

Compilação e Paradigmas de Linguagem de Programação

Prof. André Lage

lage(a)laccan.ufal.br

<https://sites.google.com/a/ic.ufal.br/andrelage/>

Maceió, 19-21/03/18

LaCCAN

Universidade Federal de Alagoas

IMPORTANTE: Apresentações são tópicos, estude pelo material indicado pelo professor.

DISCLAIMER: Fontes didáticas principais:

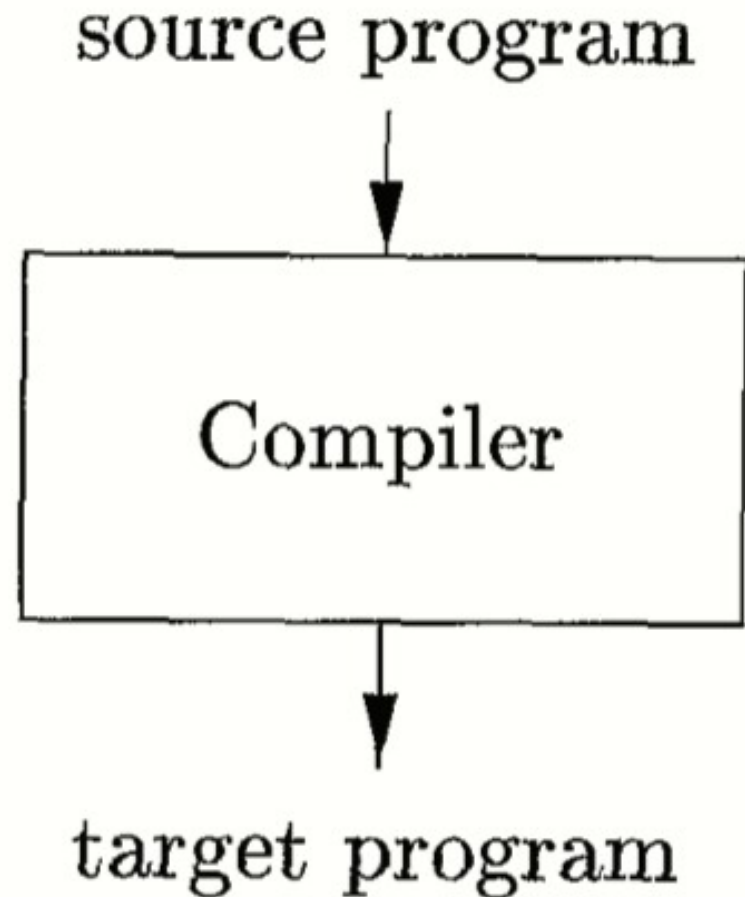
*Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman. **Compilers: Principles, Techniques, and Tools** (2nd Edition). Addison-Wesley Longman Publishing. 2006. (capítulo 1)*

Contexto

- Antes das linguagens de alto nível → programação em Assembly
- Até 1975 → SOs e assemblers em assembly por restrições da memória e ineficiência dos compiladores
- Surgimento das linguagens de alto nível → facilidade
 - Mais próximo ao ser humano
 - Código menor com referências alfabéticas
 - Fontes compilados para diferentes arquiteturas → interoperabilidade em nível de código-fonte

Compilador

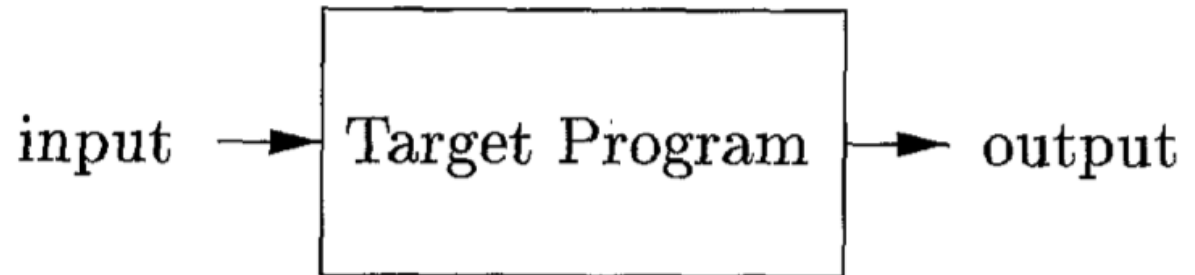
- Conversão
 - Linguagem de alto nível → linguagem de máquina
- Source program
- Target program
 - **Machine-language program**



Processadores de linguagens

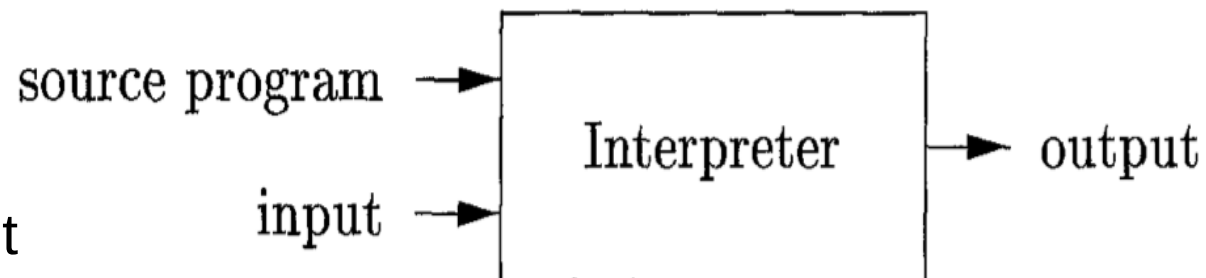
- **Compilador**

- Gera um **programa** que executará as operações
- Mais rápido
- Exemplos
 - C, C++, Fortran



