

Linguagens de Programação

Aula 3

Prof. Bonfim Amaro Junior
bonfimamaro@ufc.br

Na aula passada...

- ❑ Classificação das LPs (nível, geração e paradigma)
- ❑ Paradigmas
 - Imperativo, OO, funcional, lógico e concorrente

Na aula de hoje...

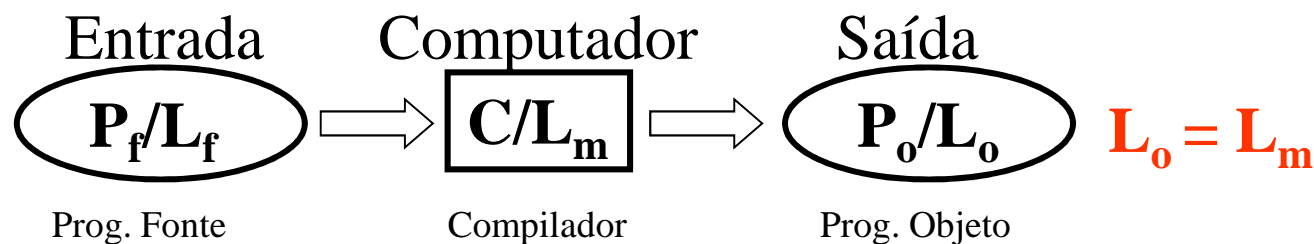
- ❑ O processo de compilação
 - O que é, para que serve e estrutura geral
- ❑ Formas de descrever uma LP

Roteiro

- ❑ Visão geral
- ❑ Compilação → análise
 1. Léxica
 2. Sintática
 3. Semântica
- ❑ Compilação → síntese
 1. Geração de código intermediário
 2. Otimização de código
 3. Geração do código objeto (linguagem máquina)
- ❑ Formas de descrever uma LP

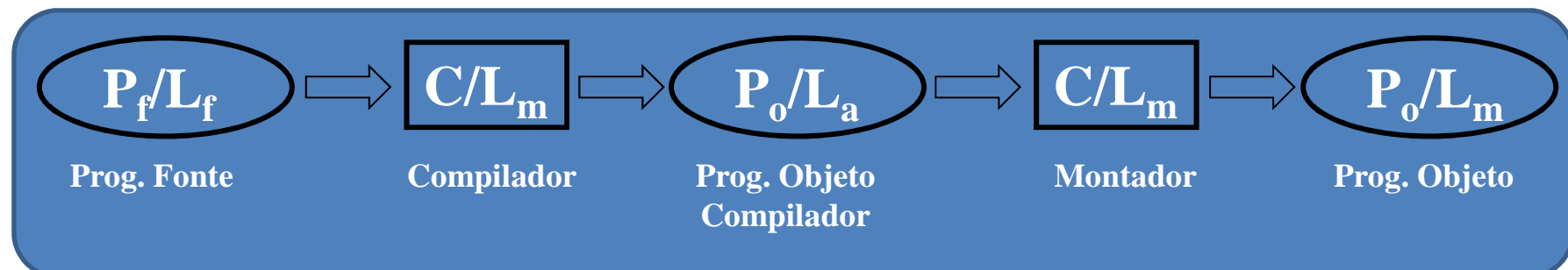
Processo de compilação

- Um **Compilador C** é um programa que tem a finalidade de **traduzir ou converter** um **programa P_f (fonte)** escrito numa **linguagem L_f** – linguagem fonte - **para** um **programa P_o (objeto)** escrito numa outra linguagem L_o - linguagem objeto; P_o é o resultado da tradução. L_m , a linguagem na qual o compilador é escrito, é em geral, a linguagem de máquina do computador onde ele é processado. Na maioria das vezes, C não é programado em L_m , sendo convertido para ela por meio de uma compilação.

