

SCC 206

Introdução a Compilação

1. Apresentar conceitos e métodos para as fases de análise e síntese de um compilador
2. Implementar um Front-end e um Back-end para uma linguagem de programação simples (FRANKIE: **Pascal + C simplificados**) via ferramentas para construção de compiladores (**JavaCC**)

Programa e Cronograma

Data	Aula	Tópico	Trabalho	Data Entrega
21/2	<u>1</u>	Apresentação do Curso, da linguagem para o projeto, Conceitos básicos		
28/2	<u>2</u>	Especificações léxicas, sintáticas e semânticas da LF Análise Léxica	P1	
14/3	<u>3</u>	<u>Javacc</u>		
21/3	<u>4</u>	<u>AS Descendente</u>		
28/3	<u>5</u>	<u>AS Descendente</u>	P2	P1
4/4	<u>6</u>	Tratamento de erros sintáticos		
11/4	<u>7</u>	Análise Semântica		
25/4	<u>8</u>	Prova 1		P2
2/5	<u>9</u>	Análise Semântica		
9/5	10	Análise Semântica	P3	
16/5	11	Checagem de Tipos		
23/5	12	Checagem de Tipos		
30/5	13	Ambientes de Execução & Geração Código		
6/6	14	Ambientes de Execução & Geração Código	P4	P3
13/6	15	Ambientes de Execução & Geração Código		
20/6	16	Prova 2		
27/6	17			P4

Bibliografia

- Aho, A. V. et alli - Compilers: Principles, Techniques and Tools. Addison-Wesley Publishing Company, 1986. (Dragãozinho Vermelho em INGLÊS)
- Aho, Lam, Sethi & Ullman. Compilers: Principles, Techniques, and Tools; 2nd Edition, Addison-Wesley Publishing Company, 2006 . (Dragãozinho Roxo em INGLÊS)
- <http://dragonbook.stanford.edu/>
- Aho, A. V. et alli - Compiladores - Princípios, Técnicas e Ferramentas, 2ª Edição, Ed. Pearson, 2007. (Dragãozinho Roxo em PORTUGUÊS)
- Aho, A.V.; Ullman, J.D.; Sethi, R. (1995). Compiladores: Princípios, Técnicas e Ferramentas. Editora LTC. (Dragãozinho Vermelho 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.

CRITÉRIO DE AVALIAÇÃO

- **PROVAS:**

- Haverá 2 provas . Cada prova vale de 0 a 10.
Não há substitutiva.

- **PROJETO:**

- Um compilador que traduz programas da linguagem FRANKIE (Pascal com partes da sintaxe de C) em código de um compilador hipotético à pilha chamado MEPA (Máquina de Execução para Pascal).

-

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
- As características gerais da MEPA e a interpretação das instruções serão vistas durante o curso.
- As instruções da MEPA encontram-se em Kowaltowsky, T. - Implementação de Linguagens de Programação, Guanabara Dois, 1983.

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>

- O projeto será desenvolvido por uma equipe de 3 alunos.
- O projeto é dividido em 4 entregas.
- Cada uma vale de 0 a 10 e suas correções serão baseadas em critérios explícitos definidos juntamente com a especificação da entrega.
- Cada dia de atraso (até 5 dias) tira 1 ponto
- **LISTAS DE EXERCÍCIOS.** Haverá várias listas. Espera-se que os alunos resolvam as listas e tirem as dúvidas com o monitor PAE (Carolina Scarton) ou professor.

Média

- MP = Média Aritmética das Provas
- MT = Média Ponderada das Notas dos Projetos: 10%, 15%, 35%, 40%
-
- MF = Média Final
- Se $MP \geq 5$ então $MF = (6MP + 4MT)/10$
- Senão $MF = (7.5MP + 2.5MT)/10$

Conceitos Básicos

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, isto é, em código de máquina”

Como transforma?

Abrindo a caixa preta

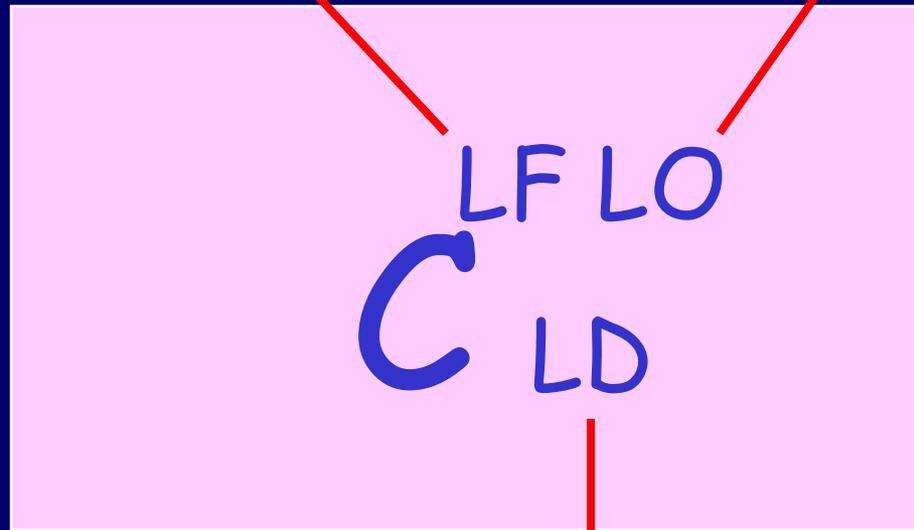
Sistema Híbrido, precisa do Interpretador para MEPA

