SCC 605 Teoria da Computação e Compiladores

Teoria da Computação

1. Teoria das Linguagens Formais e dos Autômatos
2. Teoria da Computabilidade
3. Teoria da Complexidade (não veremos)

Quais são as capacidades fundamentais e limitações dos computadores?

Objetivos do curso

1) Dar subsídios para a definição de uma linguagem de programação, ou uma linguagem dedicada,

   - definir sua gramática e seu reconhecedor
     - Construir reconhecedores de programas em uma determinada linguagem de alto nível (Análise/ Front-end) para posteriormente gerar o seu código de máquina (Síntese/ Back-end) – Compiladores.

   - apresentar as características, propriedades e aplicações das linguagens da hierarquia de Chomsky e de seus reconhecedores

2) Estudar a meta-teoria da computação, ou seja, definir o que a teoria estuda e estabelecer suas limitações: problemas indecidíveis e intratáveis (não veremos)

- Definir o que a teoria estuda é definir o que é computável.
  - Faremos uma abordagem intuitiva de computabilidade usando a noção de procedimento (efetivo) e definiremos também o conceito por meio de modelos formais: o mais famoso é a Máquina de Turing (MT).

- Apresentar a hipótese de Church-Turing (1936):

" Uma função é computável se é computável por MT" que formaliza a noção intuitiva de procedimento efetivo.

"A capacidade de computação representada pela Máquina de Turing é o limite máximo que pode ser atingido por qualquer dispositivo de computação, independentemente do curso de desenvolvimento da tecnologia" .

3) Diferenciar procedimento de algoritmo que é um procedimento que sempre pára

- apresentar linguagens (ou problemas) reconhecidas por MT (recursivamente enumeráveis) e
- decididas por MT (recursivas).
- Mostrar que tipos de problemas algoritmicos são insolúveis/indecidíveis

Compiladores

1. Apresentar conceitos e métodos para as fases de análise e síntese de um compilador

2. Implementar um Front-end para uma linguagem de programação simples (FRANKIE: Pascal + C simplificados) via ferramentas para construção de compiladores (JavaCC)

II – BIBLIOGRAFIA

- **Teoria da Computação:**

- Hopcroft, Motwani & Ullman: Introduction to Automata Theory, Languages, and Computation, 2nd. Edition. Addison-Wesley, 2001. Errata do livro em: http://www-db.stanford.edu/~ullman/ialc.html#errata.

- Sipser, M. Introduction to the Theory of Computation. PWS, 1997. (2a edição). Errata do livro em: http://www-math.mit.edu/~sipser/itoc-errs1.2.html

- Divério & Menezes. Teoria da Computação – Máquinas Universais e Computabilidade. Série Livros Didáticos 5, IF UFRGS, 2ª edição, 2000, editora SagraLuzzatto.

- Menezes, P.B. Linguagens Formais e Autômatos. Série Livros didáticos 3, IF UFRGS, 4ª edição, 2001, editora SagraLuzzatto. e-Book de Linguagens Formais & Autômatos.

- Harel, D. Algorithmics – The Spirit of Computing. Addison-Wesley Publishing Company, 1992 (2ª edição). (Existem 3 edições similares do livro na Biblioteca do ICMC: a primeira, a sua versão reduzida e a segunda edição que traz exercícios).

- **Compiladores:**
- Aho, A. V. et alli - Compilers: Principles, Techniques and Tools. Addison-Wesley Publishing Company, 1986. (Dragãozinho em INGLÊS)
- Aho, A. V. et alli - Compiladores - Princípios, Técnicas e Ferramentas, 2ª Edição, Ed. Pearson, 2007.
- Aho, A.V.; Ullman, J.D.; Sethi, R. (1995). Compiladores: Princípios, Técnicas e Ferramentas. Editora LTC. (Dragãozinho em PORTUGUÊS)
- Kowaltowsky, T. - Implementação de Linguagens de Programação, Guanabara Dois, 1983.
- Louden, K.C. (2004). Compiladores: Princípios e Práticas. Editora Thomson Learning.
- Price, A.M.A. e Toscani, S.S. (2001). Implementação de Linguagens de Programação: Compilador. Editora Sagra Luzzatto.