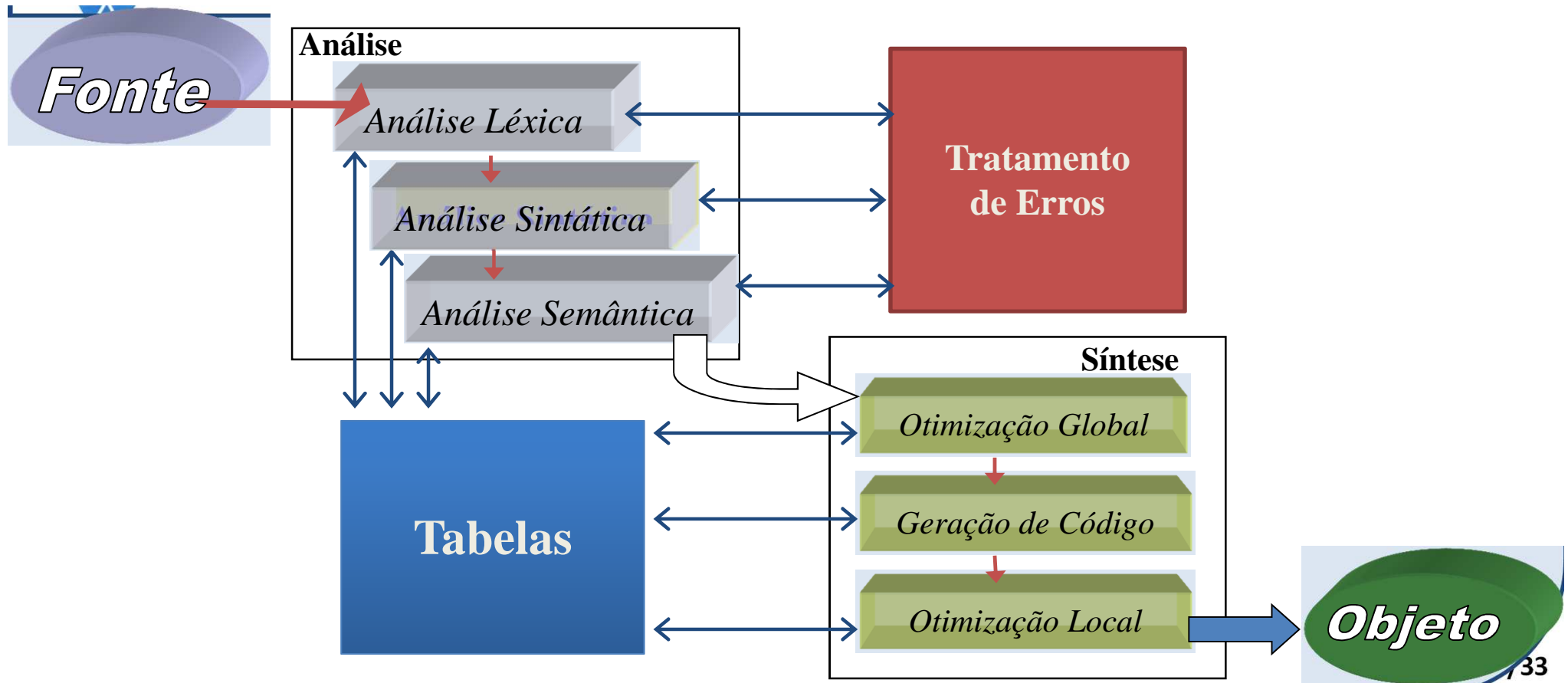


Processo de compilação

- ❑ Em geral, L_f é uma linguagem de alto nível como ALGOL, COBOL, PASCAL, etc. L_o não é necessariamente uma linguagem de máquina. Por exemplo, L_o pode ser uma Linguagem de Montagem ("*Assembler*") L_a . Nesse caso, é necessário ter-se mais uma fase de tradução de L_a para a linguagem de máquina L_m do computador a ser utilizado para processar o programa objeto.



Compilador: Arquitetura básica



Compilador: um exemplo

□ Fases da transformação

```
int gcd(int a, int b)
{
    while (a != b) {
        if (a > b) a -= b;
        else b -= a;
    }
    return a;
}
```

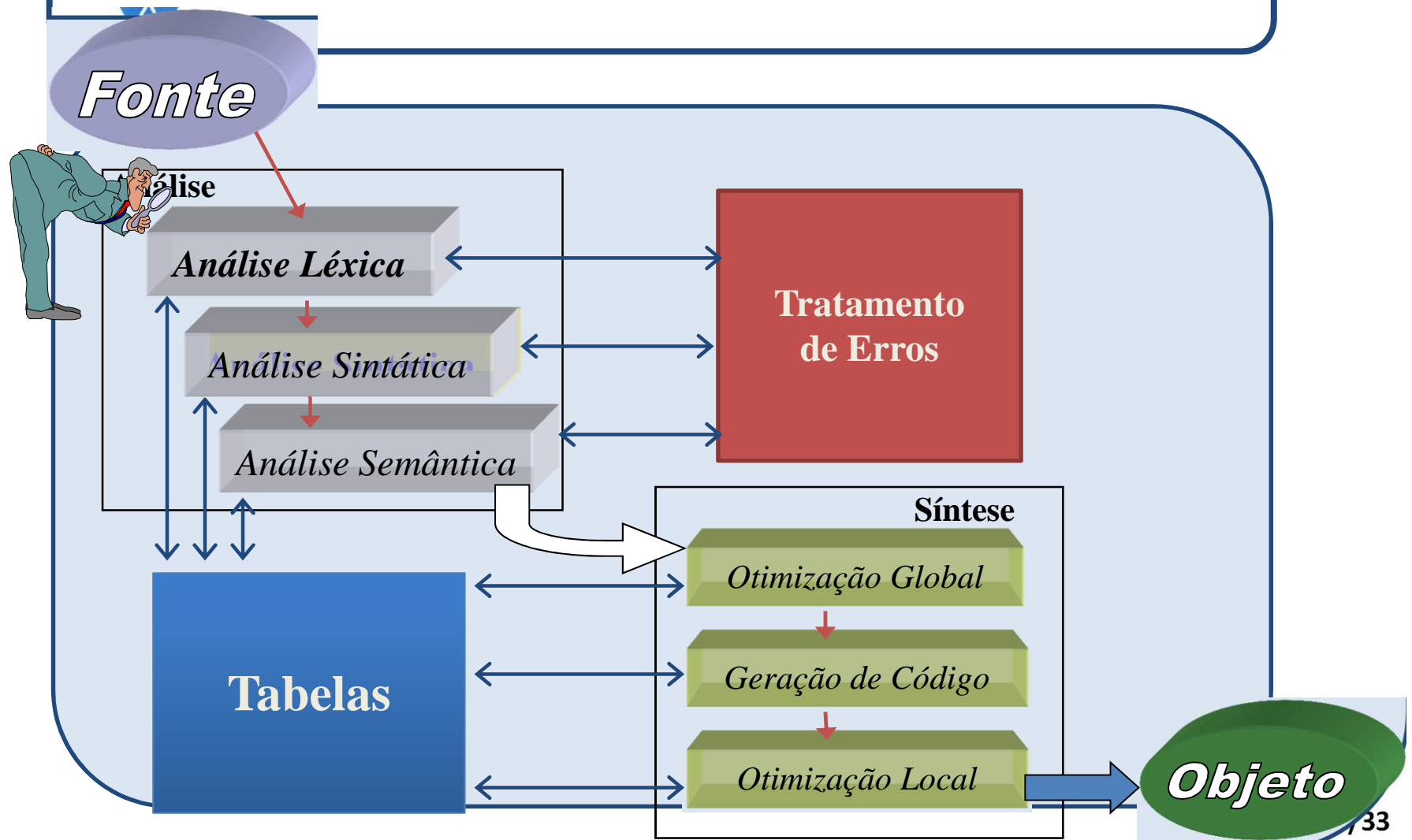

Compilador: um exemplo

- ❑ **Fases da transformação: o compilador vê o código fonte como uma sequência de caracteres**

```
int gcd(int a, int b)
{
    while (a != b) {
        if (a > b) a -= b;
        else b -= a;
    }
    return a;
}
```

