# 1. Introdução

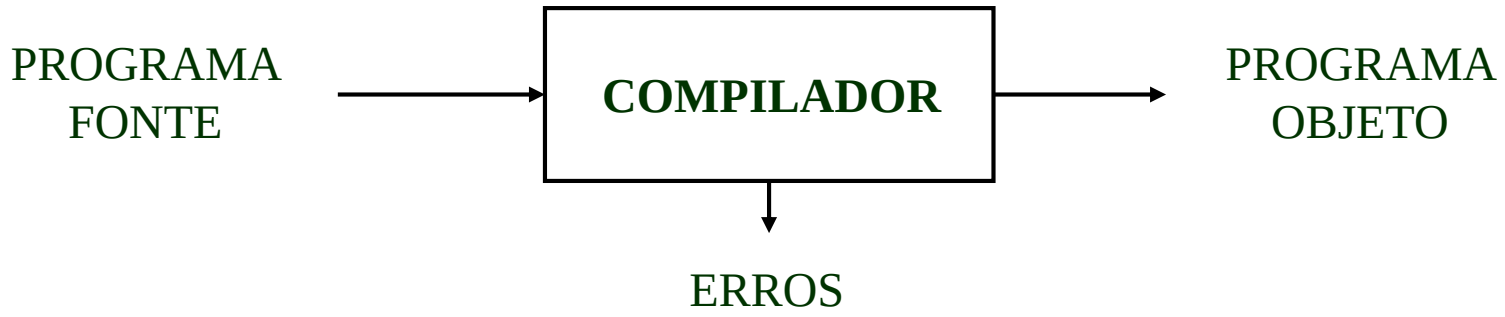


As fases e estrutura de um compilador

# Funções do compilador

- **Função principal do compilador:**

Traduzir um programa escrito em uma linguagem de alto nível (programa fonte) em um programa equivalente em linguagem de baixo nível (linguagem alvo ou objeto)