III - CRITÉRIO DE AVALIAÇÃO

- **PROVAS**: Haverá 2 provas e cada uma vale de 0 a 10. **Não teremos prova substitutiva.**

  - **1º Prova: 29/4; 2º Prova: 21/6**

- **TRABALHOS PRÁTICOS**: Haverá 3 Trabalhos Práticos (Projetos 1, 2, 3), todos relacionados com a linguagem **FRANKIE** para a qual construiremos o **front-end** de um compilador.

- Os trabalhos serão desenvolvidos por uma equipe de 4 alunos.

- Cada equipe receberá a tarefa de:

- **T1**: aumentar a gramática original de FRANKIE com extensões + gerar o analisador léxico a partir do gerador de compiladores JavaCC

- **T2**: gerar o analisador sintático a partir do gerador de compiladores JavaCC

- **T3**: implementar as rotinas de checagens de tipos e análise semântica para a gramática estendida do grupo como ações no arquivo de especificação da gramática de FRANKIE

- **Cada trabalho vale de 0 a 10.**
- **Cada dia de atraso -1, até 5 dias**

- **LISTAS DE EXERCÍCIOS.** Haverá várias listas, disponíveis na Wiki do curso.
-
- **CÁLCULO DA MÉDIA**:
- MP = Média Aritmética das Provas
- MT = Média Ponderada dos Trabalhos
- (T1 15%; T2 30%; T3 55%)

- MF = Média Final:
- **Se MP >= 5 então MF = (6MP +4MT)/10**
- **senão MF = (7.5MP + 2.5MT)/10**
-

Meet Frankie!

- Para a implementação desse projeto será fornecida a sintaxe da linguagem fonte – FRANKIE – em notação EBNF.

- Os alunos farão 1 extensão por grupo a esta gramática, por sorteio.

13 Extensões

- Tipos (e suas constantes):

1. cadeia de caracteres (string) – C
2. caractere – C
3. real (simples e duplo)– C
4. registro (struct) – C
5. união (union)– C
6. enumeração (enum) – C
7. vetores de uma dimensão -- C

13 Extensões

- **Comandos**:

1. caso (case) -- PASCAL
2. repita (do while) – C
3. para (for) -- PASCAL

13 Extensões

- **Definições**:

1. definição de tipo (typedef) – C
2. definição de constante (CONST) –PASCAL
3. definição de função – C
   (LEMBREM-SE que a passagem de parâmetros vem de PASCAL)

Vamos sortear as extensões

- Formem os grupos

Material de Suporte para as extensões

Gramática Pascal EBNF: http://www.lrz.de/~bernhard/Pascal-EBNF.html
Gramática Pascal YACC: http://www.moorecad.com/standardpascal/pascal.y

Gramática C enxuta (menos não-terminais) EBNF:
http://lists.canonical.org/pipermail/kragen-hacks/1999-October/000201.html
Gramática C YACC: http://www.lysator.liu.se/c/ANSI-C-grammar-y.html

Comparação entre linguagens C e PASCAL:
http://en.wikipedia.org/wiki/Comparison_of_Pascal_and_C
Tutorial Pascal:
http://webcourse.cs.technion.ac.il/234319/Spring2009/ho/WCFiles/Tutorial%201%20Pascal%20EBNF.pdf