- **Interpretador**

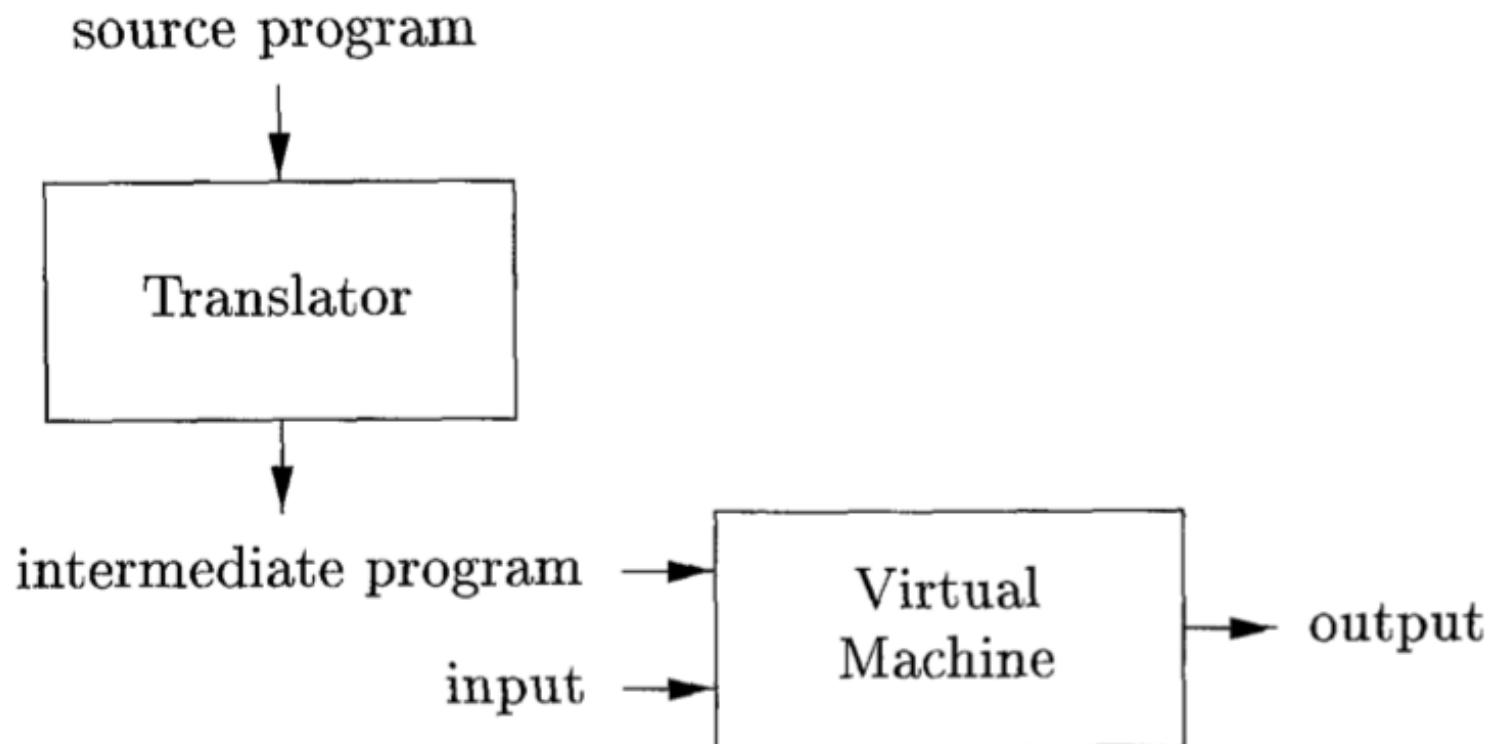
- Executa **diretamente** as operações
- Melhor depuração
- Exemplos
 - Bash, Perl, JavaScript