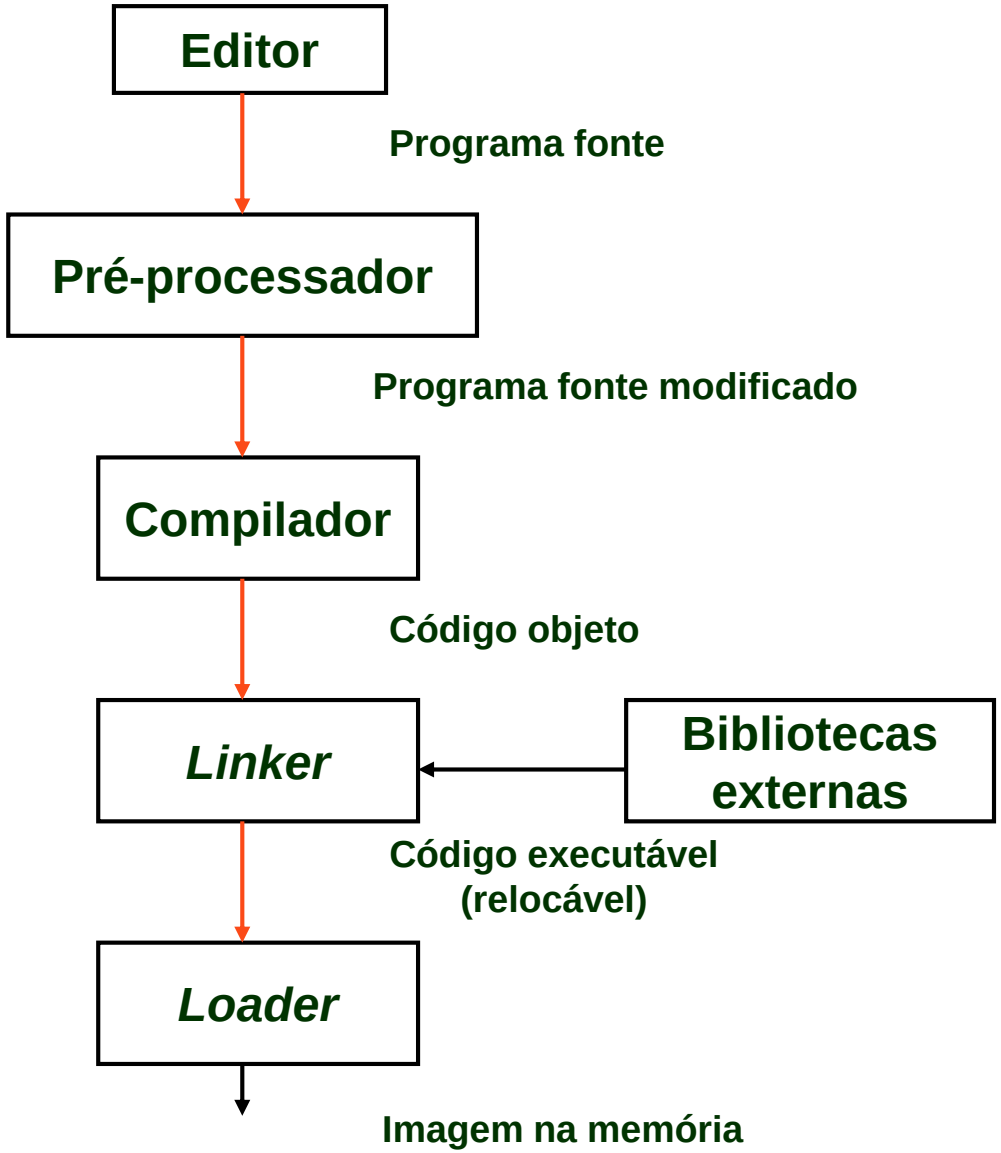


# Características:

- **Programa fonte:** geralmente uma linguagem de alto nível.
  **Programa alvo:** geralmente código máquina.
- Reconhecer erros (reportar).
- gerar código de máquina alvo correto (equivalente).
- gerir a **alocação de memória** dos **dados e código**.
- gerar o código de máquina num formato reconhecido pelos outros componentes do sistema (**linkers/ligadores** e **loaders/carregadores**)

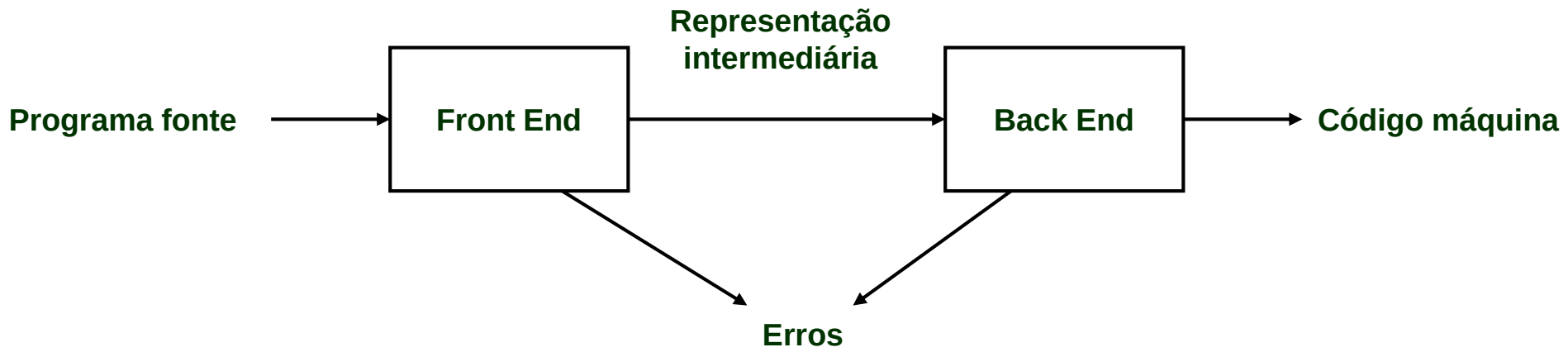
# Processo de desenvolvimento de programas