Conceitos Básicos

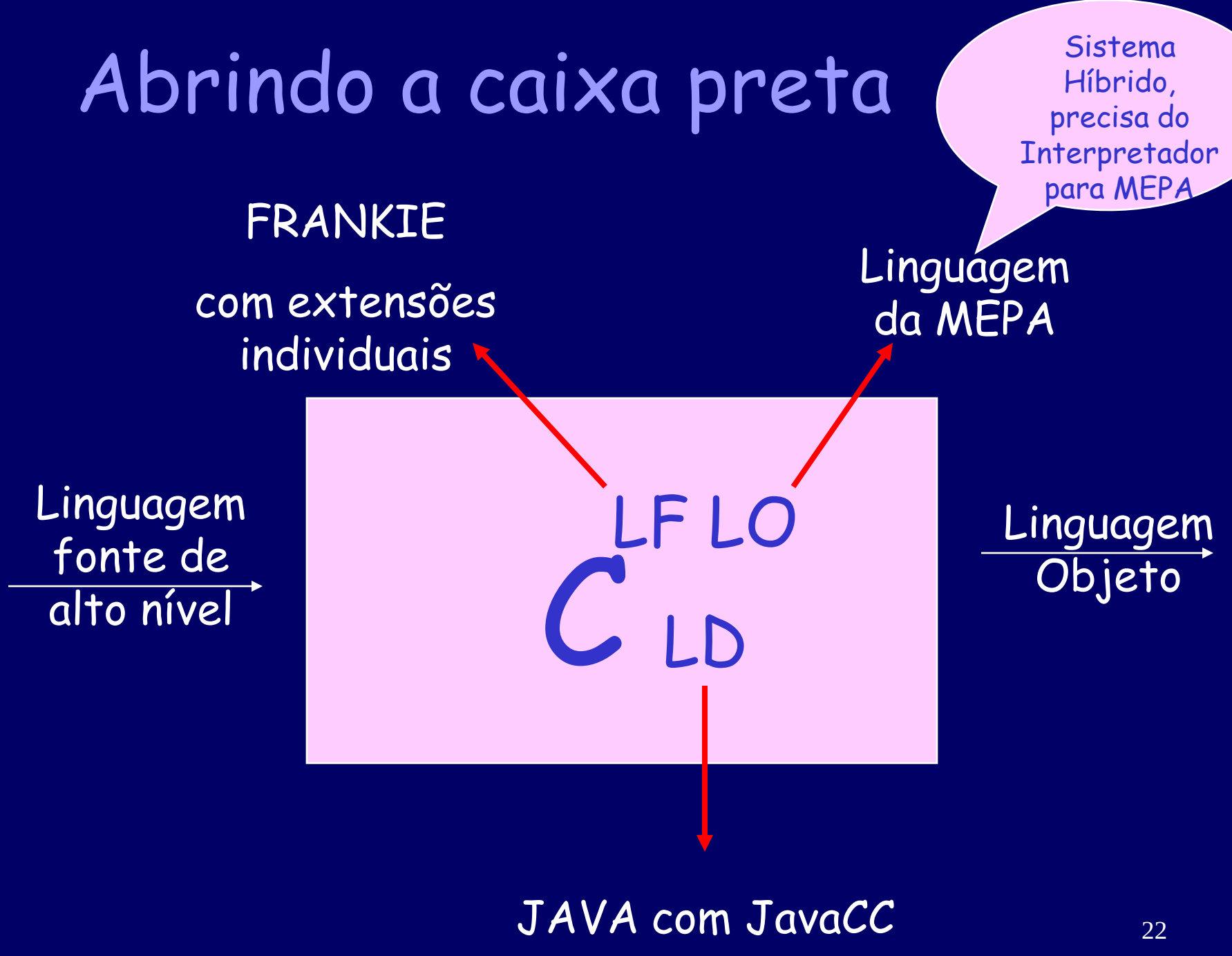
- Compiladores

Compilador

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

Como transforma?

Abrindo a caixa preta



Compilador tem responsabilidade de reportar erros!

Exercício: Leia o texto abaixo e identifique os erros cometidos

Perky viveu as duas semanas mais duras de sua vida

Pata sobrevive a bala, geladeira, cirurgia e parada cardíaca

Uma pata tida como morta depois de ser baleada em uma caçada, que passou dois dias na geladeira e ainda sofreu uma arada cardíaca na mesa de operações, sobreviveu e passa bem.