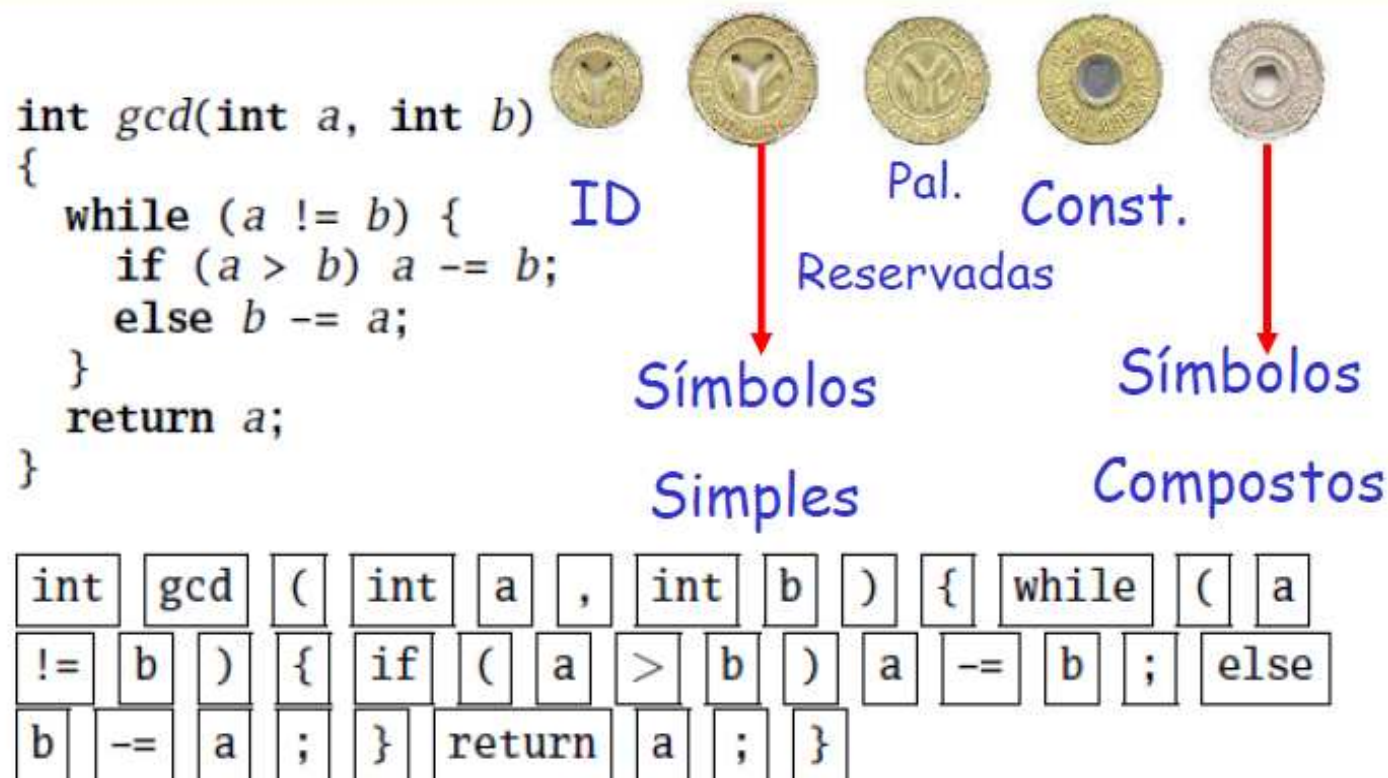
```
i  n  t  s p  g  c  d  (  i  n  t  s p  a  ,  s p  i
n  t  s p  b  )  n l  {  n l  s p  s p  w  h  i  l  e  s p
(  a  s p  !  =  s p  b  )  s p  {  n l  s p  s p  s p  s p  i
f  s p  (  a  s p  >  s p  b  )  s p  a  s p  -  =  s p  b
;  n l  s p  s p  s p  s p  e  l  s  e  s p  b  s p  -  =  s p
a  ;  n l  s p  s p  }  n l  s p  s p  r  e  t  u  r  n  s p
a  ;  n l  }  n l
```