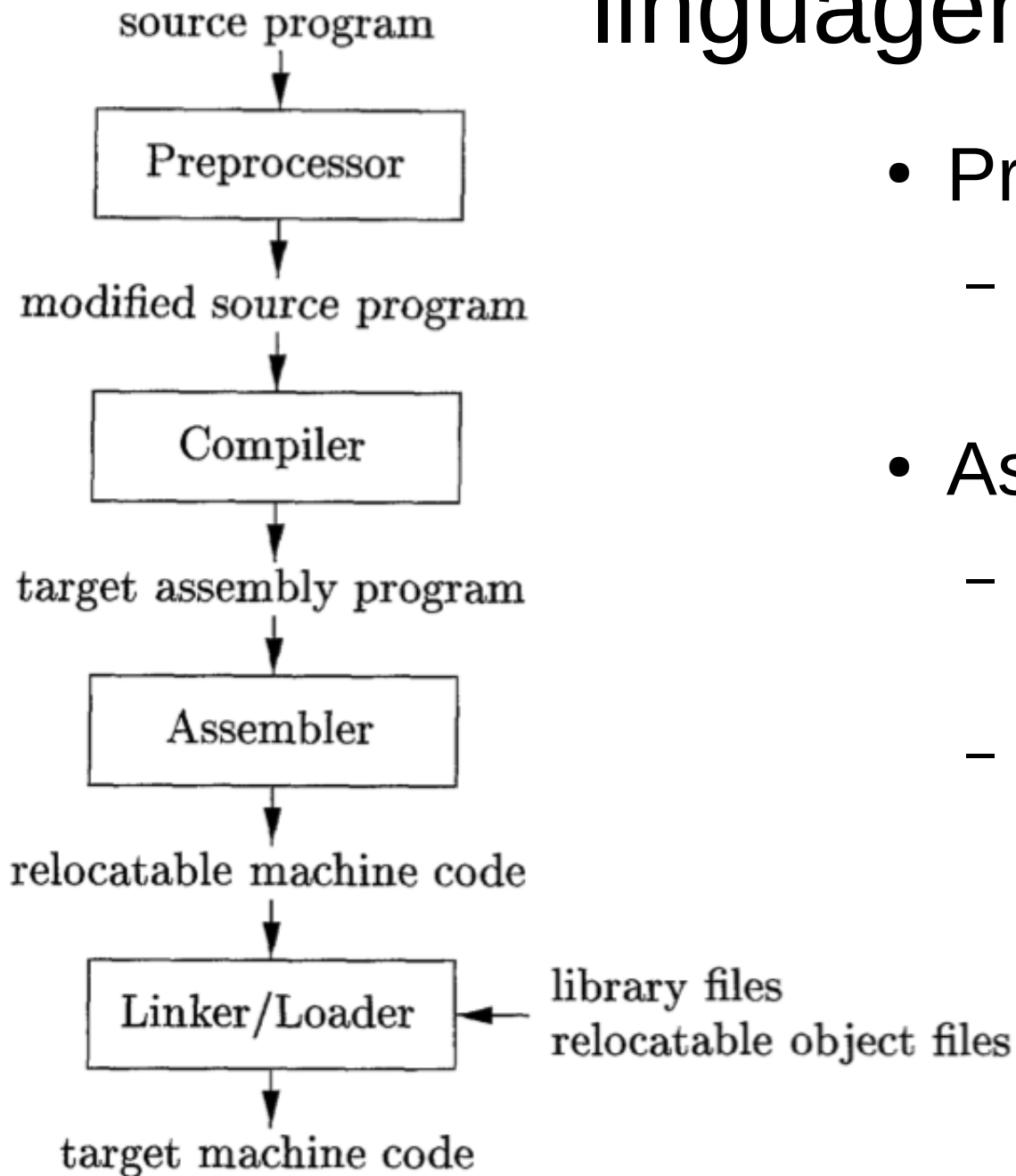
Processadores de linguagens

- **Híbrido**

- Compilação + Interpretação
- Exemplo: **Java** Virtual Machine
- *Just In-time (JIT) compilation*



Sistema de processamento de linguagens



- Pré-processador
 - Traduz macros para a linguagem fonte
- Assembler
 - Código assembly (intermediário)
 - Facilita a conversão e a depuração

Estrutura de um compilador

- **Análise**

- Analisa a estrutura gramatical do programa fonte
 - Representação intermediária (ex.: árvore sintática + tabela de símbolos)
- Indica o tipo e onde ocorreram erros
- Análise **Léxica**
- Análise **Sintática**
- Análise **Semântica**