# Estrutura básica do compilador

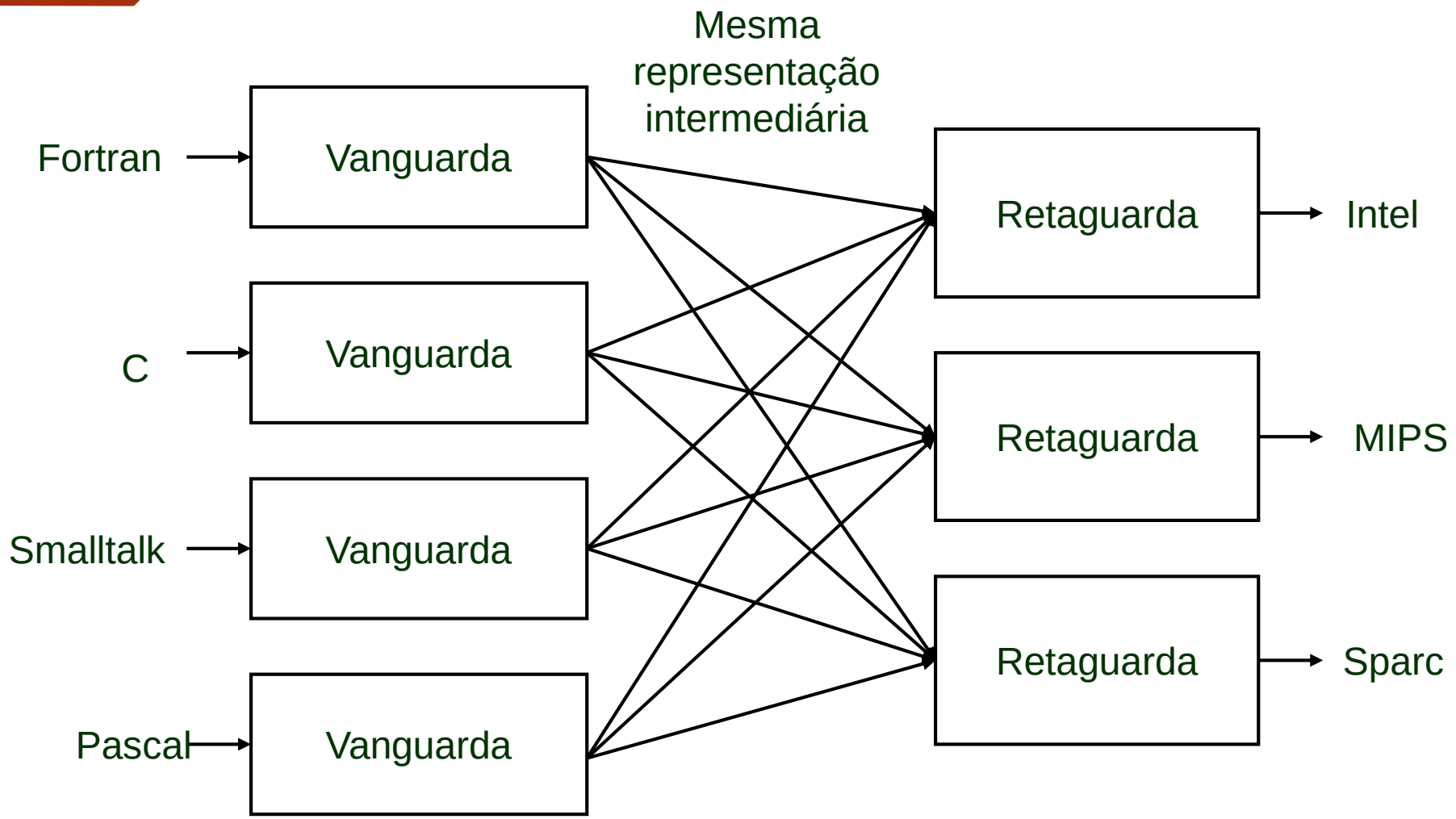
- Divisão do compilador em vanguarda (*front end*) e retaguarda (*back end*)




- Características:
  - Existência de uma representação intermediária do programa fonte (máquina abstrata).
  - A vanguarda mapeia o programa fonte numa representação intermediária.
  - A retaguarda produz o código máquina (máquina concreta) a partir da representação intermediária.
  - simplifica a produção de compiladores para várias máquinas concretas.
  - simplifica a produção de compiladores para várias linguagens fonte.
  - duas passagens ⇒ código mais eficiente que numa única passagem.

