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 (Pascal Simplificado) via ferramentas para construção de compiladores (JavaCC)

1

Programa e Cronograma

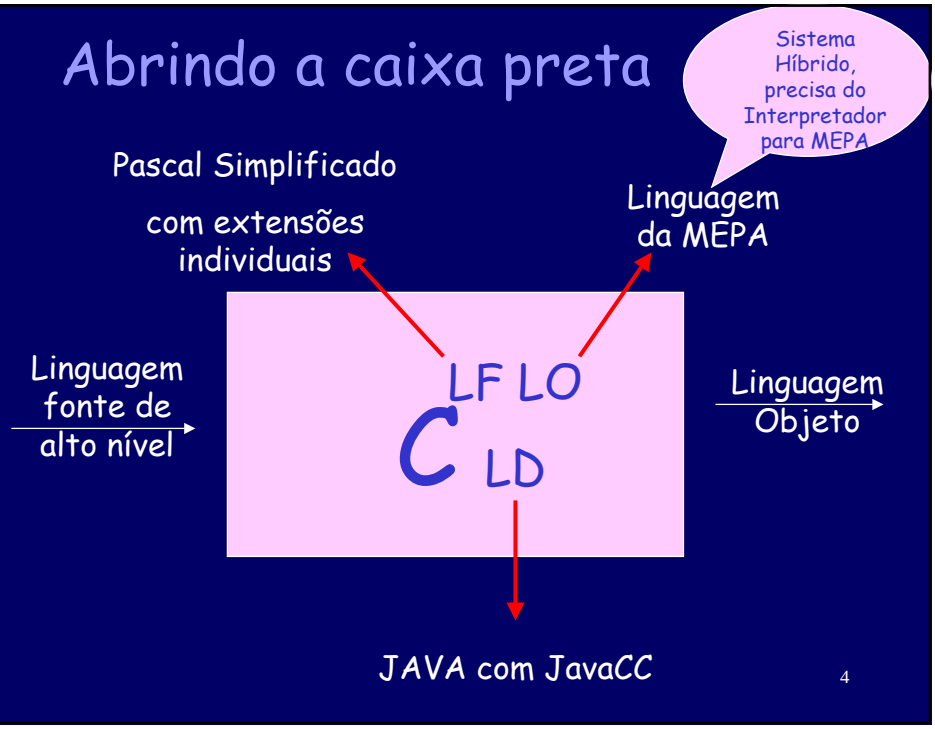
Data	Aula	Tópico	Trabalho	Data Entrega
23/2	1	Apresentação do Curso, da linguagem para o projeto, Conceitos básicos		
2/3	2	Especificações Análise léxica	P1	
9/3	3	AS Descendente		P1
16/3	4	AS Descendente		
23/3	5	Javacc	P2	
30/3	**Não há AULA**	**Semana Santa**		
6/4	**Não há AULA**	**Conferência**		
13/4	6	AS Ascendente		
20/4	7	AS Ascendente		P2
27/4	**Não há AULA**	**Conferência**		
4/5	8	ASem **PROVA 1 – das 17:20 – 19:00 h**		
11/5	9	ASem		
18/5	**Não há AULA**	**Conferência**		
25/5	10	ASem	P3	
1/6	11	Checagem de Tipos		
8/6	12	Ambientes de Execução & Geração Código		P3
15/6	13	Ambientes de Execução & Geração Código	P4	
22/6	14	Ambientes de Execução & Geração Código		
29/6	15	**PROVA 2 –** **Das 16:20 – 19:00 h**		
6/7	16			P4

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"