FRANKIE

com extensões individuais

Linguagem da MEPA

Linguagem fonte de alto nível

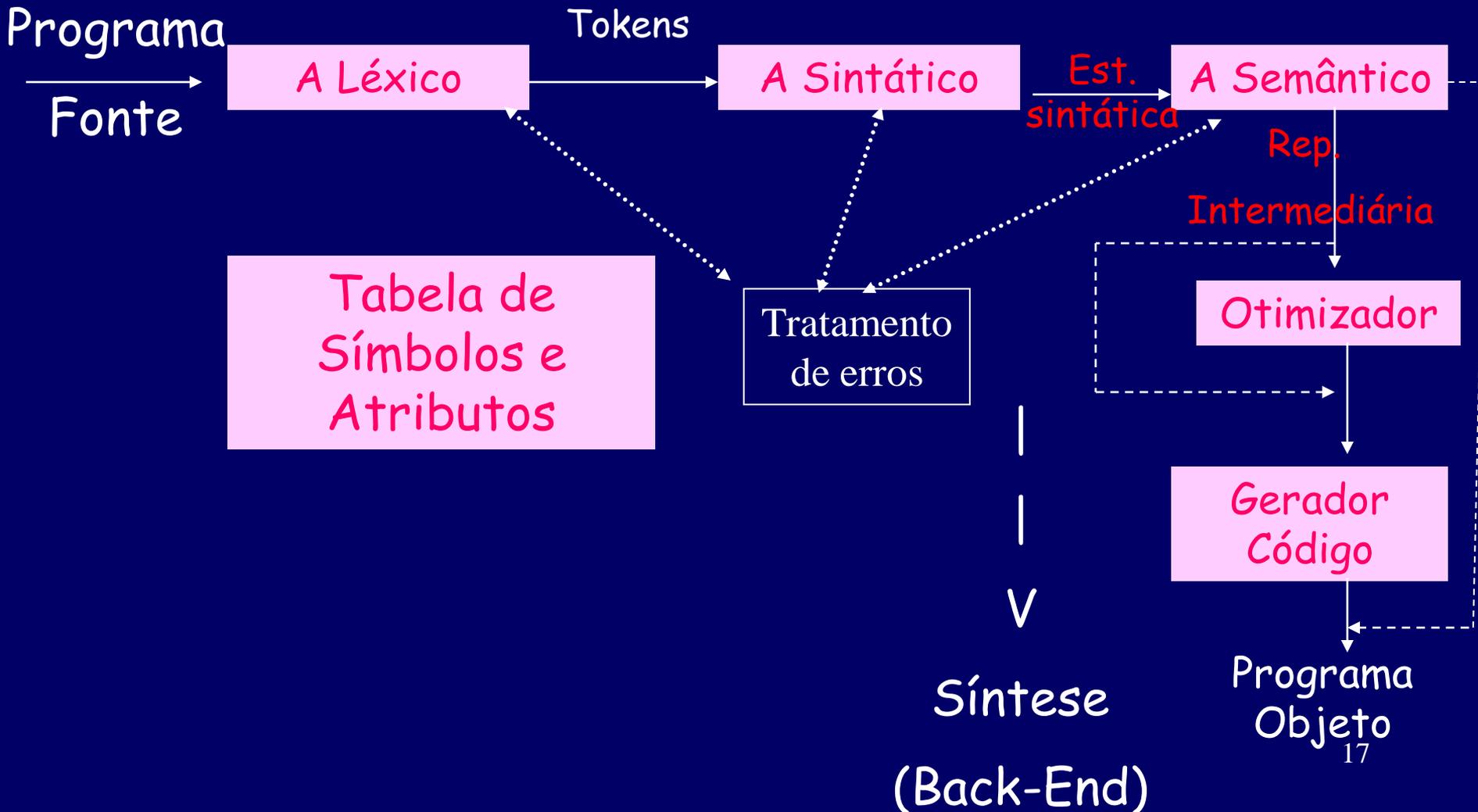


Linguagem Objeto

JAVA com JavaCC

Estrutura de um Compilador

→ Análise (Front-End)



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 parada 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: nível léxico, sintático, semântico/lógico e de referência

Pata sobrevive a bala, geladeira, cirurgia e parada cardíaca



Perky viveu as duas semanas mais duras de sua vida

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);
  m:=0;
end.
```

Erros no programa: nível sintático, semântico, lógico, 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?

Erros sintáticos no texto e no código

- Os erros de concordância nominal ("os noticiario") e de parentização (na expressão do comando While-Do) são da mesma natureza?
- Isto é, precisam do mesmo tipo de recurso para serem reconhecidos?

Erros sintáticos no texto

/* Regras Gramaticais */

1. <sentença> -> <sn> <sv>
2. <sn> -> <artigo> <substantivo>
3. <sv> -> <verbo> <sn>

GLC
para
texto

/* Vocabulário */

4. <artigo> -> o
5. <substantivo> -> gato | rato
6. <verbo> -> comeu

O gato comeu o rato.

O rato comeu o gato.

Erros sintáticos no código

```
<expressão> ::=
    <expressão simples> [<relação> <expressão simples>]
<relação> ::=
    = | <> | < | <= | >= | >
<expressão simples> ::=
    [+ | -] <termo> {(+ | - | or) <termo>}
<termo> ::=
    <fator> {(* | div | and) <fator>}
<fator> ::=
    <variavel>
    | <número>
    | ( <expressão> )
    | not <fator>
```

GLC para
código

Adicionando artigos e substantivos no plural

/* Regras Gramaticais */

<sentença> -> <sn> <sv>

<sn> -> <artigo> <substantivo>

<sv> -> <verbo> <sn>

/* Vocabulário */

<artigo> -> o | os

<substantivo> -> gato | rato

<substantivo> -> gatos | ratos

<verbo> -> comeu

O gato comeu o rato. O rato comeu o gato.

Os gato comeu o rato. Os rato comeu o gato.

O gato comeu os ratos. O rato comeu os gatos.

Os gato comeu os rato. Os rato comeu os gato.

/* Regras Gramaticais */

<sentença> -> <sn> <sv>

<sn> -> <artigos> <substantivos>

<sn2s> -> <artigos> <substantivos>

<sn2p> -> <artigop> <substantivop>

<sv> -> <verbo> <sn2s>

<sv> -> <verbo> <sn2p>

/* Vocabulário */

<artigos> -> o

<artigop> -> os

<substantivos> -> gato | rato

<substantivop> -> gatos | ratos

<verbo> -> comeu

O gato comeu o rato. O rato comeu o gato.

O gato comeu os ratos. O rato comeu os gatos.

Em uma GLC
temos que
duplicar as
regras para
adicionar o
tratamento
da
concordância
nominal

DCG (Gramática de Cláusulas Definidas)

- Definite Clause Grammars (DCGs) são uma notação muito conveniente para representar uma gramática em várias aplicações (rodam sobre Prolog):
 - tanto para trabalhar com uma língua natural como o Português
 - como uma linguagem de programação como Pascal.

/* Gramatica na notacao DCG, considerando-se:

- G = genero,
- N = numero,
- sn = sintagma nominal
- sv = sintagma verbal
- Frase = sentença/oração

*/

<http://www.icmc.sc.usp.br/~sandra/6/gramatica.htm>