# Vantagens da representação intermediária

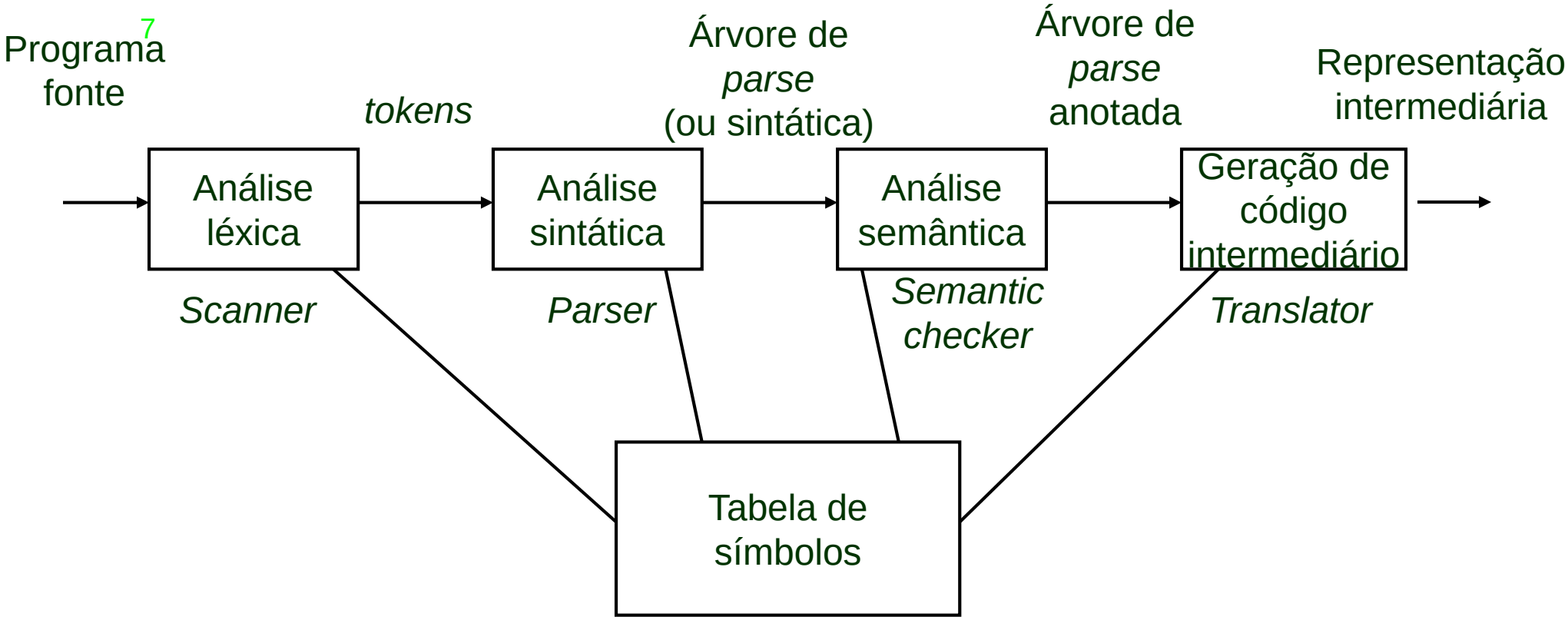





Só funciona se a representação intermediária for de baixo nível

# A Vanguarda (Front-End)

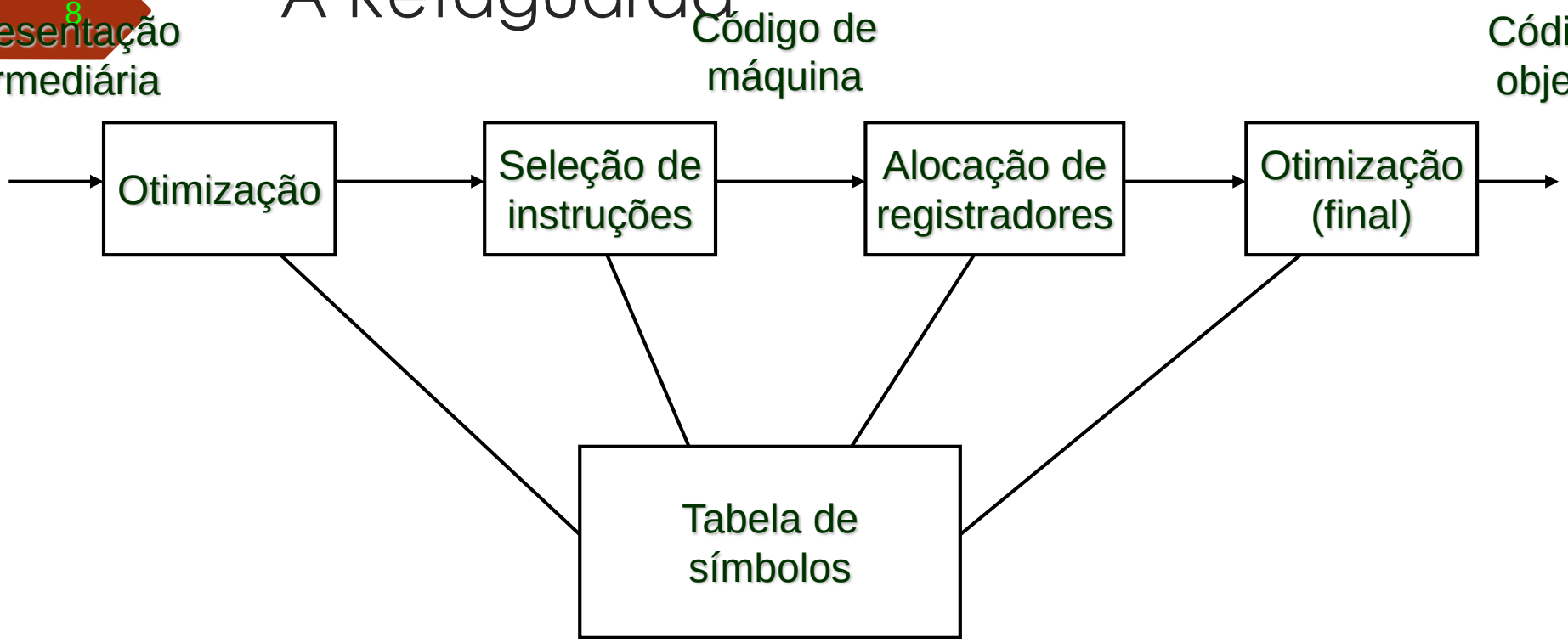





Responsabilidades:

- Reconhecer programas válidos;
- Produzir mensagens de erros;
- Produzir a representação intermediária;

# A Retaguarda




Responsabilidades:

- Traduzir a RI em código máquina (alvo);
- Escolher as instruções correspondentes a cada operação definida na RI;
- Decidir que informação manter nos registros do processador;

# Análise léxica

**if x == y then z = 1 ;** **torna-se na sequência dos seguintes *tokens*:**

- Agrupa seqüências de caracteres em *tokens* - as unidades básicas da sintaxe
- Exemplo:

**if → palavra-chave da linguagem**
**id → identificador (x)**
**== → operador**
**id → identificador (y)**
**then → palavra-chave da linguagem**
**id → identificador (z)**
**= → operador**
**num → constante 1)**
**; → separador**

- A sequência de caracteres que formam um *token* chama-se *lexema*
- *tokens* típicos: constantes (literais), id's, operadores, palavras--chave, etc.
- Elimina o chamado espaço em branco (espaços, tabs, LF, CR, comentários)
- Gerencia a tabela de símbolos

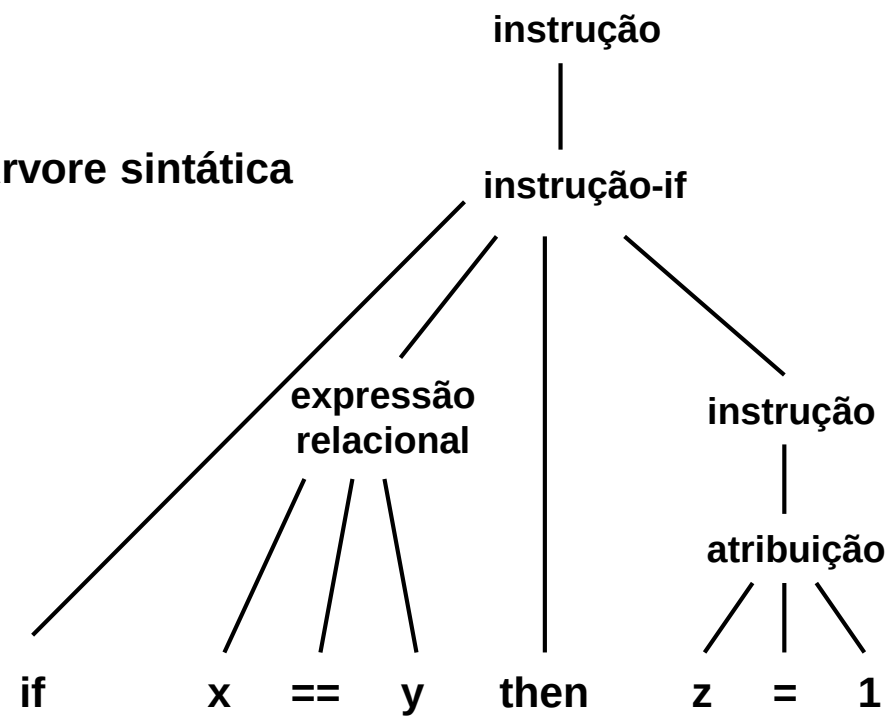
# Análise sintática

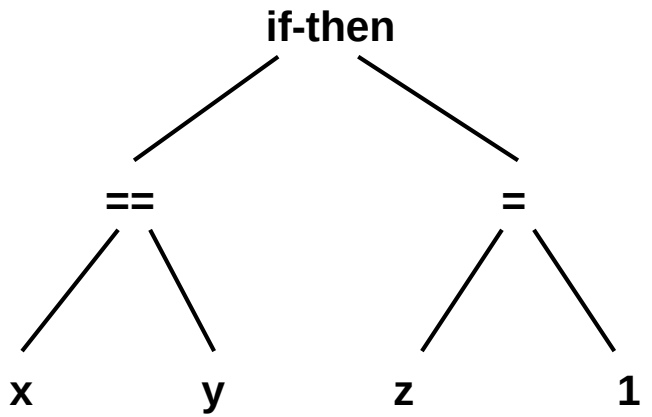
- **Reconhece a estrutura das construções da linguagem**
- **Essa estrutura é definida por uma gramática**
- **Só são válidas algumas seqüências de *tokens***
- **Constrói uma árvore (de *parse* ou de sintaxe) representando a estrutura da construção ou das regras gramaticais**