Compilador: Arquitetura básica

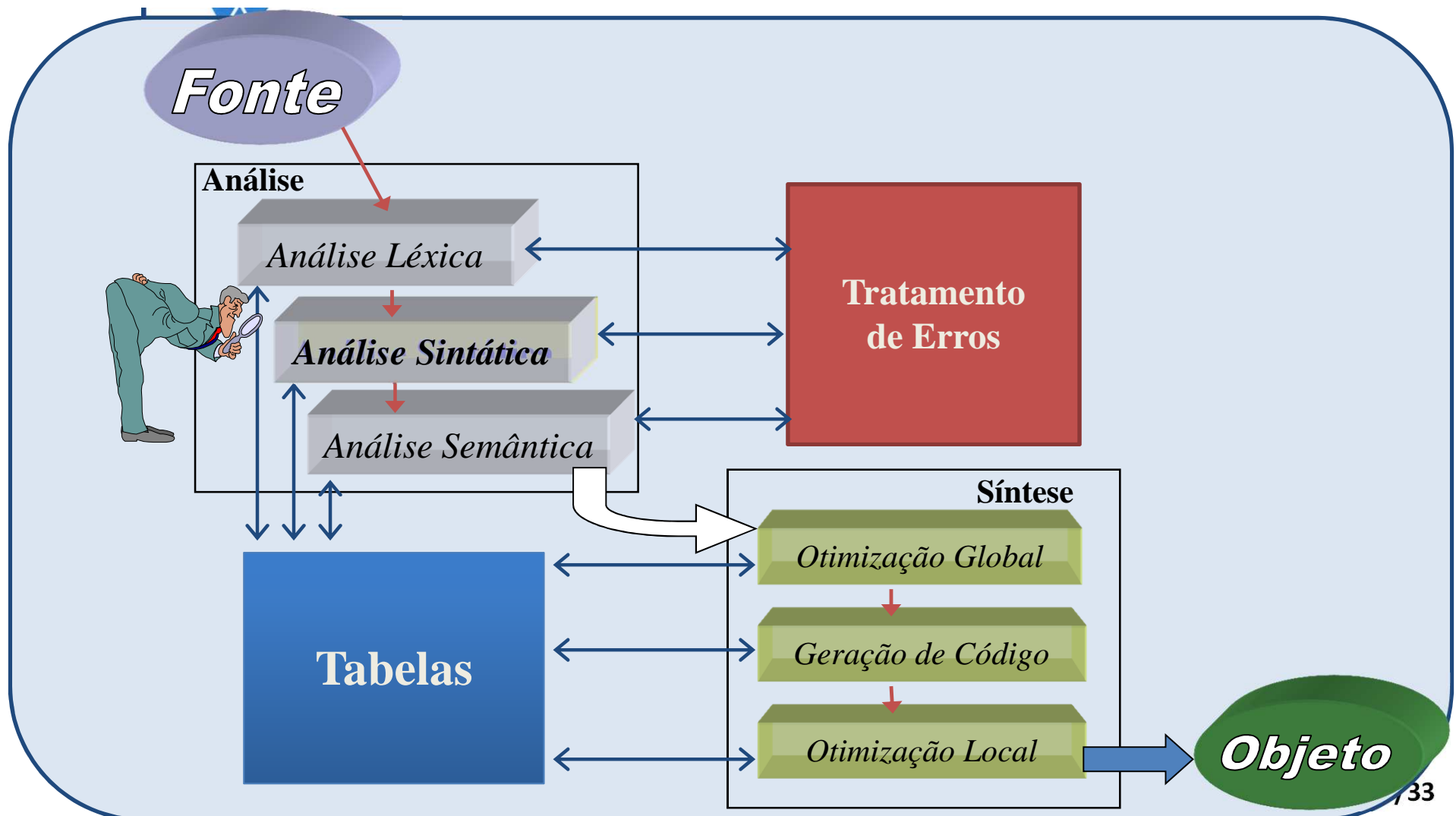


Compilador: um exemplo

- A **análise léxica** fornece *tokens*: uma cadeia de *tokens* sem espaços e comentários