Perky, a pata, ganhou os noticiário do estado americano da Flórida quando foi encontrada viva na geladeira de um caçador dois dias depois de baleada.

Veterinários que tentaram reparar os danos na asa de Perky conseguiram ressuscitar a pata em plena mesa de operações, quando o coração dela parou de bater. Perky não sobreviveu a uma segunda parada cardíaca.

Perky agora tem um pino em sua asa e os veterinários esperam que tenha uma boa recuperação. Rex não teve tanta sorte.

Erros cometidos: léxico, sintático, semântico/lógica e de referência

Perky viveu as duas semanas mais duras de sua vida

Pata sobrevive a bala, geladeira, cirurgia e parada cardíaca

Uma pata tida como morta depois de ser baleada em uma caçada, que passou dois dias na geladeira e ainda sofreu uma arada cardíaca na mesa de operações, sobreviveu e passa bem.

Perky, a pata, ganhou os noticiário do estado americano da Flórida quando foi encontrada viva na geladeira de um caçador dois dias depois de baleada.

Veterinários que tentaram reparar os danos na asa de Perky conseguiram ressuscitar a pata em plena mesa de operações, quando o coração dela parou de bater. Perky não sobreviveu a uma segunda parada cardíaca.

Perky agora tem um pino em sua asa e os veterinários esperam que tenha uma boa recuperação. Rex não teve tanta sorte.

Agora, encontre os erros no programa escrito em Pascal abaixo:

```
program super_util;
var idade, contador, n: integer;
begin
    writeln('Digite sua idade');
    readln(idade);
    n:=0;
    contador:=1;
    while (contador<=idade do
        begin
        if contadr mod 2 = 0 then n:=n+1;
        contador:=contador+1;
        end;
    write('Você já teve o seguinte número de anos pares em sua vida: ');
    writeln(contador);
    n:=0;
end.
```