# Análise sintática

**Exemplo:**

**Árvore sintática**




**Árvore de sintaxe abstrata**
**(AST - *abstract syntax tree*)**

# Análise semântica

- Funções básicas:
  - Verificar se as construções utilizadas no P.F. estão semanticamente corretas
  - Detetar e diagnosticar erros semânticos
  - Extrair informações do programa fonte que permitam a geração de código
- Verificações Semânticas Usuais
  - Análise de escopo
  - Variáveis não declaradas
  - Múltiplas declarações de uma mesma variável
  - Compatibilidade de tipos
  - Coerência entre declaração e uso de identificadores
  - Referências não resolvidas
    - Procedimentos e desvios

# Análise semântica

- A análise semântica é implementada num compilador pelo cálculo de uma série de atributos (valores que caracterizam) associados às classes sintáticas definidas na gramática da linguagem

- Exemplos de atributos:
  - Tipos dos identificadores (variáveis, procedimentos, classes, …)
  - Tipos e valores das constantes (literais)
  - endereços de armazenamento na memória (relativos)
  - Etc

# Geração da representação intermediária

- O programa fonte depois de verificado semanticamente é transformado numa representação intermediária:
  - deve ser fácil de produzir
  - representar bem todas as características da linguagem fonte
  - representar bem as operações disponíveis na máquina alvo
  - deve ser fácil de traduzir para instruções máquina (alvo)
- Algumas formas de representação intermediária:
  - Gráficas: Árvores de sintaxe abstrata com notações (os atributos da análise semântica)
  - Lineares: Sequência de instruções para uma máquina genérica e abstrata
    - Existem várias formas:
      - máquinas de 0 ou 1 endereços (máquinas de pilha)
      - máquinas de 2 endereços (próximas de alguns processadores reais)
      - máquinas de 3 endereços (de mais alto nível)

# Tabela de símbolos

- Estrutura de dados onde é armazenada toda a informação relativa aos identificadores utilizados no programa fonte
- É acessada por praticamente por todos os módulos do compilador
  - Os módulos de análise criam as entradas na tabela, e à medida que vão recolhendo informação sobre os identificadores estas vão sendo armazenadas na T. S.
  - Os módulos de síntese usam essa informação para gerar código intermediário ou o código final
- A tabela de símbolos deve suportar eficientemente operações de criação, consulta e remoção de entradas, em diversos níveis (*scopes*)
- Informação tipicamente armazenada na tabela de símbolos:
  - nome do identificador (lexema) - geralmente funciona como chave
  - tipo do identificador e outros atributos (local, global, modificável, etc)
  - endereço (relativo) onde se situa a entidade representada pelo identificador (variável, procedimento, função, etc.)

# Otimização de código intermediário

- Fase opcional, que pode exigir várias passagens sobre a representação intermediária
- Tentativa de melhoramento da representação intermediária por forma a facilitar a produção de melhor código máquina:
  - mais rápido
  - e/ou mais compacto (que utiliza menos memória)
- Algumas otimizações:
  - Reconhecer cálculos redundantes e eliminá-los
  - Remover código que é redundante ou inalcançável
- As otimizações não podem alterar resultados parciais ou finais do programa fonte

# Seleção de instruções - tradução

- Produzir uma sequência de instruções de máquina equivalente à representação intermédia
- A sequência deverá ser o mais possível rápida e compacta
- Deverá utilizar com eficiência os modos de endereçamento disponíveis na máquina alvo
- Envolve a atribuição final de endereços de memória a variáveis e a posições-destino no código
- Muito dependente da máquina alvo

# Alocação de registradores

- Os registos do processador são de acesso muito rápido, mas são limitados
- Pretende-se maximizar a permanência de valores nos registradores sempre que há necessidade de utilizar esses valores nas instruções
- Implica alterar (remover ou acrescentar) algumas instruções de *load* e *store* ou a utilização de outras instruções menos comuns
- A alocação ótima é difícil ⇒ problema NP completo para k registradores genéricos disponíveis