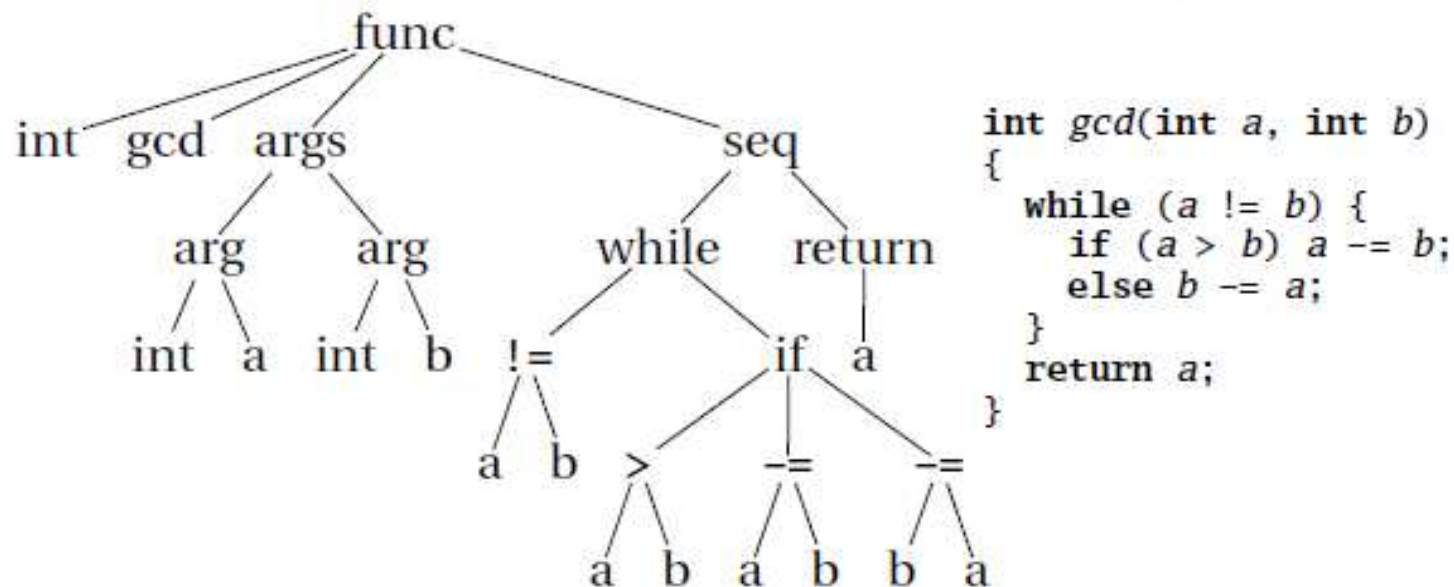


Compilador: Arquitetura básica

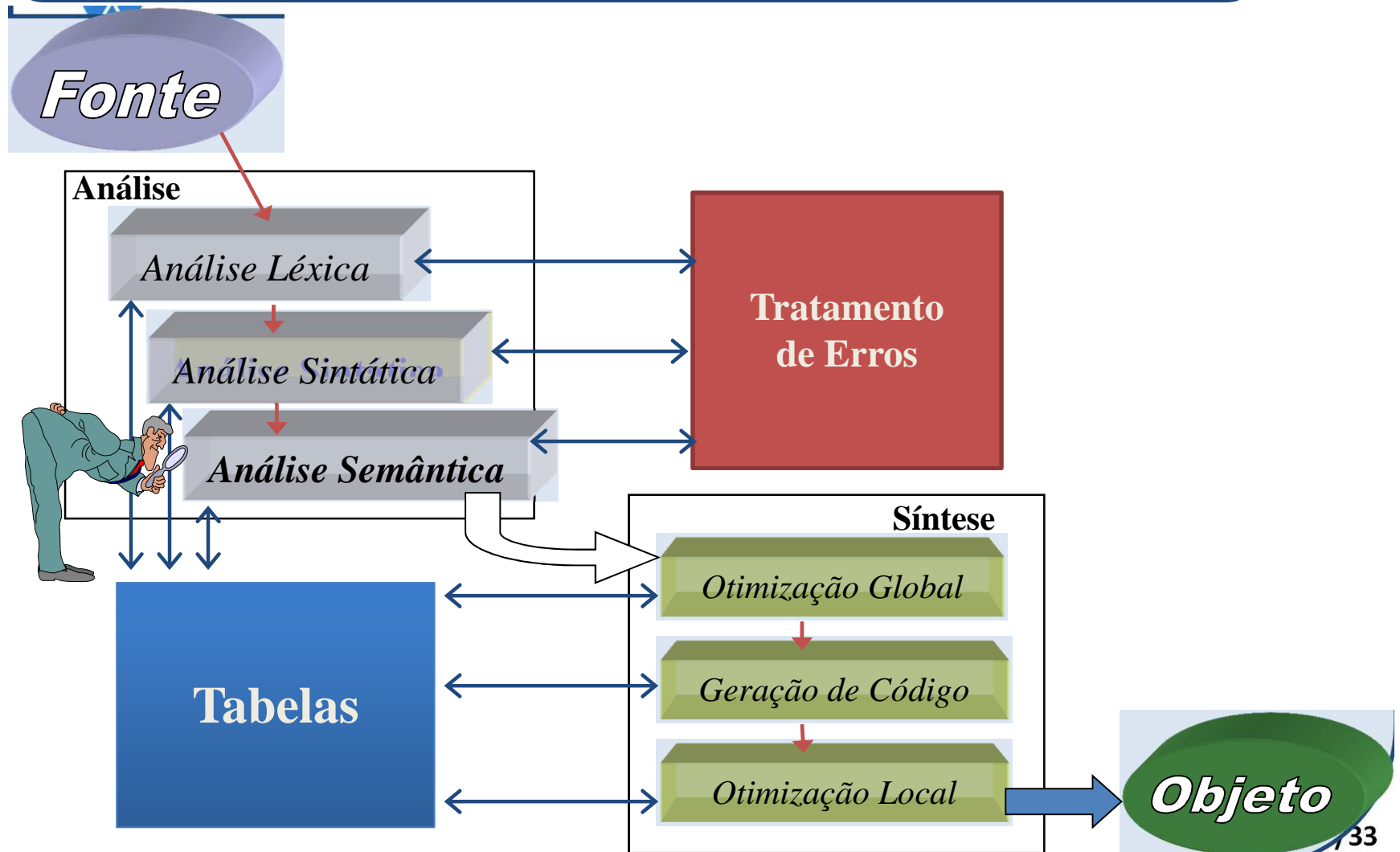


Compilador: um exemplo

- ❑ A **análise sintática** fornece uma árvore abstrata construída a partir das regras da **gramática**