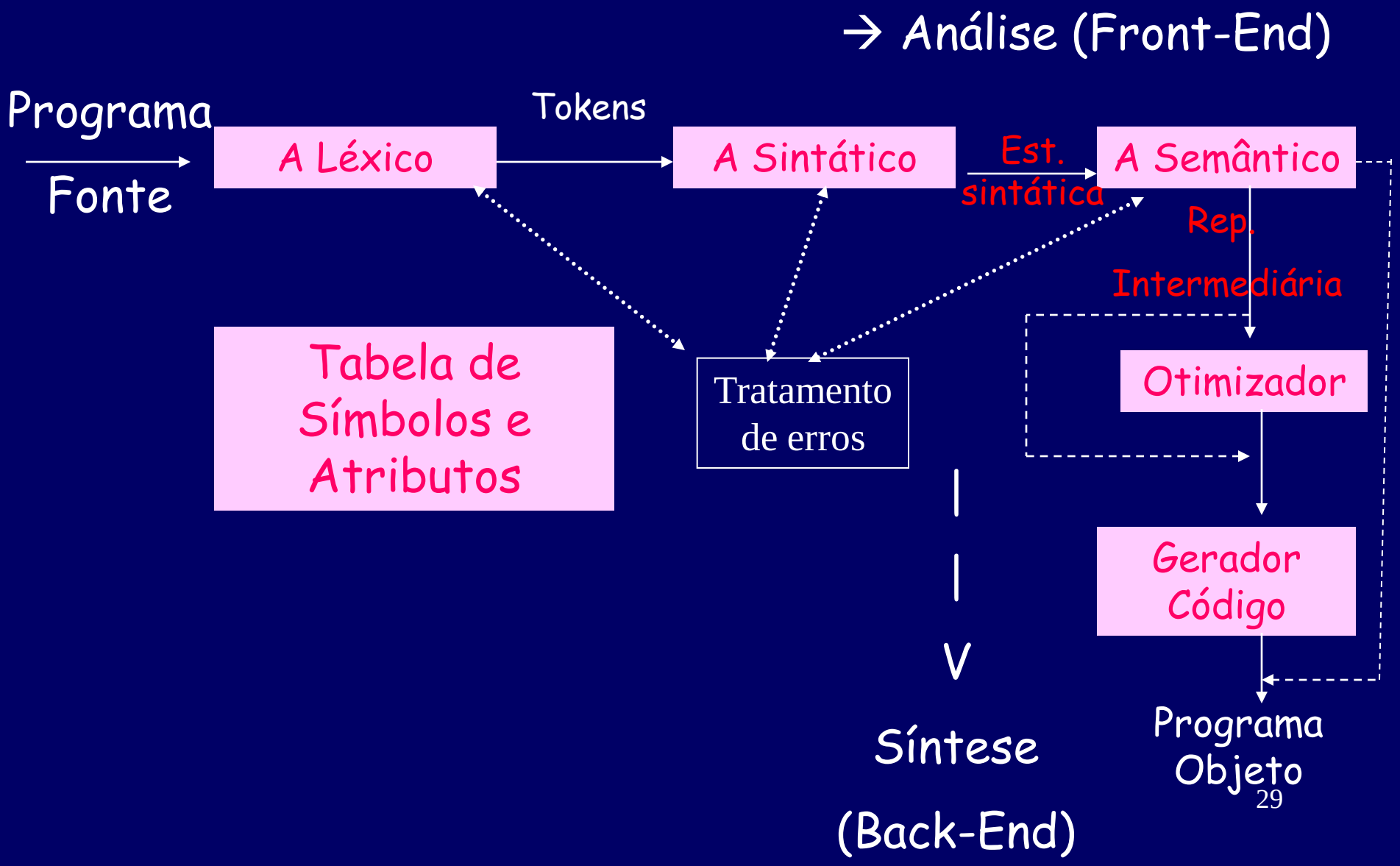
Erros no programa: sintático, semântico, lógica, semântico

```
program super_util;
var idade, contador, n: integer;
begin
    writeln('Digite sua idade');
    readln(idade);
    n:=0;
    contador:=1;
    while (contador<=idade do
        begin
        if contadr mod 2 = 0 then n:=n+1;
        contador:=contador+1;
        end;
    write('Você já teve o seguinte número de anos pares em sua vida: ');
    writeln(contador);
    m:=0;
end.
```

Há semelhanças?

- Os erros no texto e no programa pertencem a níveis diferentes?
- Isto é, precisamos de recursos diferentes para identificar cada um deles?

Estrutura de um Compilador



Compilador Ilustrado

- Uma figura fala mais do que mil palavras...
- Mostrando as fases da transformação

