sem valor sintático
    - * Esp. em branco, comentários e caract. de controle
  - Detectar e diagnosticar erros léxicos
    - * Símbolos inválidos, elementos mal formados
    - * Tam. inválido de constantes, literais e identif.

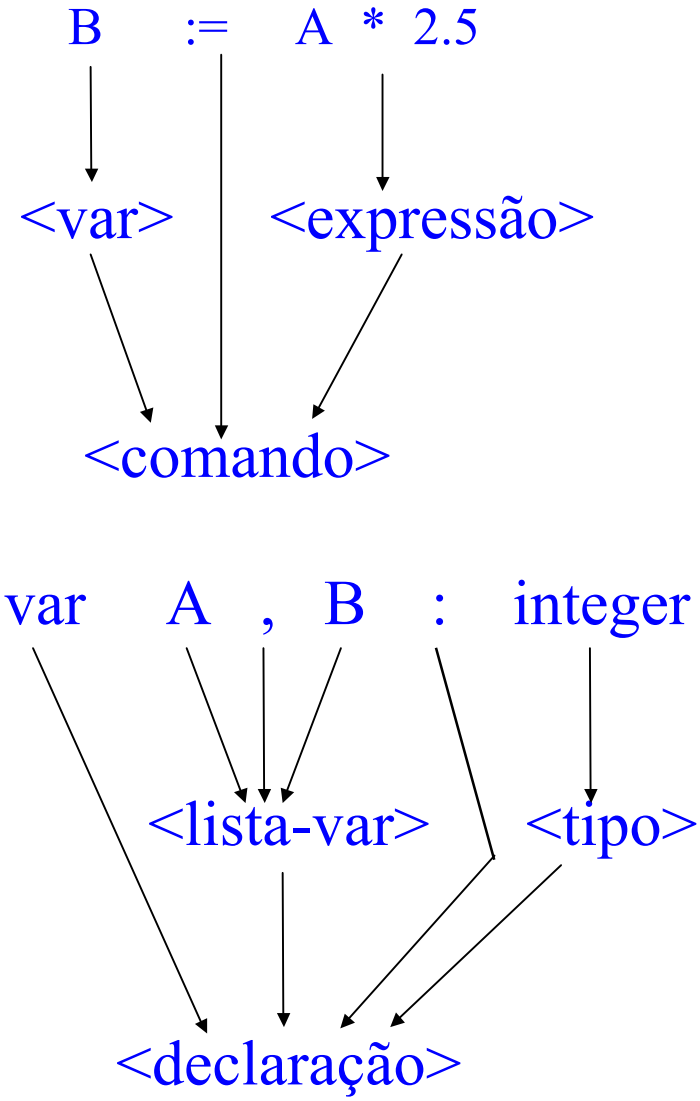
- Exemplo:

| Programa Fonte | Tokens Reconhecidos | |
|---|---|---|
| program exemplo;<br>var A, B : integer;<br>begin<br>(* Inicio do programa *)<br>read ( A );<br>B := A + 2.5;<br>. . .<br>end. | program<br>exemplo<br>;<br>var<br>A<br>,<br>...<br>end<br>. | 1 - PR<br>2 - ID<br>3 - SE<br>4 - PR<br>2 - ID<br>5 - SE<br>...<br>37 - PR<br>38 - SE |

# I.1.4.2 - Analisador Sintático

- Funções básicas
  - Agrupar TOKENS em estruturas sintáticas (expressões, comandos, declarações, etc. ...)
  - Verificar se a sintaxe da linguagem na qual o programa foi escrito está sendo respeitada
  - Detectar/Diagnosticar erros sintáticos

- Exemplos:



# I.1.4.3 - Analisador Semântico

SEMÂNTICA ≅ COERÊNCIA ≅ SIGNIFICADO ≅ SENTIDO LÓGICO

- Funções básicas:
  - Verificar se as construções utilizadas no P.F. estão semanticamente corretas
  - Detectar e diagnosticar erros semânticos
  - Extrair informações do programa fonte que permitam a geração de código

- Verificações Semânticas Usuais
  - Análise de escopo
  - Declaração de variáveis
    - Obrigatória?
    - Múltiplas declarações permitidas?
  - Compatibilidade de tipos
  - Coerência entre declaração e uso de identificadores
  - Correlação entre parâmetros formais e atuais
  - Referências não resolvidas
    - Procedimentos e desvios

# Tabela de Símbolos :

**Definição** - Estrutura onde são guardadas as informações (os atributos) essenciais sobre cada identificador utilizado no programa fonte.

## Atributos mais comuns

- **nome**
- **endereço relativo (nível e deslocamento)**
- **categoria**
  - **variável**
    - simples - tipo
    - array - dimensões, tipo dos elementos
    - record - campos (quant. e apontadores)
    - ...
  - **constante**
    - tipo e valor
  - **procedimentos**
    - procedure ou função
    - número de parâmetros
    - ponteiro para parâmetros
    - se função, tipo do resultado
  - **parâmetro**
    - tipo
    - forma de passagem (valor , referência)
  - **campo de record**
    - tipo, deslocamento dentro do Record

## Tratamento / Recuperação de ERROS:

- **Funções**
  - Diagnosticar erros léxicos, sintáticos e semânticos encontrados na etapa de análise
  - Tratar os erros encontrados, permitindo que o compilador **recupere-se** de uma situação de erro de forma que a análise possa ser concluída

---

## I.1.4.4 - Gerador de Código Intermediário

- **Função**
  - Consiste na geração de um conjunto de instruções (equivalentes ao programa fonte de entrada) para uma máquina hipotética (virtual)
- **Exemplo**

**E := ( A + B ) * ( C + D )**

| Quadrupla | Máquina de acumulador |
|---|---|
| ( + , A , B , T1)<br>( + , C , D , T2)<br>( * , T1 , T2 , E) | carregue A<br>some B<br>armazene T1<br>carregue C<br>some D<br>armazene T2<br>carregue T1<br>multiplique T2<br>armazene E |

## I.1.4.5 - Otimizador de código

- **Função**
  - Melhorar o código, de forma que a execução seja mais eficiente quanto ao tempo e/ou espaço ocupado

- **Otimizações mais comuns**
  - Agrupamento de sub-expressões comuns
    **ex. c := (a + b ) * ( a + b )**
  - Eliminação de desvios para a próxima instrução
  - Retirada de comandos invariantes ao LOOP
  - Eliminação de código inalcançável
  - Redução em força
  - Transformação/avaliação parcial
  - Alocação ótima de registradores

## I.1.4.6 - Gerador de Código

- **Função :**
  - Converter o programa fonte (diretamente ou a partir de sua representação na forma de código intermediário) para uma sequência de instruções (assembler ou máquina) de uma máquina real.