```
/* Regras gramaticais */
```

```
sn(G, N) --> substantivo(G, N).
```

```
sn(G, N) --> substantivo_proprio(G, N).
```

```
sn(G, N) --> artigo_definido(G, N), substantivo(G, N).
```

```
sn(G, N) --> artigo_definido(G, N), substantivo(G, N), adjetivo(G, N).
```

```
sn(G, N) --> artigo_indefinido(G, N), substantivo(G, N).
```

```
sn(G, N) --> artigo_indefinido(G, N), substantivo(G, N), adjetivo(G, N).
```

```
sv(G, N) --> verbo(N).
```

```
sv(G, N) --> verbo(N), sn(G, N).
```

```
/* Sentencas */
```

```
frase --> sn(G, N), sv(G, N).
```

```
frase --> sn(G, N), sv(G, N), adjetivo(G, N).
```

```
% para interrogar: frase(X, []). ou seja, quais são as possíveis frases X
```

- /* Vocabulario */
- adjetivo(masc, sing) --> [bonito].
adjetivo(fem, sing) --> [bonita].
adjetivo(masc, pl) --> [bonitos].
adjetivo(fem, pl) --> [bonitas].
adjetivo(_) --> [fragil].
- verbo(sing) --> [eh].
- verbo(sing) --> [ama].
- verbo(pl) --> [sao].
- verbo(pl) --> [amam].

```
artigo_definido(masc, sing) --> [o].
artigo_definido(fem, sing) --> [a].
artigo_definido(masc, pl) --> [os].
artigo_definido(fem, pl) --> [as].
artigo_indefinido(masc, sing) --> [um].
artigo_indefinido(fem, sing) --> [uma]. artigo_indefinido(masc, pl) --> [uns].
artigo_indefinido(fem, pl) --> [umas].
```

```
substantivo(fem, sing) --> [Nome], {pertence(Nome, [mulher, pessoa, mae,
brasileira])}.
substantivo(masc, sing) --> [Nome], {pertence(Nome, [homem, pai, brasileiro])}.
substantivo(fem, pl) --> [Nome], {pertence(Nome, [mulheres, pessoas, maes,
brasileiras])}.
substantivo(masc, pl) --> [Nome], {pertence(Nome, [homens, pais, brasileiros])}.
substantivo_proprio(fem, sing) --> [Nome], {pertence(Nome, [maria, ana])}.
substantivo_proprio(masc, sing) --> [Nome], {pertence(Nome, [joao, zeca])}.
```

```
pertence(X, [X|_]).          % se for o primeiro elemento
pertence(X, [_|Cauda]):-
    pertence(X, Cauda).
```

Usando DCG como reconhecedor

- execução para verificar se uma frase está gramaticamente correta ou não
-
- ?- frase([joao, eh, um, homem], []).
- yes.
- ?- frase([um, eh, homem], []).
- no.

Usando DCG como gerador (mais natural, pois é gramática)

- execução para formação de sentenças - possíveis soluções

?- frase(X, []).

X = [mulher,eh,pessoa] ;

X = [mulher,eh,uma,pessoa] ;

X = [homem,eh,um,brasileiro,bonito] ;

X = [homens,sao,brasileiros] ;

X = [o,homem,eh,joao] ;

X = [uma,mae,eh,uma,pessoa] .

yes.

Porém,...

- Embora parecendo similares na forma, uma *GLC* e uma *GCD* tem poderes diferentes, isto é, não reconhecem o mesmo conjunto de linguagens.
- O poder de uma *GCD* é muito maior do que o de uma *GLC*, pois por traz existe o Prolog → assim tem o poder máximo de uma gramática.
- Vejam detalhes deste contraste *GCD* e *GLC* em:
<http://www.coli.uni-saarland.de/~kris/learn-prolog-now/html/node54.html>

Compilador Ilustrado

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

Compilando um programa simples

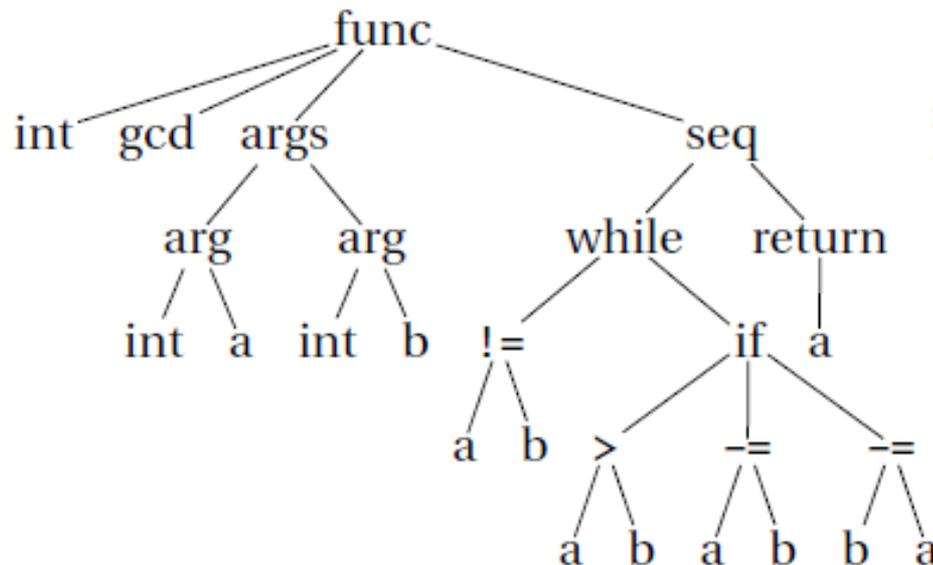
```
int gcd(int a, int b)
{
    while (a != b) {
        if (a > b) a -= b;
        else b -= a;
    }
    return a;
}
```

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;
}
```