Compilando um programa simples

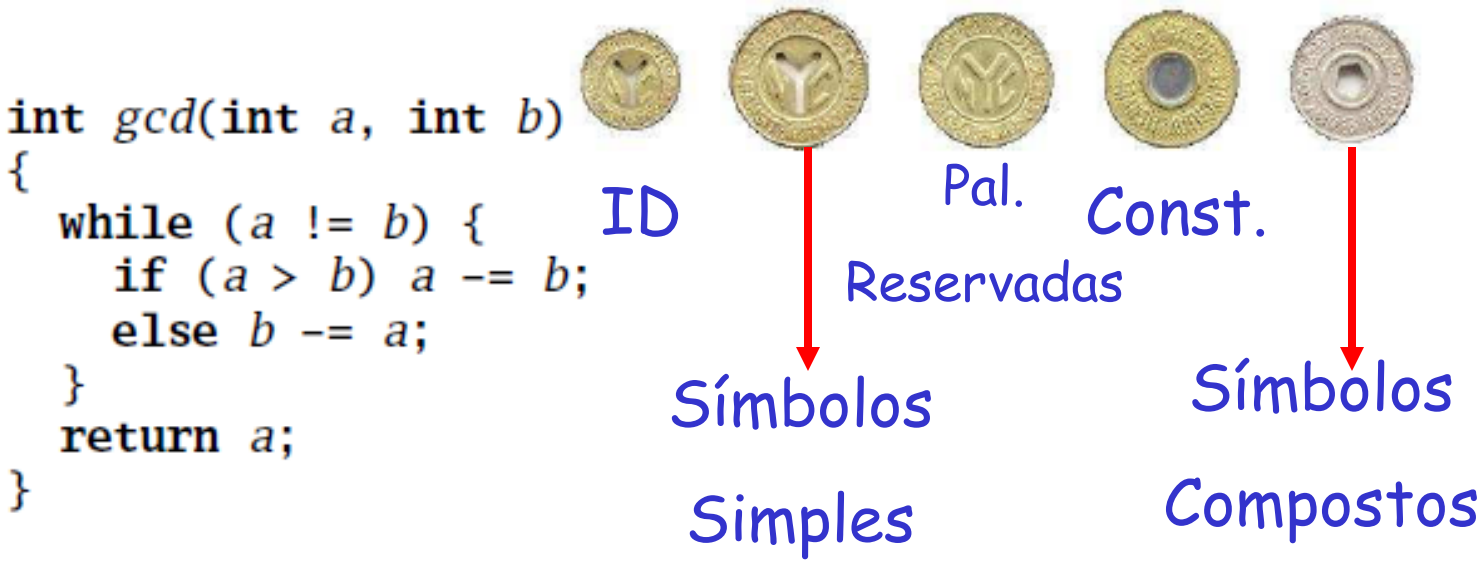
```
int gcd(int a, int b)
{
  while (a != b) {
    if (a > b) a -= b;
    else b -= a;
  }
  return a;
}
```

O que o compilador vê: o texto é uma sequência de caracteres

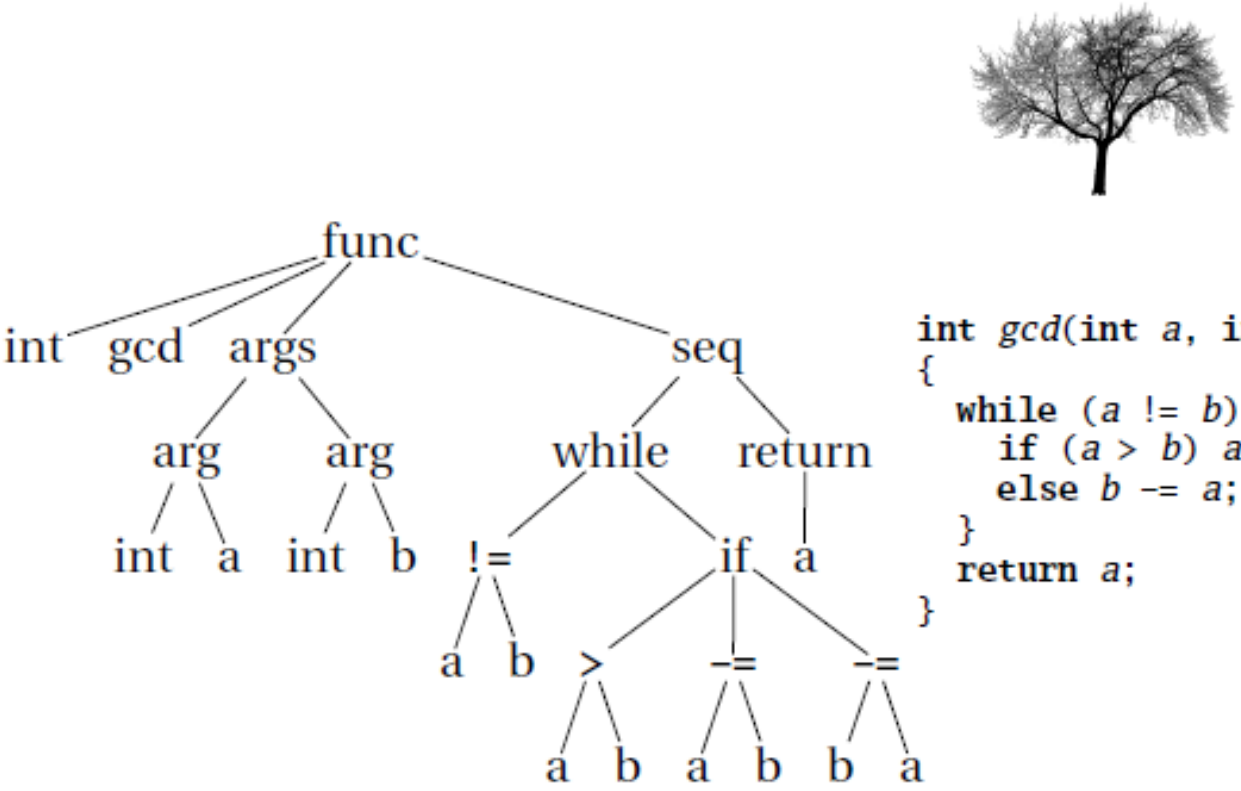
```
int gcd(int a, int b)
{
  while (a != b) {
    if (a > b) a -= b;
    else b -= a;
  }
  return a;
}
```