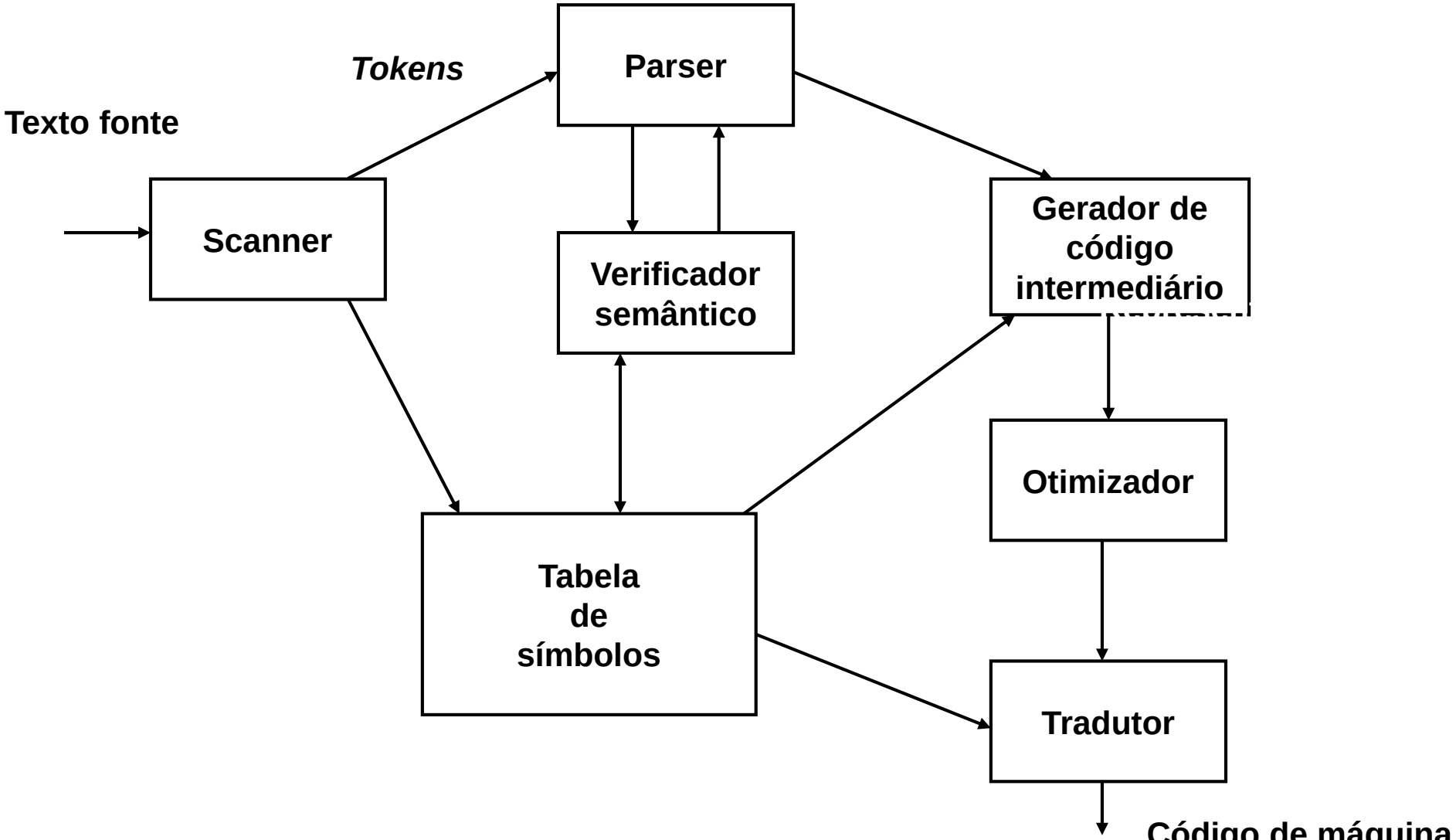
# Otimização final

- Otimização efetuada localmente no código de máquina através da análise de seqüências curtas de instruções
- Certas seqüências podem ser substituídas por outras equivalentes, mas mais eficientes:
  - Incremento em vez de soma
  - deslocamento (shift) em vez de multiplicação
- Eliminação de instruções redundantes, inúteis ou inalcançáveis

# Deteção e recuperação de erros

- É possível encontrar erros em praticamente todas as fases da compilação
- A maior parte dos erros é detectada nas fases de análise (léxica, sintática e semântica)
- Quando um erro é detectado deve ser emitida uma mensagem esclarecedora e dever-se-á prosseguir na compilação
- Sempre que um erro é detectado dever-se-á recuperar esse erro (descartando parte do texto fonte ou assumindo uma correção do erro) de forma que a tarefa de análise em curso possa prosseguir.