- **Síntese**

- Constrói o programa alvo (binário)
- Utiliza a saída da Análise

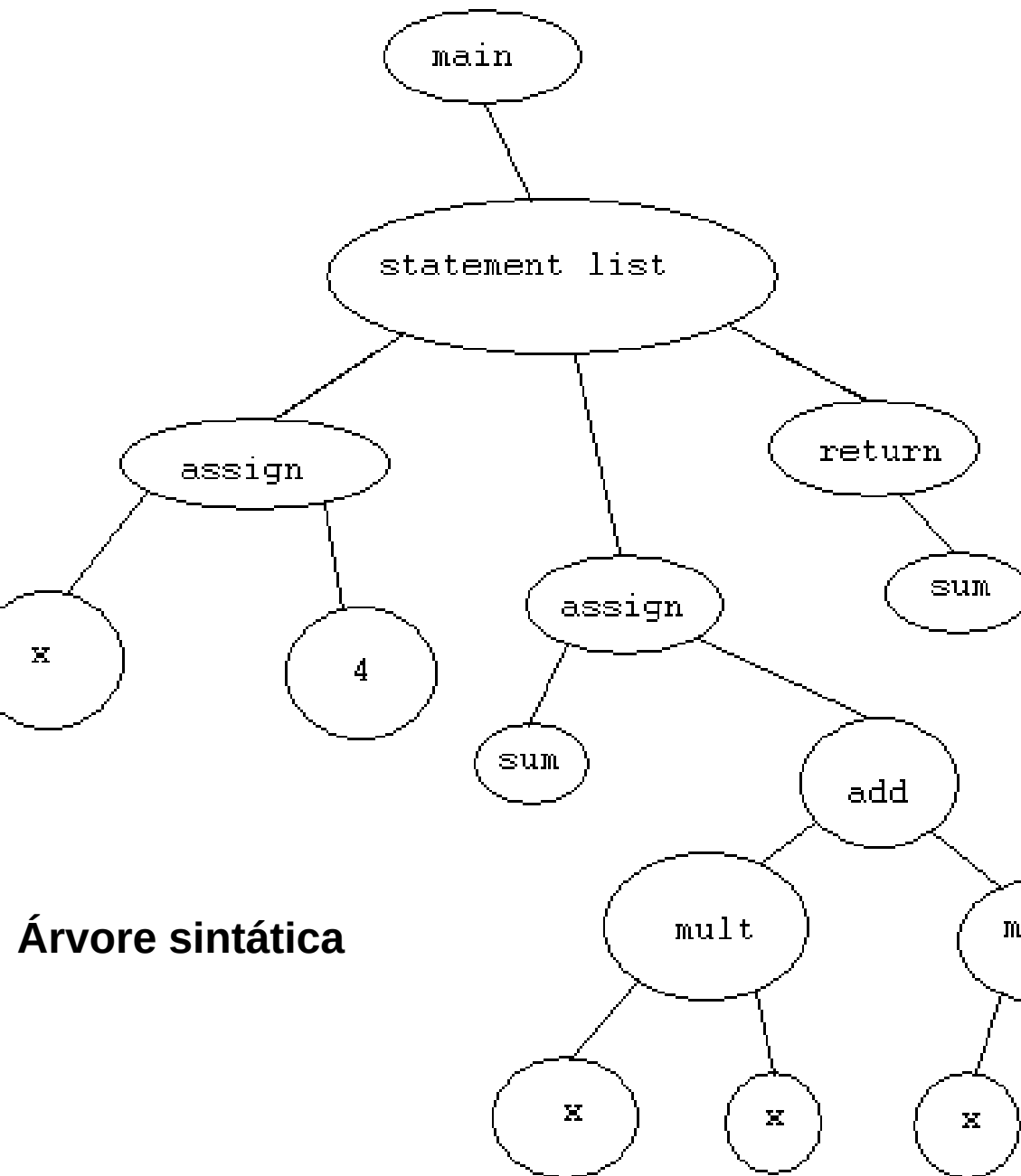
Estrutura de um compilador

Exemplo

```
1 int main()  
2 {  
3     int x, sum;  
4     x = 4;  
5     sum = x * x + x * x;  
6     return sum;  
7 }
```


Exemplo

```
1 int main()  
2 {  
3     int x, sum;  
4     x = 4;  
5     sum = x * x + x * x;  
6     return sum;  
7 }
```