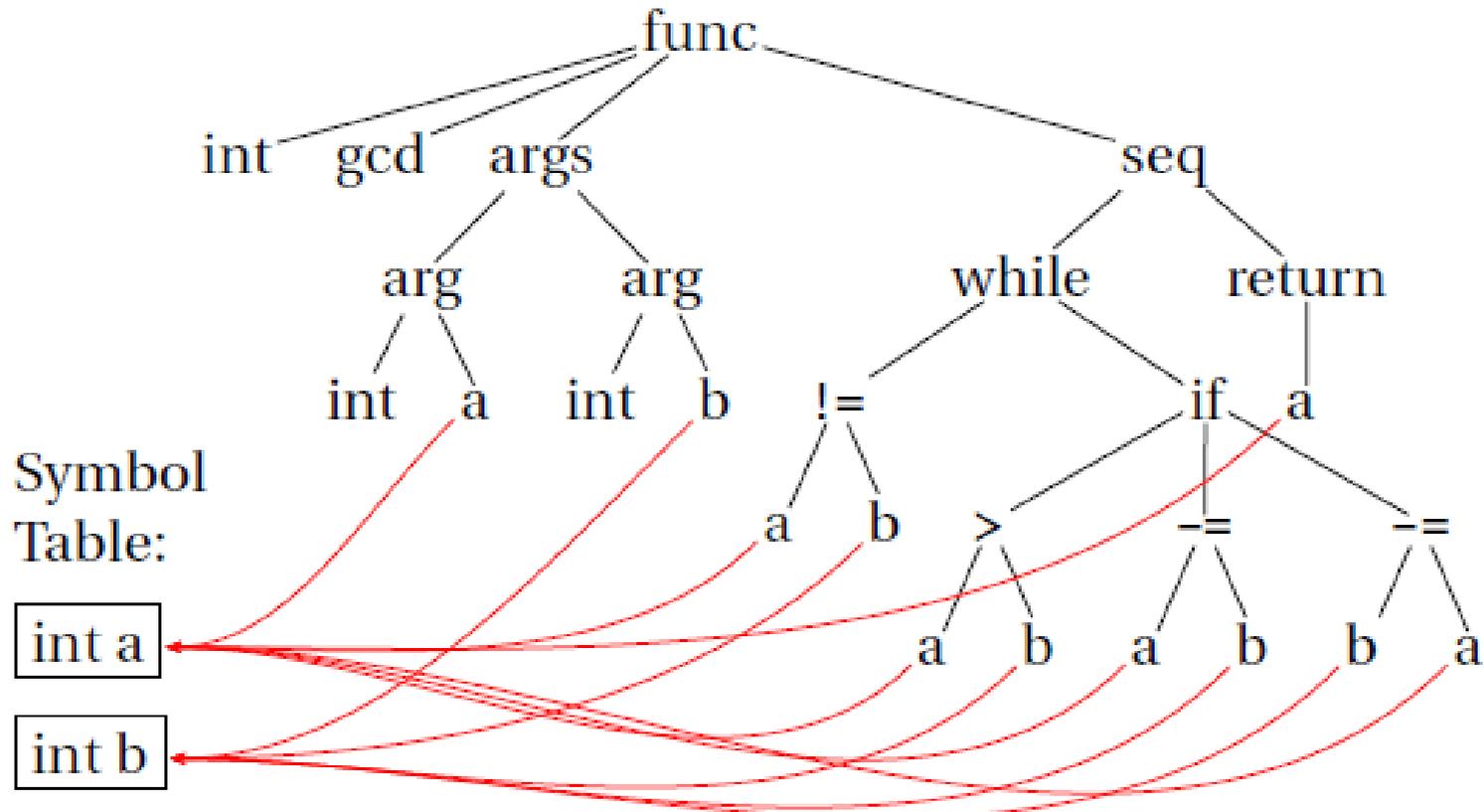
```
i n t s p g c d ( i n t s p a , s p i
n t s p b ) n l { n l s p s p w h i l e s p
( a s p ! = s p b ) s p { n l s p s p s p s p i
f s p ( a s p > s p b ) s p a s p - = s p b
; n l s p s p s p s p e l s e s p b s p - = s p
a ; n l s p s p } n l s p s p r e t u r n s p
a ; n l } n l
```


Análise Sintática fornece uma árvore abstrata construída das regras da gramática



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

Análise Semântica resolve símbolos, com tipos checados



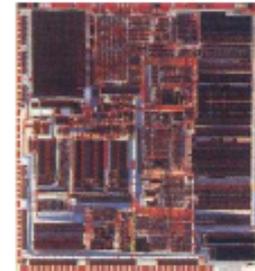
Tradução para uma linguagem intermediária (three-address code: linguagem assembler idealizada com infinitos registradores)

```
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 linguagem 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),

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

O que fazer com o resto das linguagens?

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)

Assembly Language

Before: numbers

After: Symbols

55	gcd: pushl %ebp
89E5	movl %esp, %ebp
8B4508	movl 8(%ebp), %eax
8B550C	movl 12(%ebp), %edx
39D0	cmpl %edx, %eax
740D	je .L9
39D0	.L7: cmpl %edx, %eax
7E08	jle .L5
29D0	subl %edx, %eax
39D0	.L2: cmpl %edx, %eax
75F6	jne .L7
C9	.L9: leave
C3	ret
29C2	.L5: subl %eax, %edx
EBF6	jmp .L2

FORTRAN

Before