# A estrutura do compilador



# Exemplo

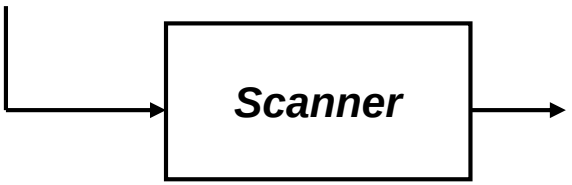


**Compilação de uma instrução de atribuição de uma dada linguagem de acordo com as regras gramaticais:**

**...**
**<atribuição> → id := <expressão>**
**<expressão> → <expressão> * <expressão>**
**| <expressão> + <expressão>**
**| id**
**| num**
**...**

**Atribuição a compilar:**

**...**
**posicao := inicio + veloc * 60**
**...**




**id, 1 := id, 2 + id, 3 * num,int, 60**

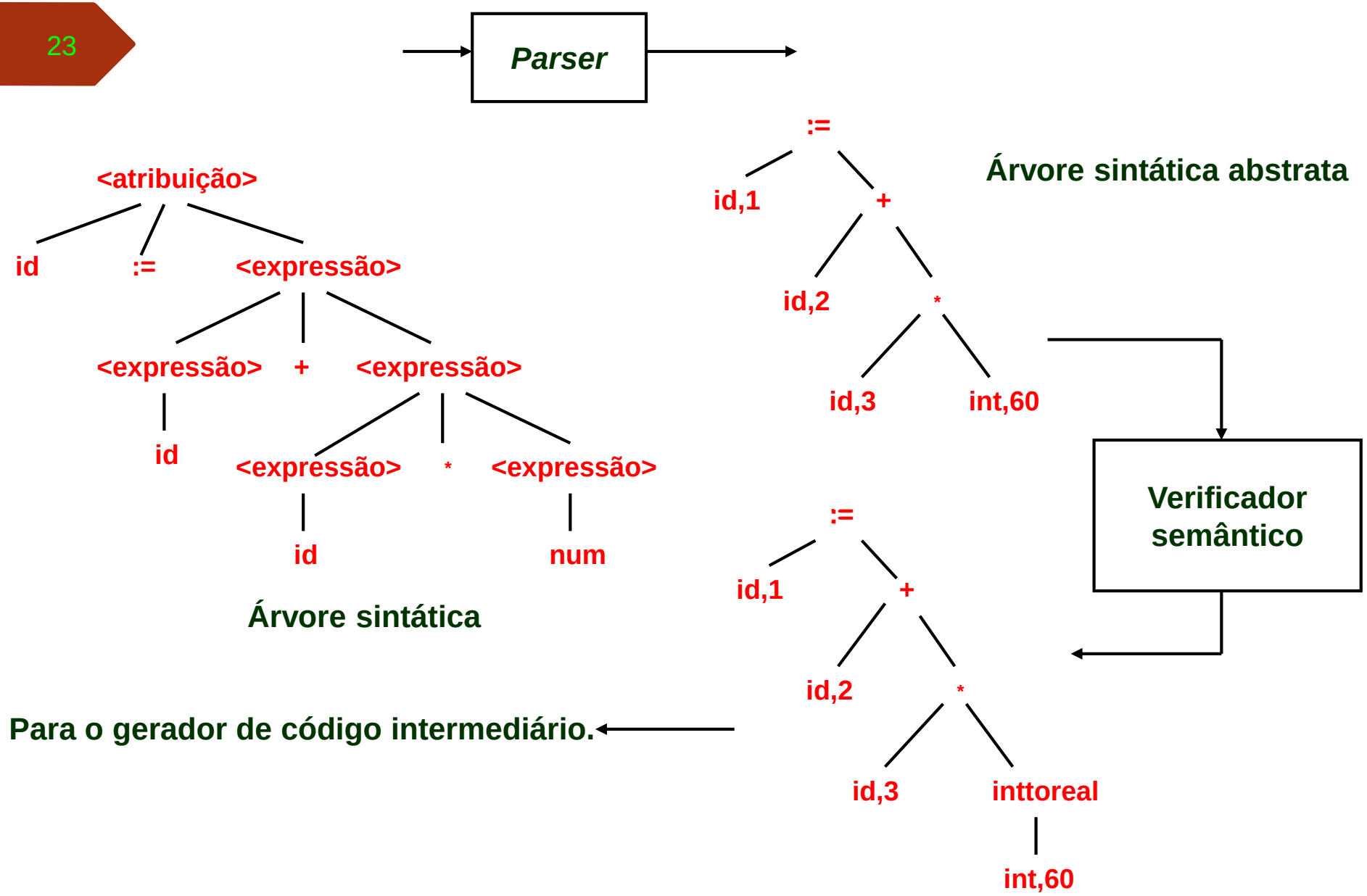
**Para o *parser***

**Tabela de Símbolos**

| | | | | |
|---|---|---|---|---|
| **1** | **posicao** | **...** | **real** | **...** |
| **2** | **inicio** | **...** | **real** | **...** |
| **3** | **veloc** | **...** | **real** | **...** |
| | **...** | | | |

# Exemplo (cont.)





# Exemplo (cont.)