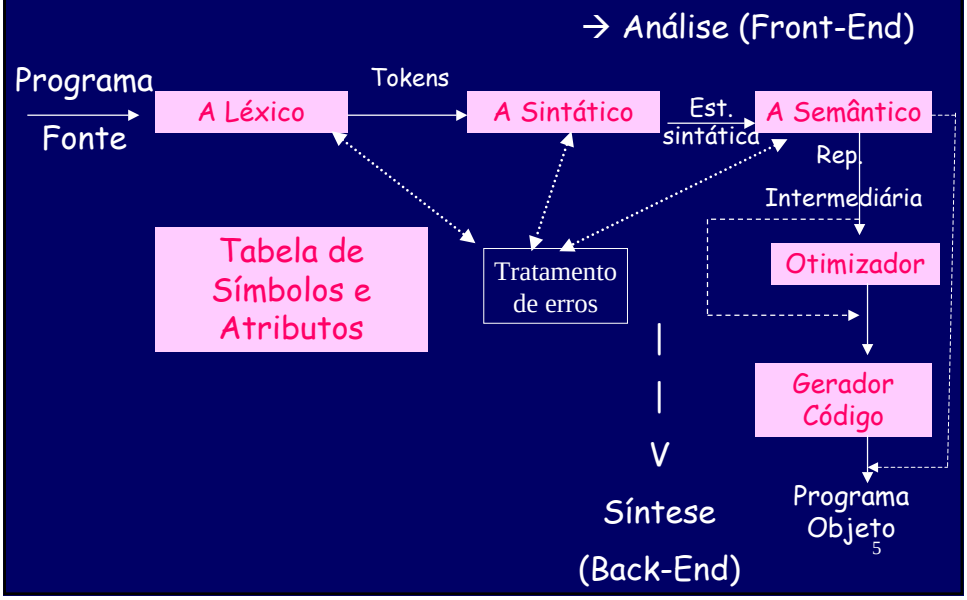
3

Abrindo a caixa preta



4

Estrutura de um Compilador



5

Compilador tem responsabilidade de reportar erros!

6

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: nível léxico, sintático, semântico/lógico 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);
    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?

11

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?

12

Erros sintáticos no texto

/* Regras Gramaticais */
1. <sentença> -> <sn> <sv>
2. <sn> -> <artigo> <substantivo>
3. <sv> -> <verbo> <sn>

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

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

GLC para texto

13

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

14

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.

15

/* 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

16

DCG (Gramática de Cláusulas Definidas)

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

17

/* 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

18

```
/* 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
```

19

- /* 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].

20

```
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).
```

21

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.

22

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.
```

23

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

24

Compilando um programa simples

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

25

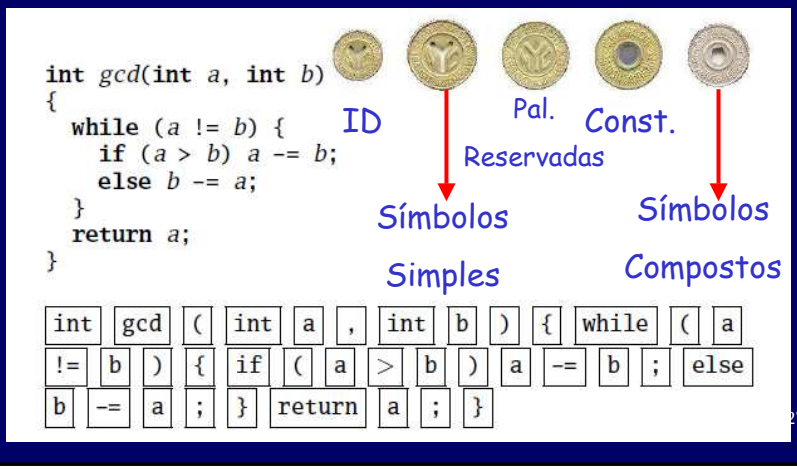
O que o compilador vê: o texto é uma sequência de caracteres