```
gcd: pushl %ebp
      movl  %esp, %ebp
      movl  8(%ebp), %eax
      movl  12(%ebp), %edx
      cmpl  %edx, %eax
      je    .L9
.L7:  cmpl  %edx, %eax
      jle  .L5
      subl %edx, %eax
.L2:  cmpl  %edx, %eax
      jne  .L7
.L9:  leave
      ret
.L5:  subl  %eax, %edx
      jmp  .L2
```

After: Expressions, control-flow

```
10   if (a .EQ. b) goto 20
      if (a .LT. b) then
          a = a - b
      else
          b = b - a
      endif
      goto 10
20   end
```

COBOL

Added type declarations, record types, file manipulation

```
data division.  
file section.  
*   describe the input file  
fd  employee-file-in  
      label records standard  
      block contains 5 records  
      record contains 31 characters  
      data record is employee-record-in.  
01  employee-record-in.  
    02  employee-name-in   pic x(20).  
    02  employee-rate-in  pic 9(3)v99.  
    02  employee-hours-in pic 9(3)v99.  
    02  line-feed-in      pic x(1).
```



From cafepress.com

LISP, Scheme, Common LISP

Functional, high-level languages

```
(defun gnome-doc-insert ()
  "Add a documentation header to the cu
Only C/C++ function types are properly
  (interactive)
  (let (c-insert-here (point))
    (save-excursion
      (beginning-of-defun)
      (let (c-arglist
            c-funcname
            (c-point (point))
            c-comment-point
            c-isvoid
            c-doinstert)
        (search-backward "(")
        (forward-line -2)
        (while (or (looking-at "^$")
                     (looking-at "^ *}")
                     (looking-at "^ \\*")
                     (looking-at "^#"))
          (forward-line 1))
```

APL

Powerful operators, interactive language, custom character set

```
[0] Z←GAUSSRAND N;B;F;M;P;Q;R
[1] ⍠Returns  $\omega$  random numbers having a Gaussian normal distribution
[2] ⍠ (with mean 0 and variance 1) Uses the Box-Muller method.
[3] ⍠ See Numerical Recipes in C, pg. 289.
[4] ⍠
[5] Z←10
[6] M←1+2*31 ⍠ largest integer
[7] L1:Q←N-ρZ ⍠ how many more we need
[8] →(Q≤0)/L2 ⍠ quit if none
[9] Q←⌈1.3×Q÷2 ⍠ approx num points needed
[10] P←1+(2÷M-1)×1+?(Q,2)ρM ⍠ random points in -1 to 1 square
[11] R←+/P×P ⍠ distance from origin squared
[12] B←(R≠0)∧R<1
[13] R←B/R ⍠ P←B×P ⍠ points within unit circle
[14] F←(1-2×(ρR)÷R)★.5
[15] Z←Z,.,P×F,.[1.5]F
[16] →L1
[17] L2:Z←N+Z
[18] ⍠ ArchDate: 12/16/1997 16:20:23.170
```

Source: Jim Weigang, <http://www.chilton.com/~jimw/gstrand.html>

At right: Datamedia APL Keyboard



Algol, Pascal, Clu, Modula, Ada

Imperative, block-structured language, formal syntax definition, structured programming

```
PROC insert = (INT e, REF TREE t)VOID:
  # NB inserts in t as a side effect #
  IF TREE(t) IS NIL THEN t := HEAP NODE := (e, TREE(NIL), TREE(NIL))
  ELIF e < e OF t THEN insert(e, l OF t)
  ELIF e > e OF t THEN insert(e, r OF t)
  FI;

PROC trav = (INT switch, TREE t, SCANNER continue, alternative)VOID:
  # traverse the root node and right sub-tree of t only. #
  IF t IS NIL THEN continue(switch, alternative)
  ELIF e OF t <= switch THEN
    print(e OF t);
    traverse( switch, r OF t, continue, alternative)
  ELSE # e OF t > switch #
    PROC defer = (INT sw, SCANNER alt)VOID:
      trav(sw, t, continue, alt);
    alternative(e OF t, defer)
  FI;
```

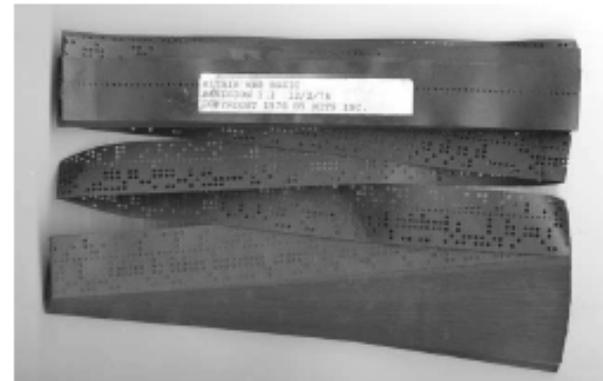
Algol-68, source <http://www.csse.monash.edu.au/~lloyd/tildeProgLang/Algol68/treemerge.a68>

BASIC

Programming for the masses

```
10 PRINT "GUESS A NUMBER BETWEEN ONE AND TEN"  
20 INPUT A$  
30 IF A$ <> "5" THEN GOTO 60  
40 PRINT "GOOD JOB, YOU GUESSED IT"  
50 GOTO 100  
60 PRINT "YOU ARE WRONG. TRY AGAIN"  
70 GOTO 10  
100 END
```

Started the whole Bill Gates/
Microsoft thing. BASIC was
invented by Dartmouth
researchers John George Kemeny
and Thomas Eugene Kurtz.



Hoje, qual seria uma linguagem para
as massas?

Simula, Smalltalk, C++, Java, C#

The object-oriented philosophy