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

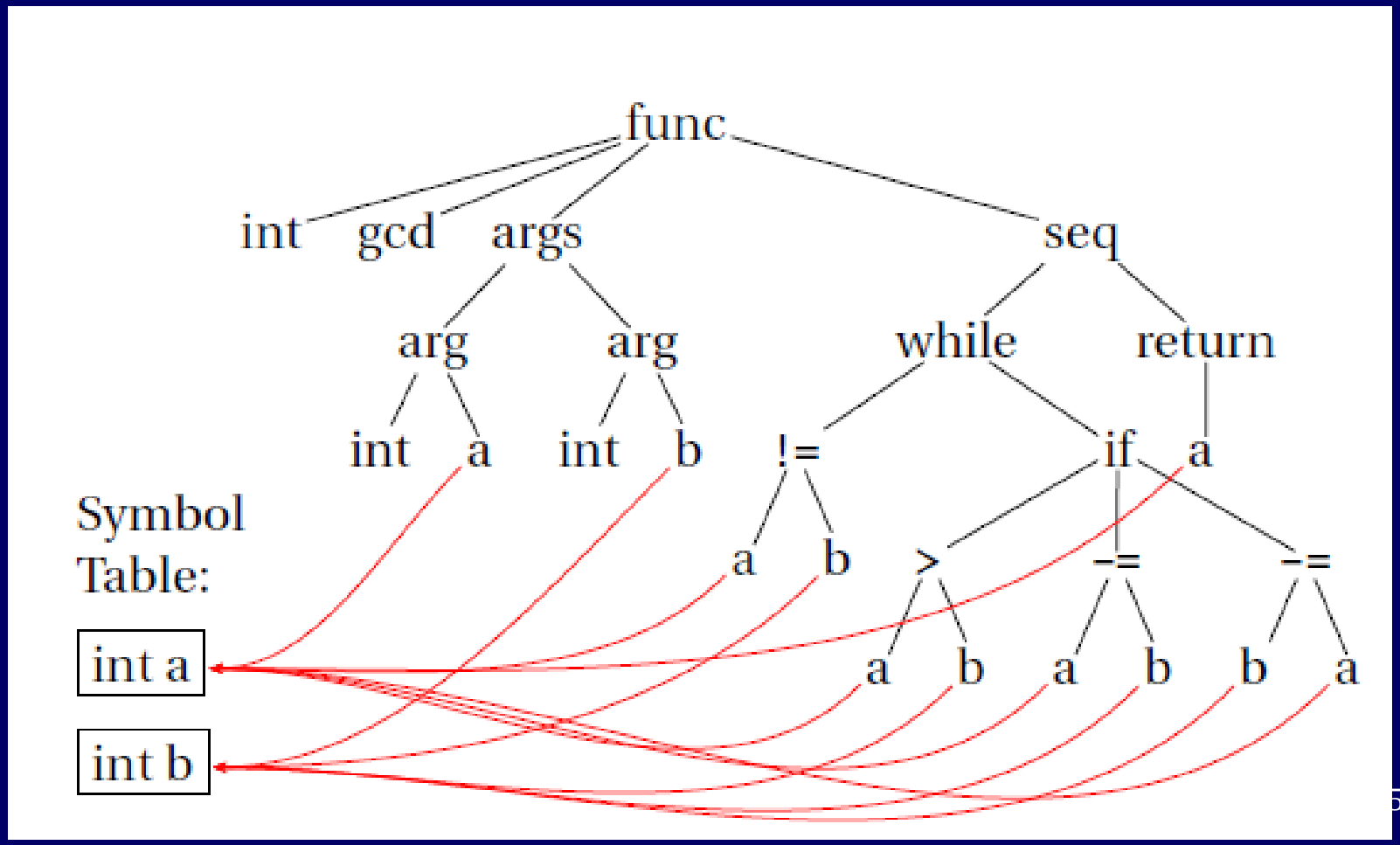
Análise Léxica fornece tokens: uma cadeia de tokens sem espaços e comentários