x	local_0
sum	local_1

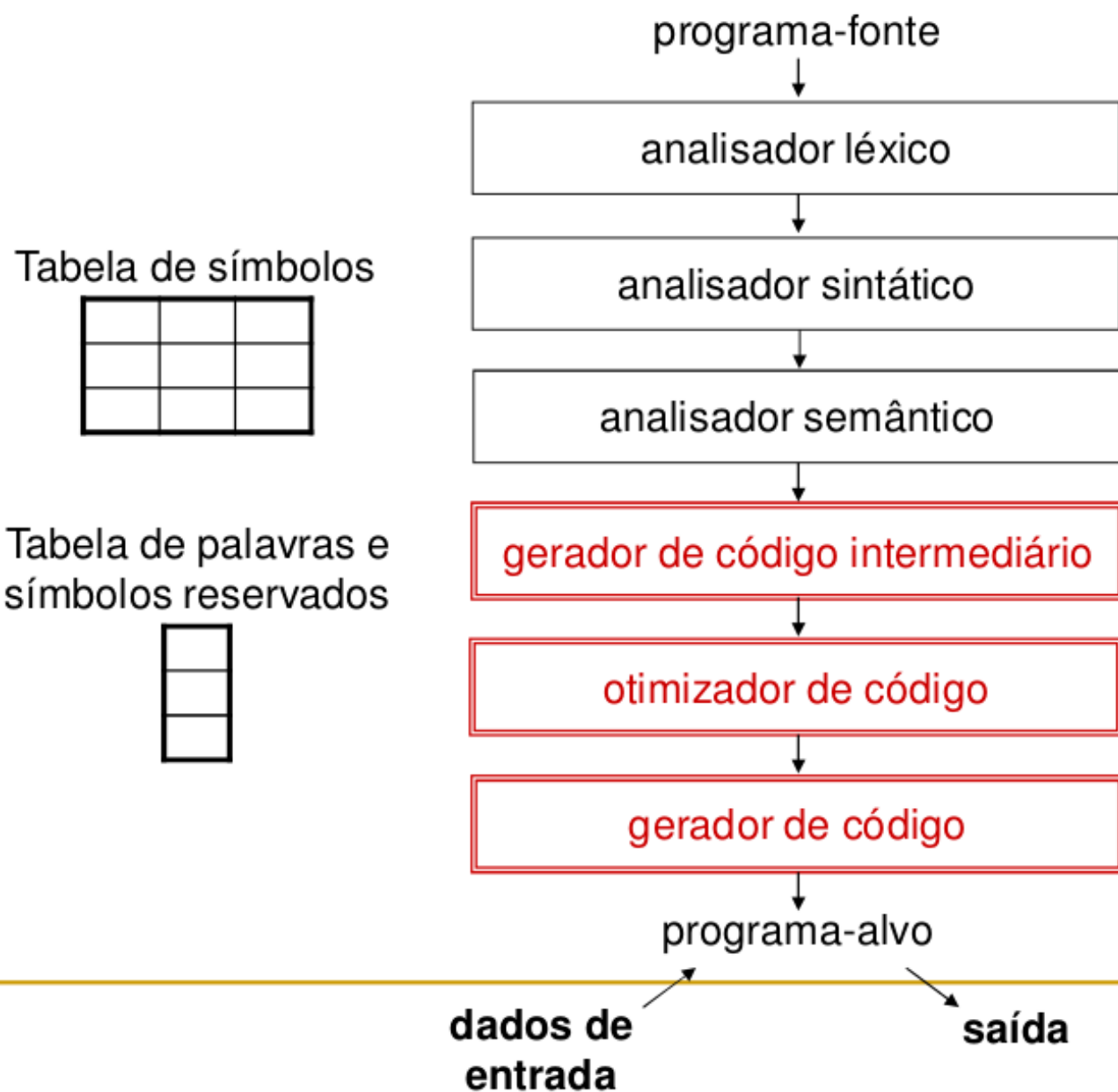
Tabela de símbolos

Árvore sintática

Fonte:
<http://www.quora.com/Why-should-I-learn-about-compilers>

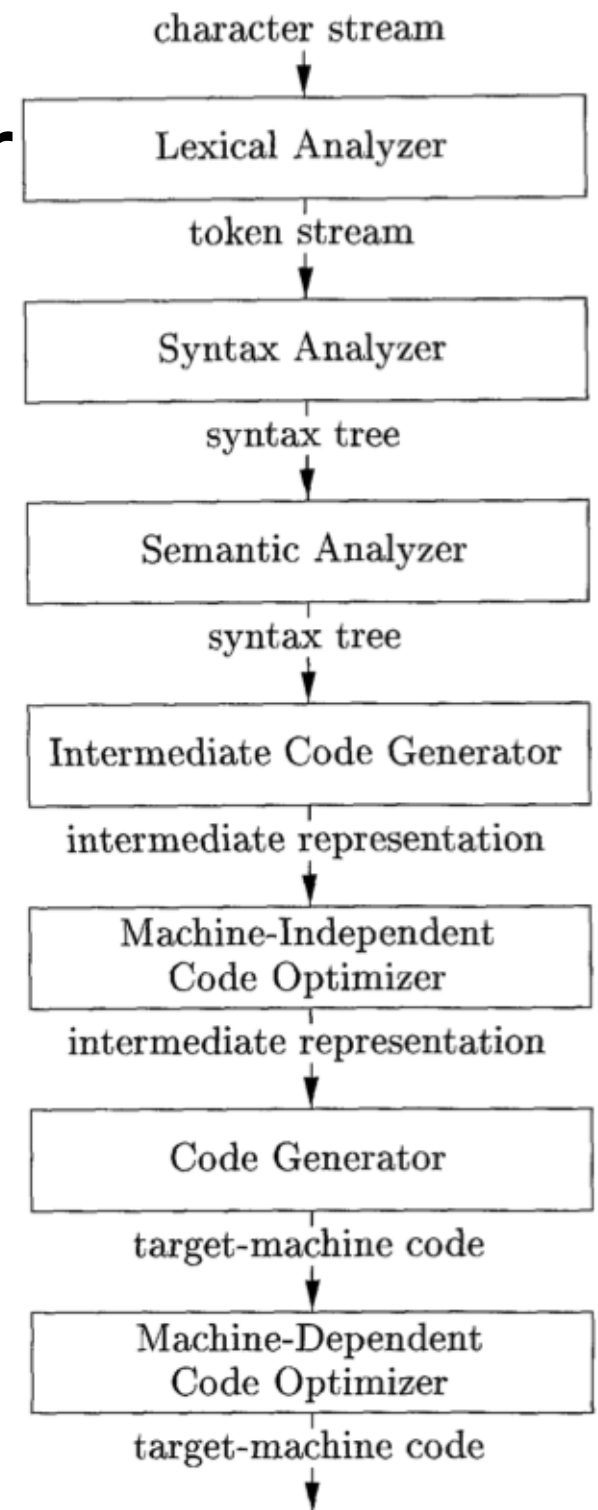
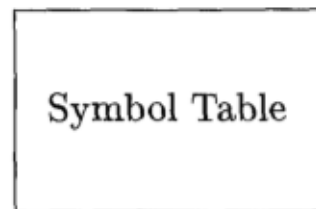
Estrutura de um compilador

- **Otimização de código**
 - Opcional
 - Desempenho
 - Independente da arquitetura
 - Arquitetura específica



Estrutura de um compilador


- **Otimização de código**
 - Opcional
 - Desempenho
 - Independente da arquitetura
 - Arquitetura específica



Análise Léxica – *Scanning*

- *Separa os símbolos léxicos (ou tokens)*
- *Verifica se os símbolos léxicos fazem parte do alfabeto da linguagem*
- *Forma: $\langle token_name, attribute_value \rangle$*
 - *token_name*: símbolo abstrato
 - *attribute_value*: aponta para uma entrada da tabela de símbolos para esse token
- Exemplo:

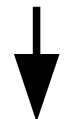
```
position = initial + rate * 60
```

 **Análise Léxica**

$\langle id, 1 \rangle \langle = \rangle \langle id, 2 \rangle \langle + \rangle \langle id, 3 \rangle \langle * \rangle \langle 60 \rangle$

Análise Sintática - *Parsing*

- Extrai a estrutura gramatical a partir dos tokens (saída da Análise Léxica)
- Representação em forma de árvore
 - **Árvore Sintática**
- Verifica se uma sequência dos símbolos formam um programa válido
 - Obedece a **gramática** da linguagem
- Exemplo (continua)
position = initial + rate * 60



Análise Léxica

$\langle \text{id}, 1 \rangle \langle = \rangle \langle \text{id}, 2 \rangle \langle + \rangle \langle \text{id}, 3 \rangle \langle * \rangle \langle 60 \rangle$

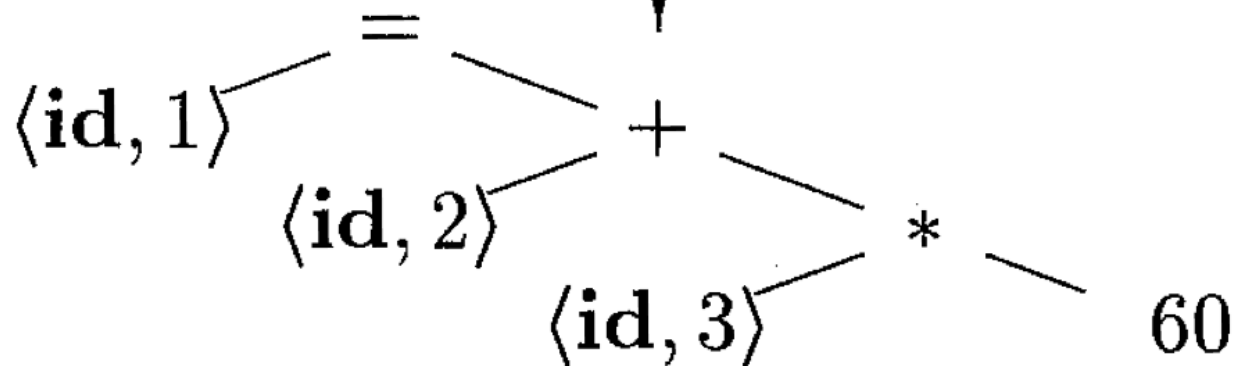
Análise Sintática - Exemplo

`position = initial + rate * 60`

Lexical Analyzer

$\langle \text{id}, 1 \rangle \langle = \rangle \langle \text{id}, 2 \rangle \langle + \rangle \langle \text{id}, 3 \rangle \langle * \rangle \langle 60 \rangle$