```
class Shape(x, y); integer x; integer y;
virtual: procedure draw;
begin
    comment -- get the x & y coordinates --;
    integer procedure getX;
        getX := x;
    integer procedure getY;
        getY := y;

    comment -- set the x & y coordinates --;
    integer procedure setX(newx); integer newx;
        x := newx;
    integer procedure setY(newy); integer newy;
        y := newy;
end Shape;
```

C

Efficiency for systems programming

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

ML, Miranda, Haskell

Functional languages with a syntax

```
structure RevStack = struct
  type 'a stack = 'a list
  exception Empty
  val empty = []
  fun isEmpty (s:'a stack):bool =
    (case s
     of [] => true
      | _ => false)
  fun top (s:'a stack): =
    (case s
     of [] => raise Empty
      | x::xs => x)
  fun pop (s:'a stack):'a stack =
    (case s
     of [] => raise Empty
      | x::xs => xs)
  fun push (s:'a stack,x: 'a):'a stack = x::s
  fun rev (s:'a stack):'a stack = rev (s)
end
```

sh, awk, perl, tcl, python, php

Scripting languages: glue for binding the universe together

```
class() {
  classname='echo "$1" | sed -n '1 s/ *:.*/p' '
  parent='echo "$1" | sed -n '1 s/^.*: */p' '
  hppbody='echo "$1" | sed -n '2,$p' '

  forwarddefs="$forwarddefs
class $classname;"

  if (echo $hppbody | grep -q "$classname()"); then
    defaultconstructor=
  else
    defaultconstructor="$classname() {}"
  fi
}
```

SQL

Database queries

```
CREATE TABLE shirt (  
    id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,  
    style ENUM('t-shirt', 'polo', 'dress') NOT NULL,  
    color ENUM('red', 'blue', 'white', 'black') NOT NULL,  
    owner SMALLINT UNSIGNED NOT NULL  
        REFERENCES person(id),  
    PRIMARY KEY (id)  
);
```