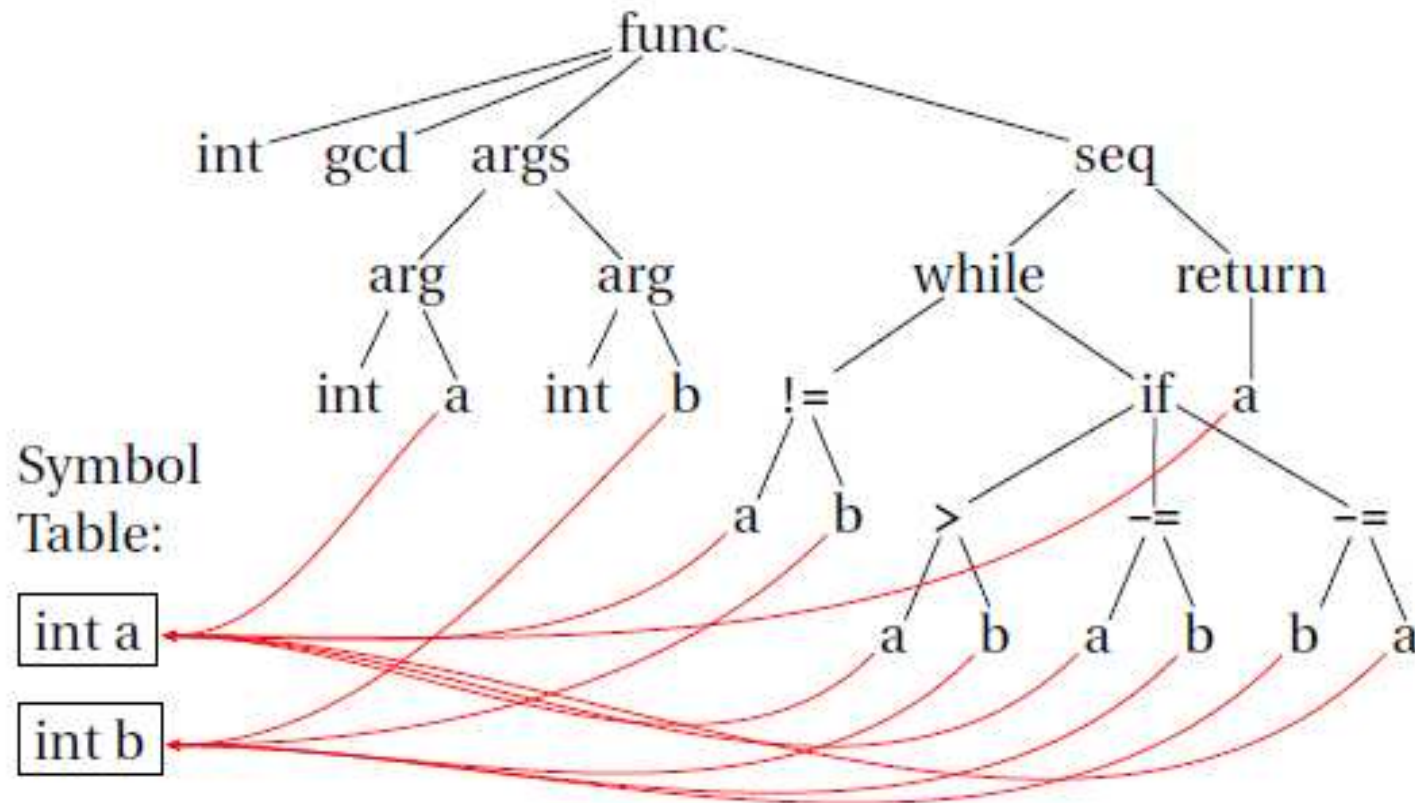


Compilador: Arquitetura básica

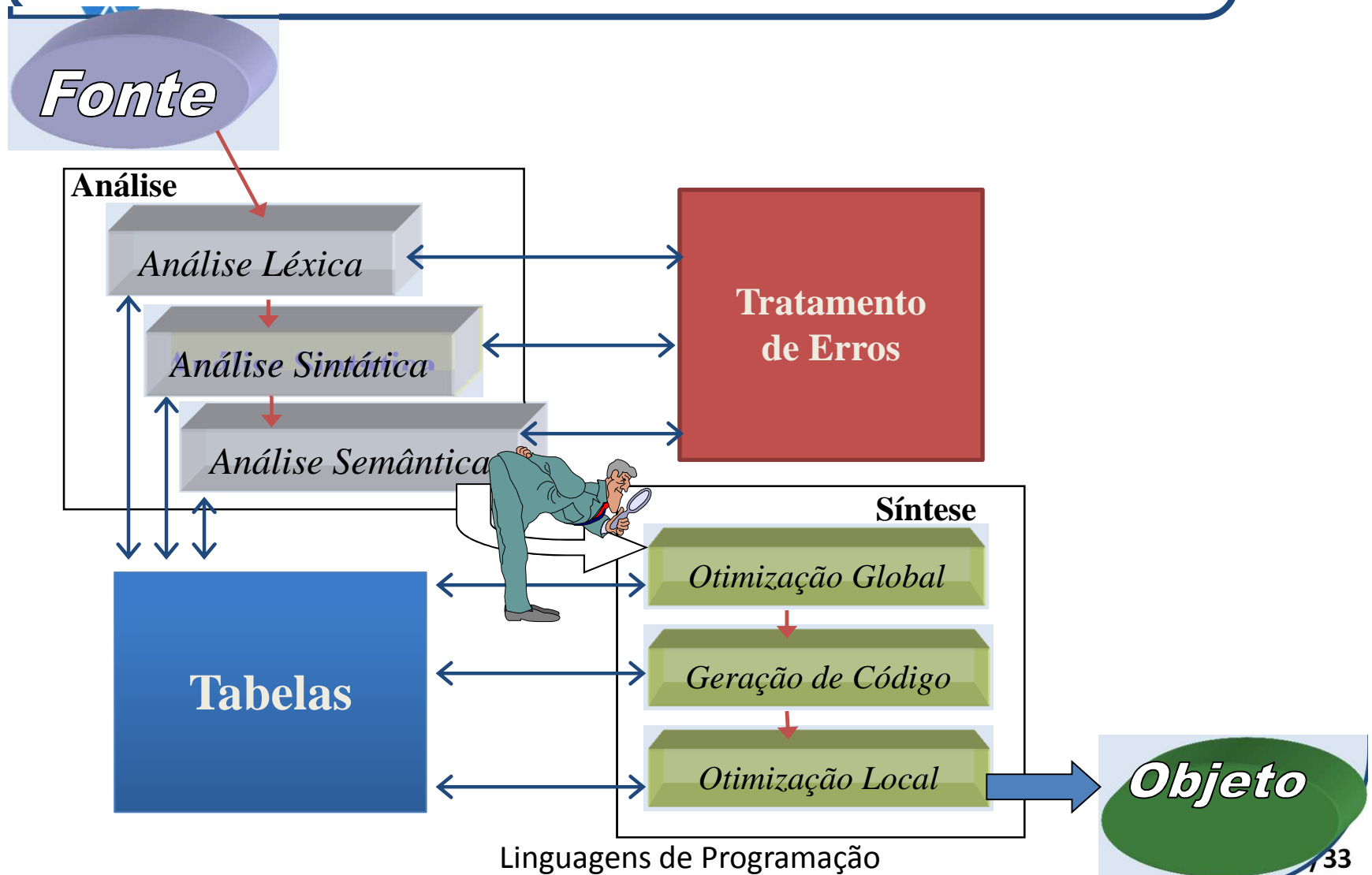


Compilador: um exemplo

- ❑ A **análise semântica** resolve símbolos, com tipos checados



Compilador: Arquitetura básica



Compilador: um exemplo

❑ Tradução para **linguagem intermediária**.

➤ Código de 3 endereços

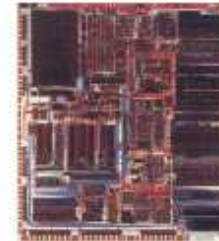
```
L0: sne    $1,  a,  b
      seq    $0, $1, 0
      btrue  $0, L1    % while (a != b)
      sl     $3,  b,  a
      seq    $2, $3, 0
      btrue  $2, L4    % if (a < b)
      sub    a,   a,  b % a -= b
      jmp    L5
L4: sub    b,   b,  a % b -= a
L5: jmp    L0
L1: ret    a
```

```
int gcd(int a, int b)
{
    while (a != b) {
        if (a > b) a -= b;
        else b -= a;
    }
    return a;
}
```

Compilador: um exemplo

- ❑ Geração de código – linguagem de montagem *Assembly 80386*

```
gcd:  pushl %ebp                % Save FP
      movl %esp,%ebp
      movl 8(%ebp),%eax         % Load a from stack
      movl 12(%ebp),%edx        % Load b from stack
.L8:  cmpl %edx,%eax
      je   .L3                 % while (a != b)
      jle  .L5                 % if (a < b)
      subl %edx,%eax           % a -= b
      jmp  .L8
.L5:  subl %eax,%edx           % b -= a
      jmp  .L8
.L3:  leave                    % Restore SP, BP
      ret
```



Compilador

- ❑ Um **compilador** é um programa que **transforma** um outro **programa** escrito em uma **linguagem** de **programação** de alto **nível** qualquer em instruções que o computador é capaz de entender e executar.

Foco na linguagem fonte

Linguagens de programação

- ❑ Uma LP é uma linguagem destinada para ser usada por uma **pessoa** para expressar um **processo** através do qual um **computador** pode resolver um **problema**
- ❑ Dependendo da perspectiva, têm-se
 - ❑ **Pessoa** = paradigma lógico/declarativo
 - ❑ **Processo** = paradigma funcional
 - ❑ **Computador** = paradigma imperativo
 - ❑ **Problema** = paradigma orientado a objetos

Linguagens de programação

❑ Estilo/forma de programação

➤ Imperativa

- ✓ especifica-se como uma **computação** deve ser feita para resolver **problemas**, passo a passo, via execução de instruções. Ex: C, Pascal, Java,...