Syntax Analyzer



Análise Semântica

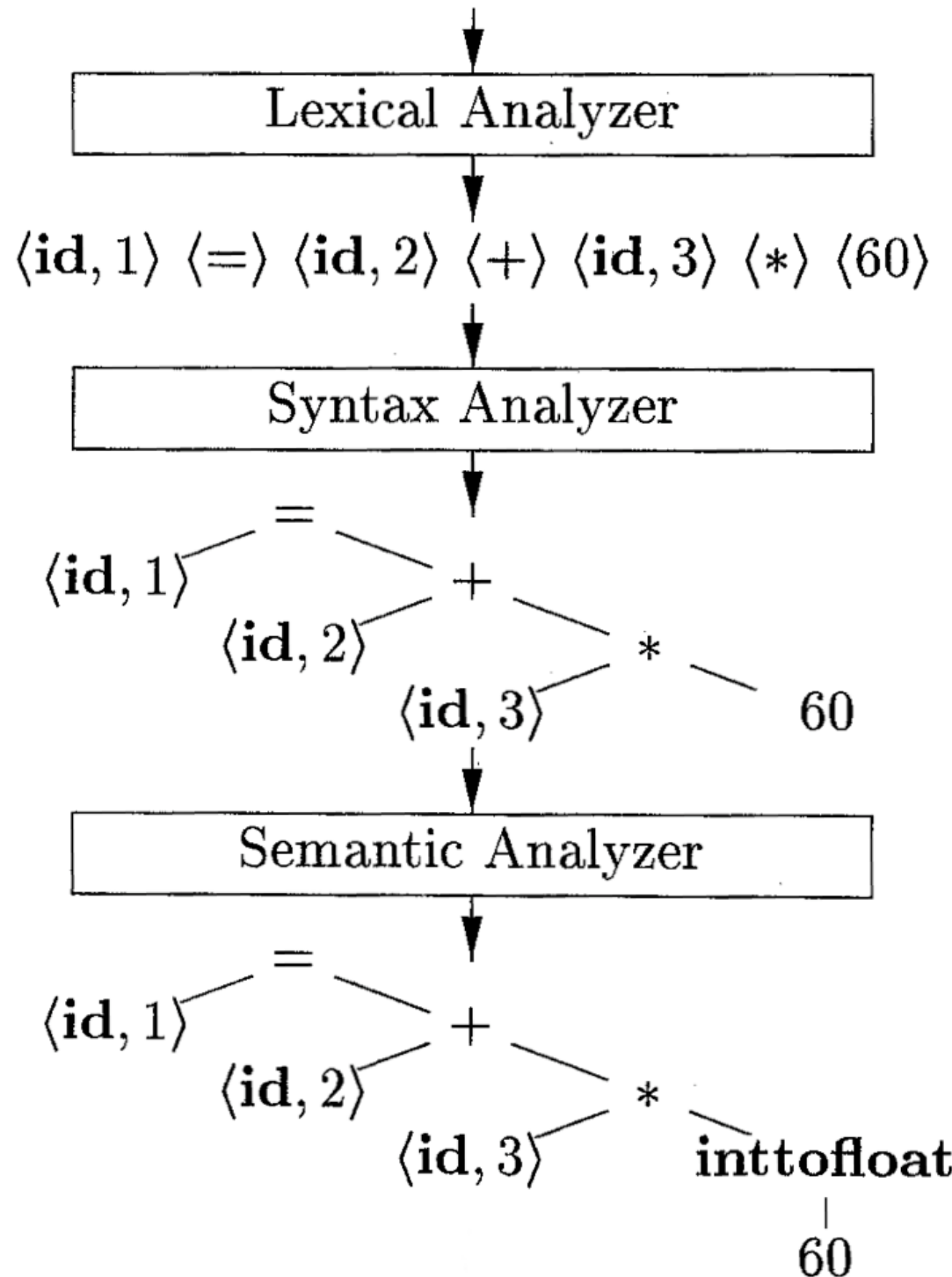
- Verifica se há sentido na operação (instrução) a ser realizada
- Verifica tipos da dados
 - Fracamente “tipada”: C, C++
 - Fortemente “tipada”: Pascal, Java
 - Tipos dinâmica: Python, PHP, Julia
- Exemplos
 - Verificação de tipos de variáveis em operações
 - Verificação de objeto-método
 - Verificação de divisão por zero

Análise Semântica

Exemplo

- Suponha que:
 - *Floating-pointing*: position, initial, rate
 - *Integer*: 60
- Integer → Floating-point number
 - **inttofloat**