```
INSERT INTO shirt VALUES  
(NULL, 'polo', 'blue', LAST_INSERT_ID()),  
(NULL, 'dress', 'white', LAST_INSERT_ID()),  
(NULL, 't-shirt', 'blue', LAST_INSERT_ID());
```

Prolog

Logic Language

edge(a, b).

edge(b, c).

edge(c, d).

edge(d, e).

edge(b, e).

edge(d, f).

path(X, X).

path(X, Y) :- edge(X, Z), path(Z, Y).

Sintaxe e semântica

- A descrição de uma linguagem de programação 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

Sintaxe

- As gramáticas de linguagens de programação são utilizadas para **produzir** ou **reconhecer** cadeias?

Sintaxe

- Descrição de linguagens de programação por meio de gramáticas livres de contexto
- A maioria das linguagens não são livres de contexto, mas sensíveis ao contexto
 - Por exemplo, variável deve ser declarada antes de ser usada
- Métodos para reconhecer gramáticas sensíveis ao contexto são complexos. Na prática, especifica-se uma gramática livre de contexto para a linguagem de programação e trata-se a sensibilidade ao contexto de maneira informal
 - Tabela de símbolos

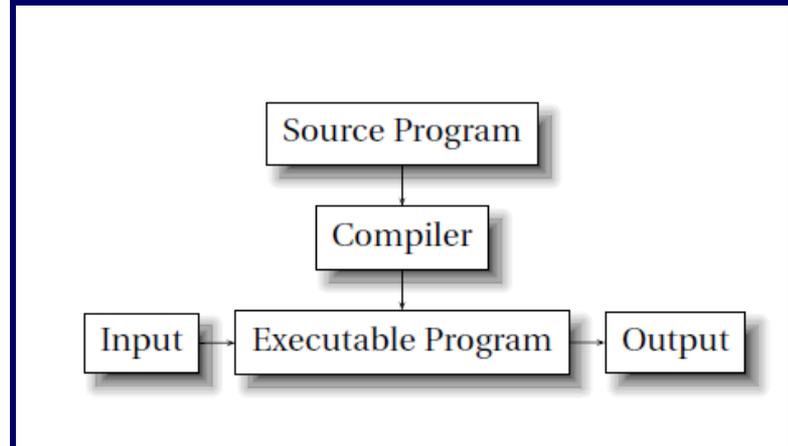
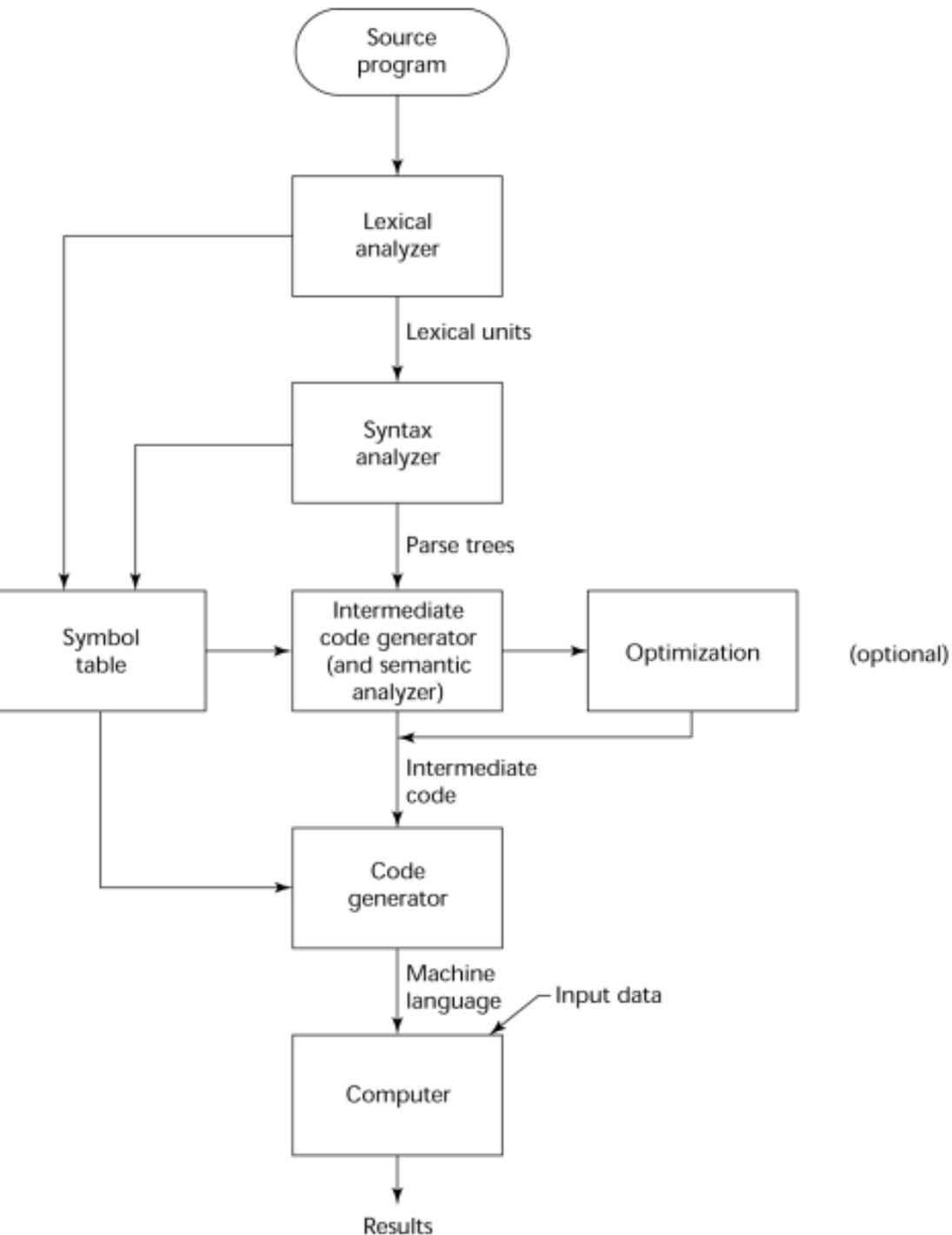
Gramáticas e reconhecedores

Gramáticas	Reconhecedores
Irrestrita	Máquina de Turing
Sensível ao contexto	Máquina de Turing com memória limitada
Livre de contexto	Autômato a pilha
Regular	Autômato finito

Métodos de Implementação

- Compilação
 - Programas são traduzidos em linguagem de máquina
- Interpretação Pura
 - Programas são interpretados por outro programa (interpretador)
- Sistemas Híbridos
 - Oferecem um compromisso entre compiladores e interpretadores puros

Compilador

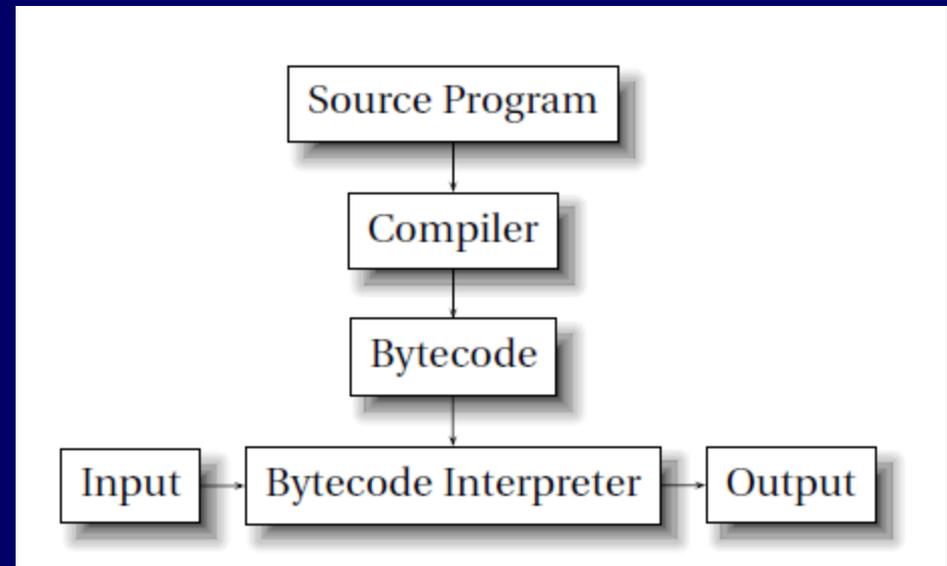
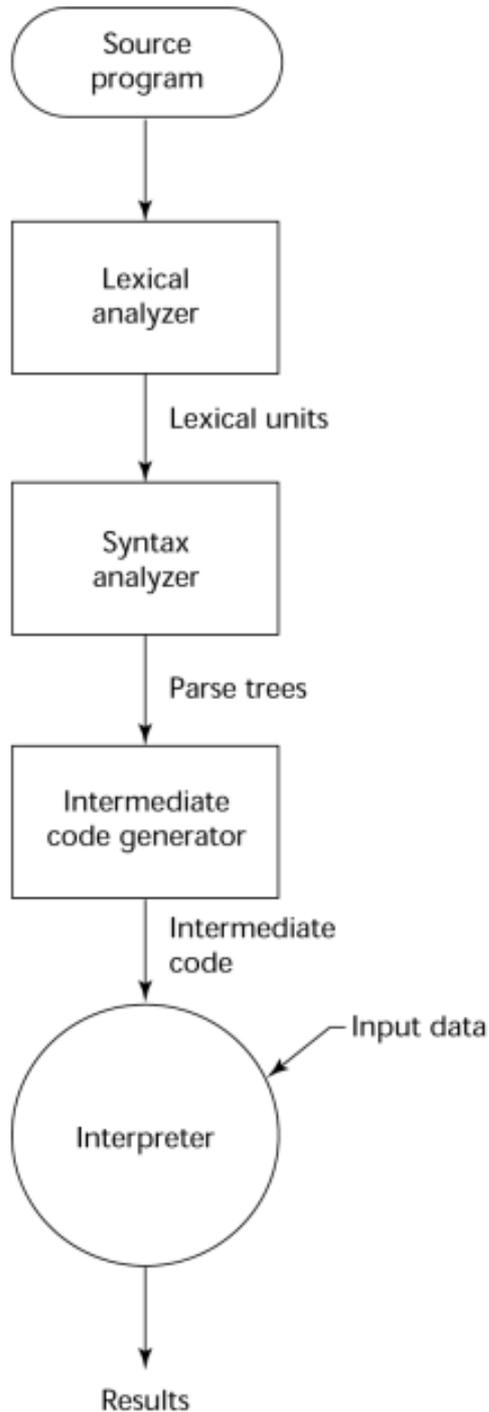


Tradução lenta
Execução rápida

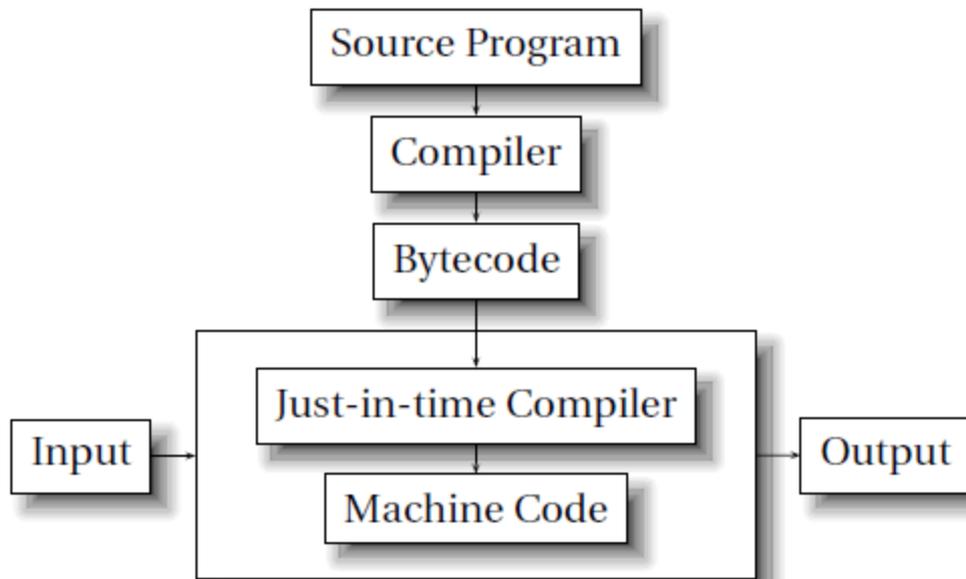
Sistemas Híbridos

- Há a tradução para uma linguagem intermediária para facilitar a interpretação
- Mais rápido que interpretação pura
- Exemplos:
 - Programas Perl são compilados parcialmente para detectar erros antes da interpretação.
 - Implementações iniciais de Java foram híbridas; o código intermediário, *byte code*, fornece portabilidade para qualquer máquina que tenha um interpretador de byte code e um ambiente de execução (juntos, são chamados de *Java Virtual Machine*)

Interpretador de Bytecodes



O conceito não é novo...Pascal P-code difundiu a linguagem Pascal



Compilador Just in Time

Une bytecodes com
compilação dinâmica

- Traduz inicialmente para uma linguagem intermediária (bytecode)
- Então compila a linguagem intermediária dos subprogramas em código de máquina nativa quando eles são chamados (tempo de execução) para executar na máquina nativa
- Este código é mantido para chamadas subsequentes
- Sistemas JIT são muito usados para Java

TIOBE Programming Community Index for February 2011

Position Feb 2011	Position Feb 2010	Delta in Position	Programming Language	Ratings Feb 2011	Delta Feb 2010	Status
1	1	=	Java	18.482%	+1.13%	A
2	2	=	C	14.988%	-1.62%	A
3	4	↑	C++	8.187%	-1.26%	A
4	7	↑↑↑	Python	7.038%	+2.72%	A
5	3	↓↓	PHP	6.973%	-3.03%	A
6	6	=	C#	6.809%	+1.79%	A
7	5	↓↓	(Visual) Basic	4.924%	-2.13%	A
8	12	↑↑↑↑	Objective-C	2.571%	+0.79%	A
9	10	↑	JavaScript	2.558%	-0.08%	A
10	8	↓↓	Perl	1.907%	-1.69%	A
11	11	=	Ruby	1.615%	-0.82%	A
12	-	=	Assembly*	1.269%	-	A-
13	9	↓↓↓↓	Delphi	1.060%	-1.60%	A
14	19	↑↑↑↑↑	Lisp	0.956%	+0.39%	A
15	37	↑↑↑↑↑↑↑↑	NXT-G	0.849%	+0.58%	A--
16	30	↑↑↑↑↑↑↑↑	Ada	0.805%	+0.44%	A--
17	17	=	Pascal	0.735%	+0.13%	A
18	21	↑↑↑	Lua	0.714%	+0.21%	A--
19	13	↓↓↓↓↓	Go	0.707%	-1.07%	A--
20	32	↑↑↑↑↑↑↑↑	RPG (OS/400)	0.628%	+0.27%	A--

Atualizado mensalmente.

Usa máquinas de busca(Google, MSN, Yahoo!, Wikipedia e YouTube) para calcular o número mundial de usuários, cursos e vendedores de linguagens pagas.

Não se refere à melhor linguagem nem àquela para qual muitas linhas foram escritas.

Tiobe Programming Community Index