➤ Declarativa

- ✓ especifica-se qual computação deve ser feita para resolver problemas. Ex: Haskell, Prolog, ML,...

Roteiro

- ❑ Visão geral
- ❑ Compilação → análise
 1. Léxica
 2. Sintática
 3. Semântica
- ❑ Compilação → síntese
 1. Geração de código intermediário
 2. Otimização de código
 3. Geração do código objeto (linguagem máquina)
- ❑ Formas de descrever uma LP

LP: sintaxe e semântica

❑ A **descrição de uma LP** envolve dois aspectos principais:

❑ **Sintaxe:** conjunto de regras que determinam quais construções são corretas

❑ **Semântica:** descrição de como as construções da linguagem devem ser interpretadas e executadas

❖ Em Pascal: $a := b$

Sintaxe: comando de atribuição correto

Semântica: substituir o valor de a pelo valor de b

LP: sintaxe e semântica

exemplo

- ❑ Analise o código em linguagem C abaixo e verifique se existem

erros.

```
1. int j=0, conta, V[10]; float i@;  
2. conta = '0'  
3. for (j=0, j<10; j++  
4. {  
5.     V[j] = conta++;  
6. }
```

O compilador tem a responsabilidade de reportar ERROS!

❖ É necessário algum recurso para identificá-los

O que diferencia os tipos de ERROS?

LP: sintaxe e semântica

exemplo

- ❑ Esses erros são verificados e diferenciados durante a fase de **análise** da compilação
 - ❑ **léxica:** palavras (*tokens*) do programa
 - ❑ i, j, for, =, (, <, int, ++, conta, V[]
 - ❑ **Erro:** i@
 - ❑ **sintática:** combinação de *tokens* que formam o programa
 - ❑ comando_for → for (expr1; expr2; expr3) {comandos}
 - ❑ **Erros:** ; for(**j=0**,...)
 - ❑ **semântica:** adequação do uso
 - ❑ Tipos semelhantes em comandos (atribuição, por exemplo), uso de identificadores declarados
 - ❑ **Erro:** conta = '0'

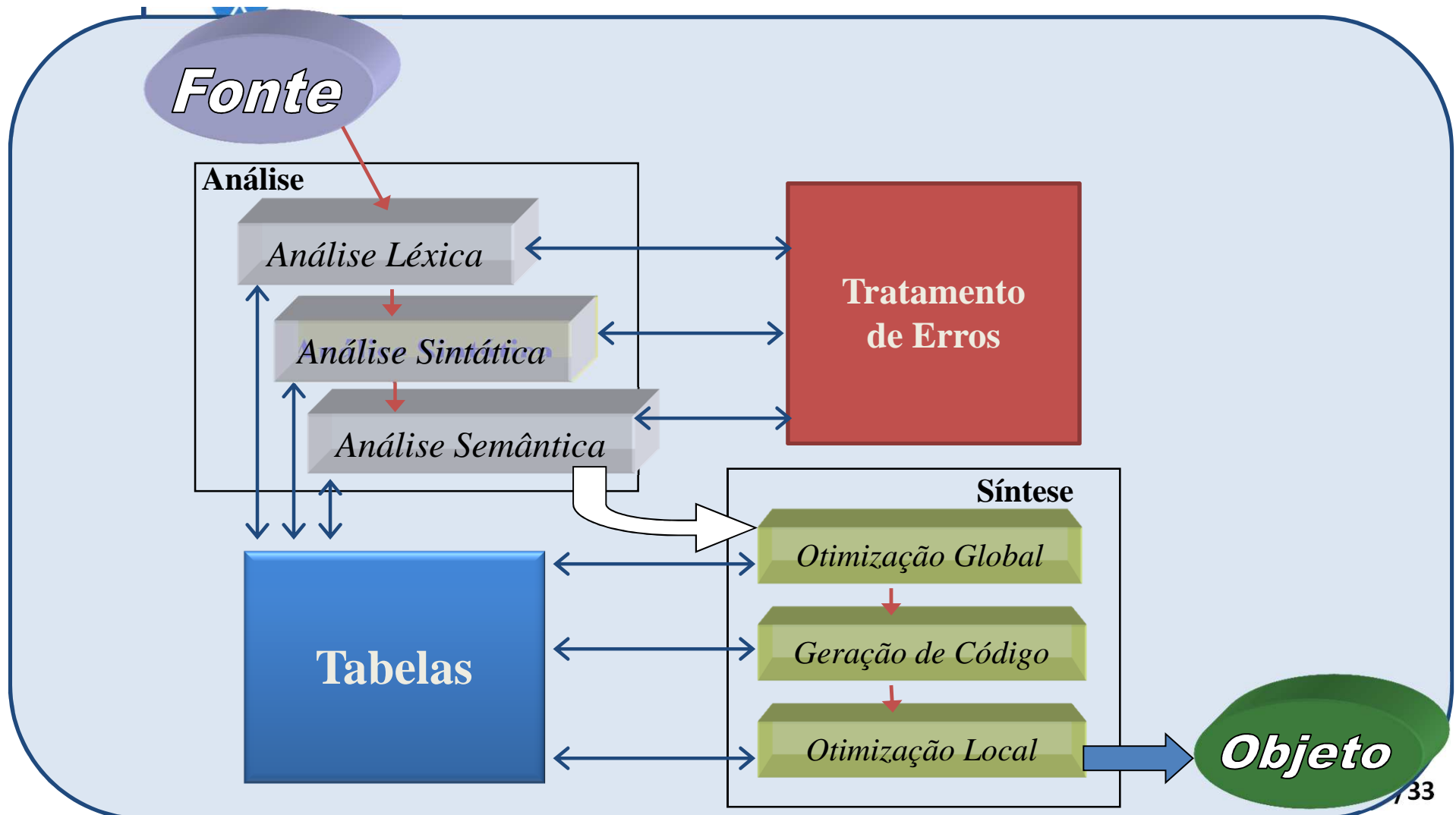
Estruturas da compilação

❑ Como diferenciar as palavras e símbolos reservados (for, int, float, =) de identificadores definidos pelo usuário?

➤ Tabela de palavras e símbolos reservados

for
int
float
...