position = initial + rate * 60

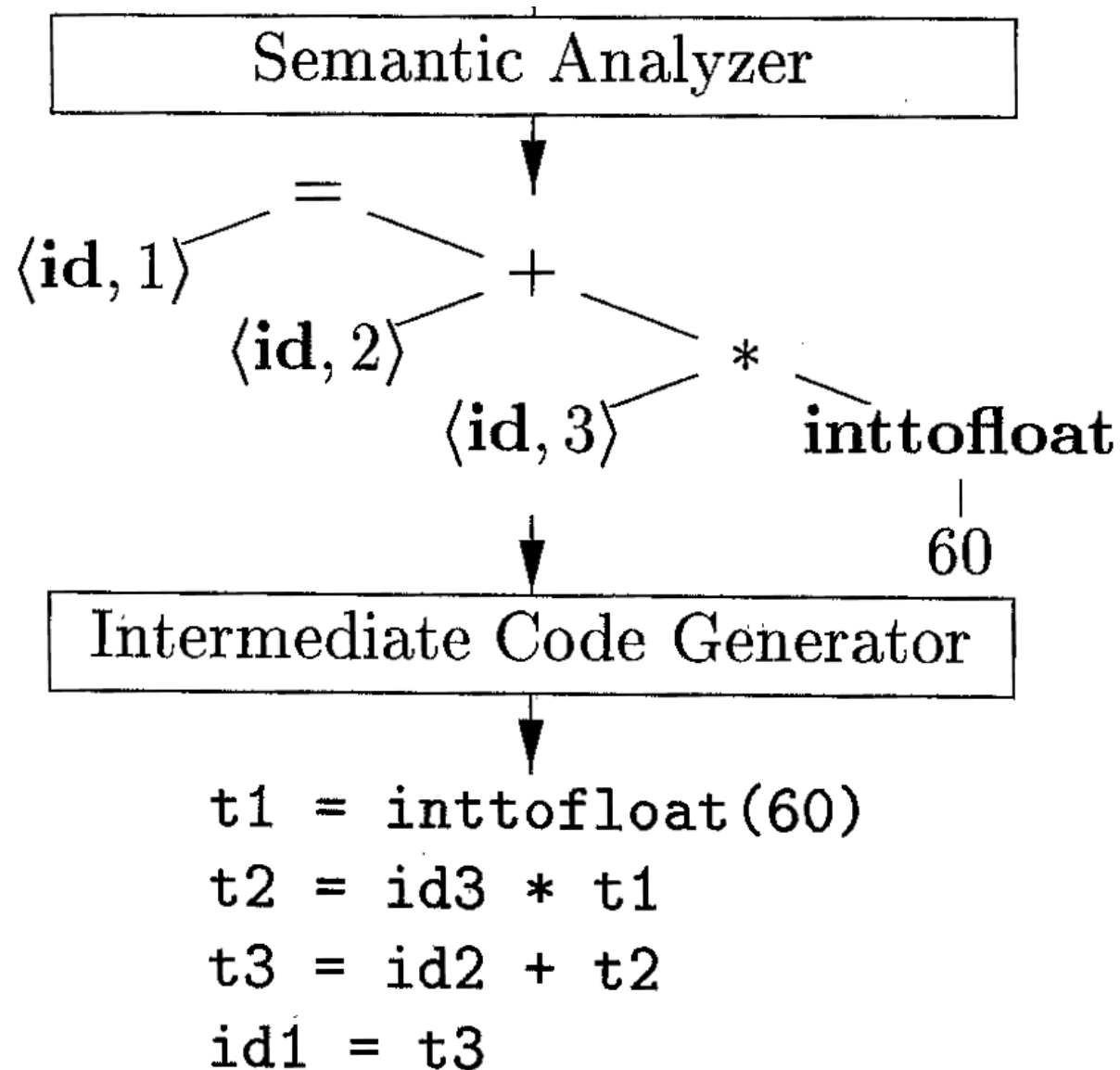


Geração de Código Intermediário

- Transforma o fonte em outro código mais simples de ser tratado pelo compilador
 - Nível mais baixo de abstração porém não é linguagem de máquina
 - Nível mais baixo que o fonte porém mais alto que o Assembly
- Representada por estruturas de dados
- Linearização da árvore sintática para verificação das instruções
- Pode ter referências à abstrações de alto nível ou detalhes da arquitetura da máquina (como disponibilidade de registradores, tamanho dos tipos primitivos)