```
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
```

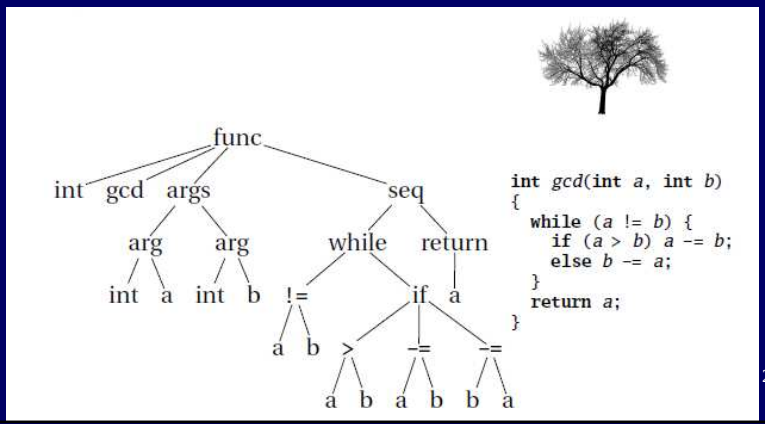
26

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



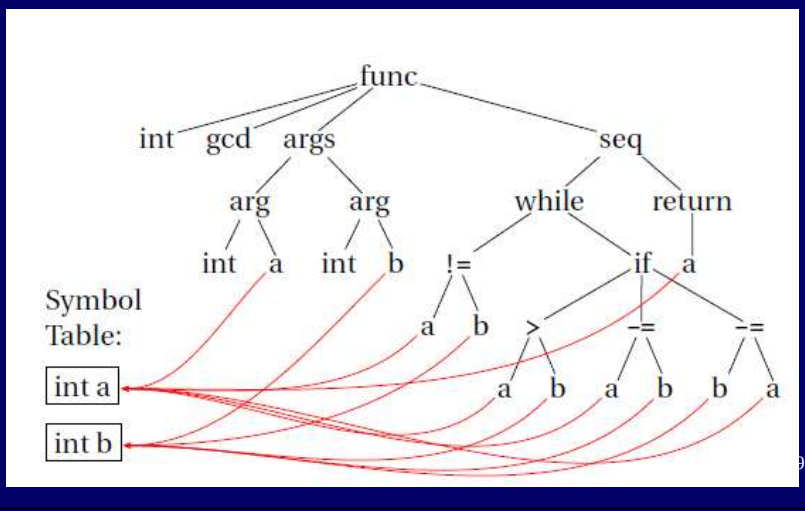
27

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



28

Análise Semântica resolve símbolos, com tipos checados

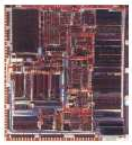


Tradução para uma linguagem intermediária (three-address code: linguagem assembler idealizada com infinitos registradores)

```
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
```

31

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."

32

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

33

Paradigma lógico/declarativo

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

34

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

35

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

36

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),

37

O que fazer com o resto das linguagens?

38

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, melhores estruturas de controle)
  - Anos 70 e 80: MODULA 2, ADA, C++, Java (mudança de processos para dados; Abstração, herança e polimorfismo)

39

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

40

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
```

41

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

42

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-doinsert)
        (search-backward "(")
        (forward-line -2)
        (while (or (looking-at "^$")
                   (looking-at "^ *}")
                   (looking-at "^ \\*")
                   (looking-at "^#"))
          (forward-line 1))
```

43

APL

Powerful operators, interactive language, custom character set

```
[0]   Z←GAUSSRAND N;B;F;M;P;Q;R
[1]   ⍝Returns ⍵ 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←⍬
[6]   M←¯1+2*31          ⍝ largest integer
[7]  L1:Q←N-⍴Z           ⍝ 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)⍴M  ⍝ 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←(¯2×(⍟R)÷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/gsrand.html

At right: Datamedia APL Keyboard

44

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

45

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.

46

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;
```

47

C

Efficiency for systems programming

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

48

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
```

49

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
}
```

50

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());
```