Análise Sintática fornece uma uma árvore abstrata construída das regras da gramática



Análise Semântica resolve símbolos, com tipos checados

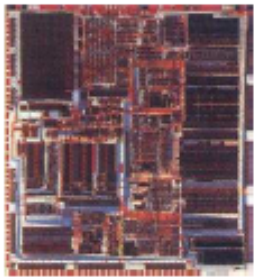


Tradução para uma linguagem intermediária (three-address code: linguagem assembler idealizada com infinitos registradores)

```
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;
}
```

Geração de código para assembler 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

O que é uma linguagens de programação

- Uma linguagem de programação é uma linguagem destinada a 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

Paradigma lógico/declarativo

- Perspectiva da pessoa
- Um programa lógico é equivalente à descrição do problema expressada de maneira formal, similar à maneira que o ser humano raciocinaria sobre ele
- Exemplo de linguagem: PROLOG

Paradigma funcional

- Perspectiva do processo
- A visão funcional resulta num programa que descreve as operações que devem ser efetuadas (processos) para resolver o problema
- Exemplo de linguagem: LISP

Paradigma imperativo/procedimental

- Perspectiva do computador
- Baseado na execução seqüencial de comandos e na manipulação de estruturas de dados
- Exemplos de linguagens: FORTRAN, COBOL, ALGOL 60, APL, BASIC, PL/I, ALGOL 68, PASCAL, C, MODULA 2, ADA

Paradigma orientado a objetos

- Perspectiva do problema
- Modelagem das entidades envolvidas como objetos que se comunicam e sofrem operações
- Exemplos de linguagens: SIMULA 67, SMALLTALK
  - C++, C# e Java: linguagens híbridas (paradigmas imperativo e orientado a objetos),

O que fazer com o resto das linguagens?

Não são Turing-completas, pois não podem simular uma MT, mas são usadas para preparação de documentos

Linguagens de Markup: HTML, SGML, XML

Linguagens de Scripts ou extensão: AWK, Perl, PHP, Python, Ruby, LUA, JavaScript

Linguagens de propósito especial:

YACC para criar parsers

- LEX para criar analisadores léxicos
- MATLAB para computação numérica
- SQL para aplicações com BD

Um pouco de história

- Linguagens que introduziram conceitos importantes e que ainda estão em uso
  - 1955-1965: FORTRAN, COBOL, ALGOL 60, LISP, APL, BASIC (aplicações simples; preocupação com a eficiência)
  - 1965-1971 (com base em ALGOL): PL/I, SIMULA 67, ALGOL 68, PASCAL (pessoas se tornam importantes; preocupação com a inteligibilidade do código (programação estruturada), melhores estruturas de controle)
  - Anos 70 e 80: MODULA 2, ADA, C++, Java (mudança de processos para dados; Abstração, herança e polimorfismo)