Estruturas da compilação



Estruturas da compilação

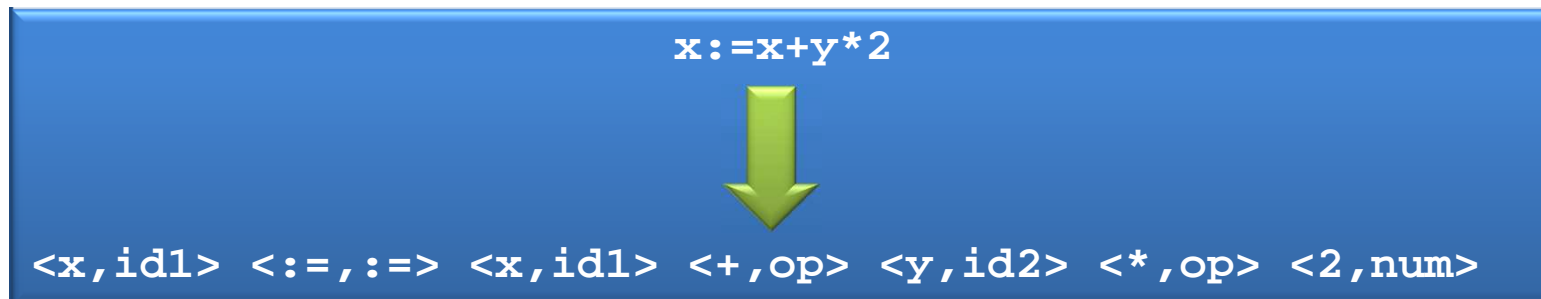
- ❑ Como saber durante a compilação de um programa o tipo e o valor dos identificadores, escopo das variáveis, número e tipo dos parâmetros de um procedimento, etc.?

identificador	Classe	Tipo	Valor	...
j	var	int	0	
fat	function	-	-	
...	

Compilação passo-a-passo

❑ **Análise léxica**: atua no reconhecimento e classificação dos *tokens* (palavras)

➤ Expressões regulares e autômatos finitos

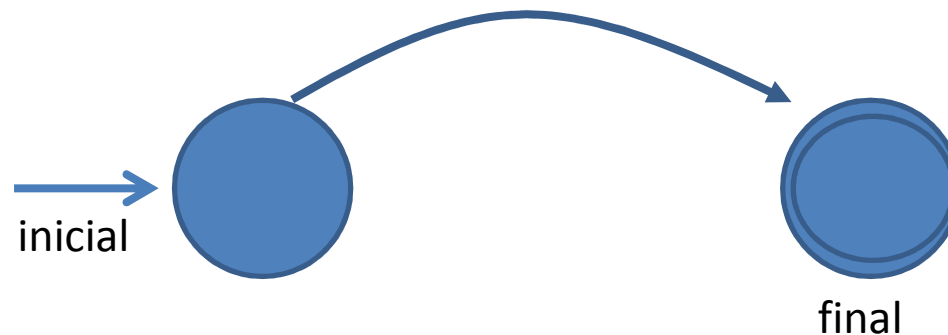


Compilação passo-a-passo

❑ **Análise léxica:** pode ser feita através de autômatos finitos (AF) ou expressões regulares

❑ AF é uma máquina de estados finitos. Formada por um conjunto de estados (um estado inicial e um ou mais estados finais)

❑ Simulador Jflap



➔ Estado inicial indicado por uma seta

Estado final representado por um círculo com 2 bordas

Análise léxica

Autômatos finitos

❑ **AF:** exemplos...

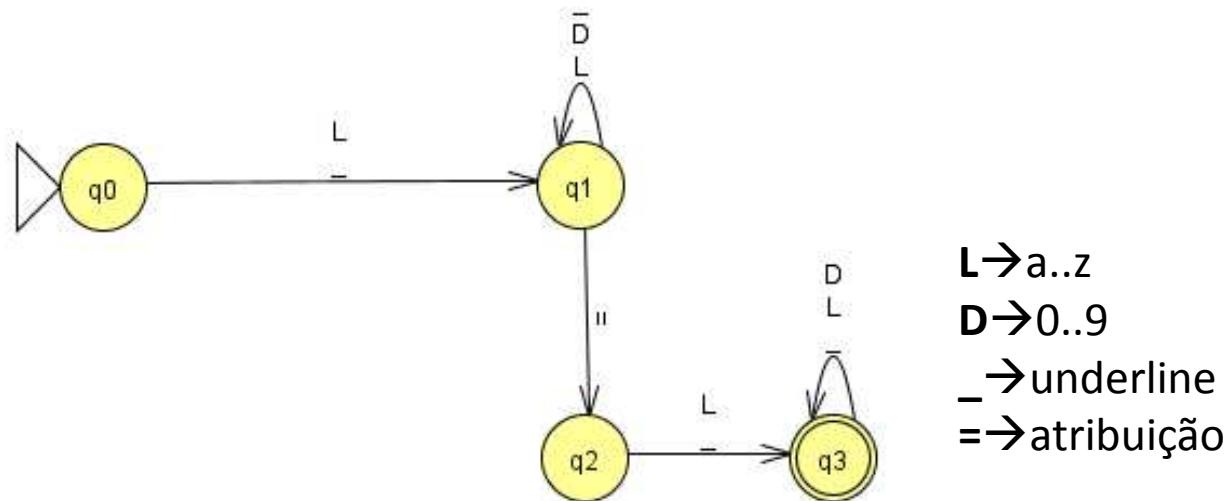
1. Cadeia de caracteres a,b,c
2. Números inteiros (com ou sem sinal)
3. Números reais (com ou sem sinais)
4. Identificador
5. Comparação (>, >=, <, <=, !=) entre dois identificadores
6. Atribuição (=) entre dois identificadores
7. Comando IF → `if(condicao){ comandos;} else { comandos;}`
8. Comando While

Compilação passo-a-passo

❑ **Análise léxica:** pode ser feita através de autômatos finitos (AF) ou expressões regulares

❑ Simulador Jflap

❑ Exemplos: AF para números inteiros, reais, identificadores, atribuição entre duas variáveis



IMPORTANTE

- ❑ Para próxima aula leiam os itens 3.1 a 3.4 do livro do SEBESTA