51

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).
```

52

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

53

Sintaxe

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

54

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

55

Gramáticas e reconhecedores

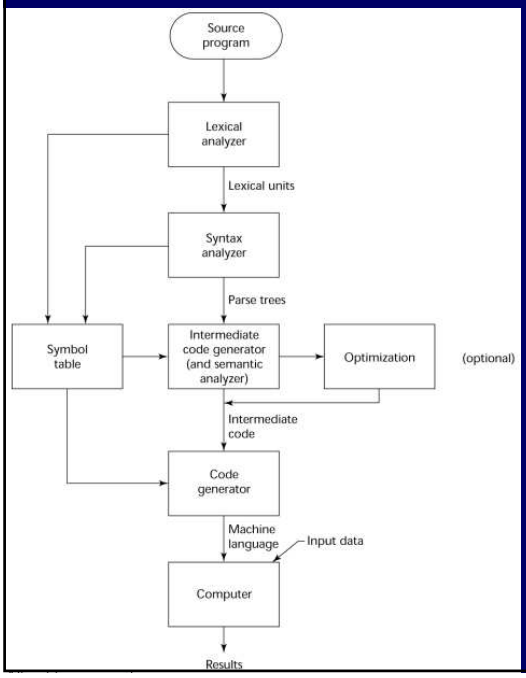
Gramáticas	Reconhecedores
Irrestrita	Máquina de Turing
Sensível ao contexto	Máquina de Turing com memória limitada
Livre de contexto	Autômato a pilha
Regular	Autômato finito

56

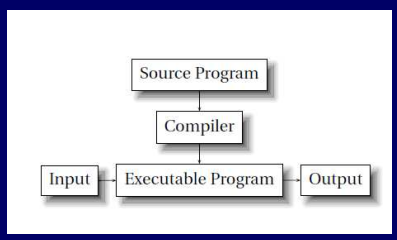
Métodos de Implementação

- 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

57



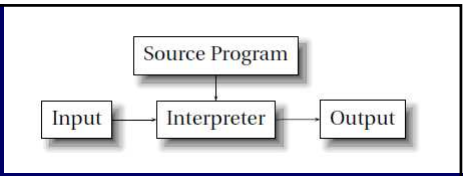
Compilador



Tradução lenta
Execução rápida

58

Interpretador



- Não há tradução
- Execução lenta (10 a 100 vezes mais lento que programas compilados)
- Frequentemente requer mais espaço
- Atualmente raro para as linguagens tradicionais, mas está havendo um retorno com as linguagens de script para a Web (por exemplo, JavaScript, PHP)

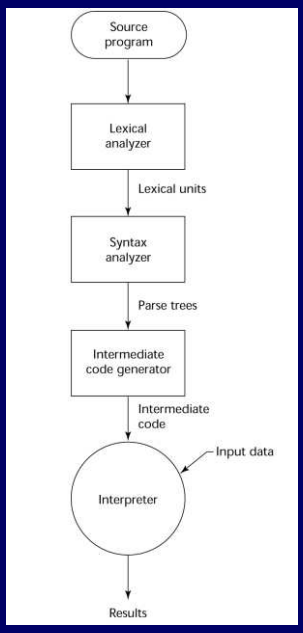
59

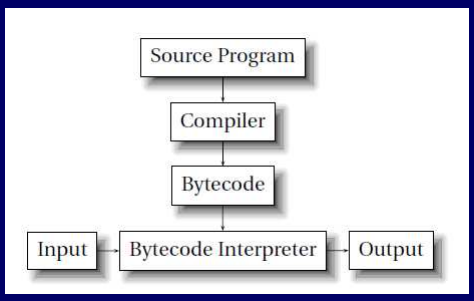
Sistemas Híbridos

- 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.
  - Impementações 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*)

60

Interpretador de Bytecodes

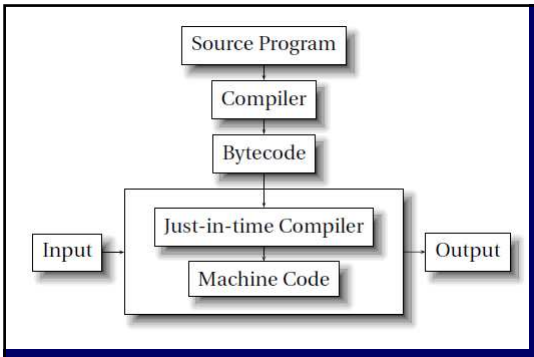




O conceito não é novo...Pascal P-code difundiu a linguagem Pascal

61

Compilador Just in Time



- Traduz inicialmente para uma linguagem intermediária
- Então compila a linguagem intermediária dos subprogramas em código de máquina quando eles são chamados
- Este código é mantido para chamadas subsequentes
- Sistemas JIT são muito usados para Java
- Linguagens .NET são implementadas com um sistema JIT

62

TIOBE Programming Community Index for February 2010

Position Feb 2010	Position Feb 2009	Delta in Position	Programming Language
1	1	=	Java
2	2	=	C
3	5	↑↑	PHP
4	3	↓	C++
5	4	↓	(Visual) Basic
6	6	=	C#
7	7	=	Python
8	8	=	Perl
9	9	=	Delphi
10	10	=	JavaScript
11	11	=	Ruby
12	32	↑↑↑↑↑↑↑↑↑↑	Objective-C
13	-	↑↑↑↑↑↑↑↑↑↑	Go
14	14	=	SAS
15	13	↓↓	PL/SQL
16	17	↑	ABAP
17	16	↓	Pascal
18	18	=	ActionScript
19	23	↑↑↑↑	Lisp/Scheme
20	24	↑↑↑↑	MATLAB

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.

63