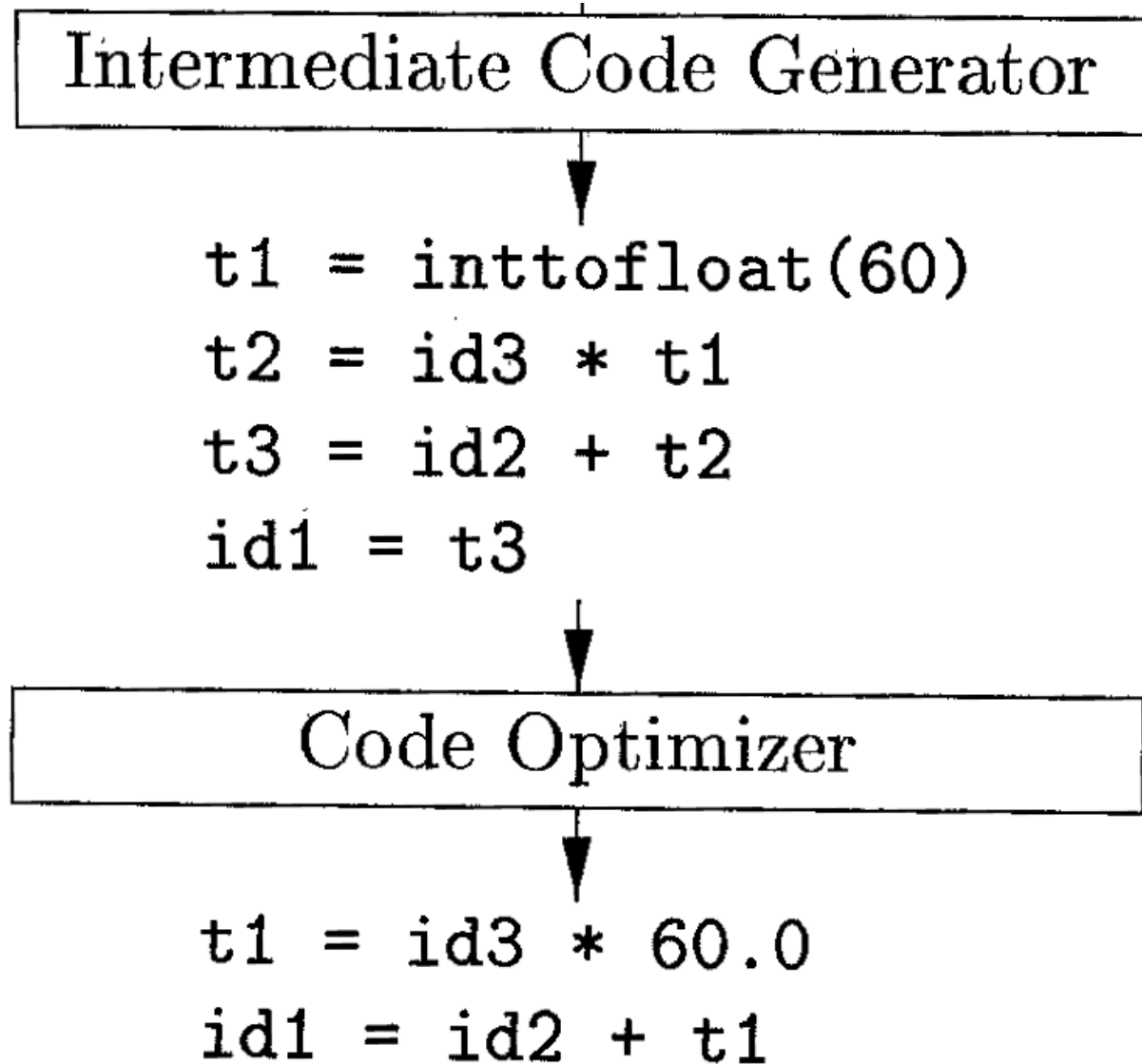
Geração de Código Intermediário

Exemplo



Otimização de Código

- Objetivo
 - **Melhorar** o código intermediário
 - Desempenho
 - Tamanho
 - Menos gasto de energia
- Arquitetura
 - Independente
 - Específica

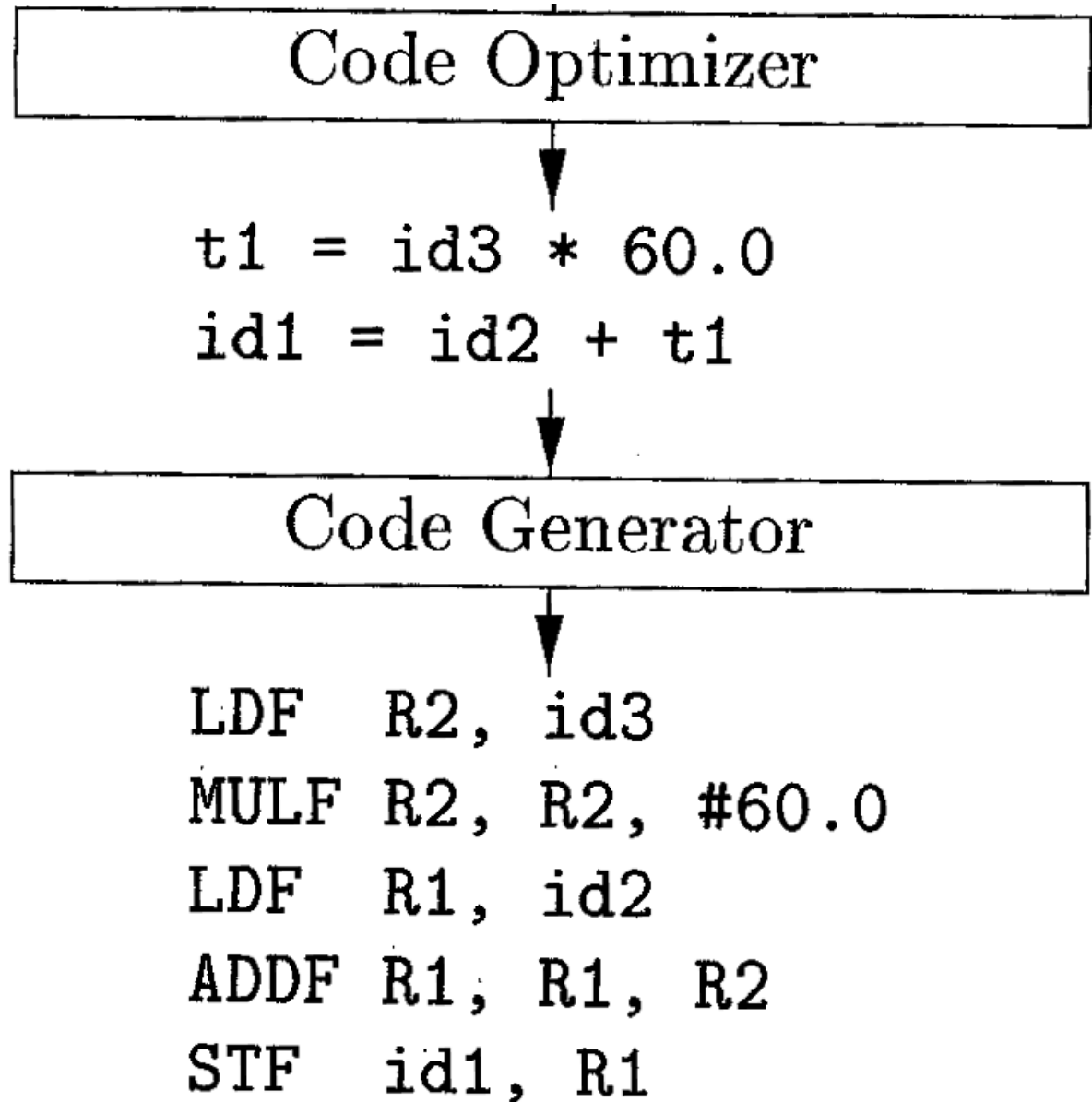


Geração de Código

- Traduz o código intermediário para o código de máquina
 - Intermediate Code → Assembly
 - Variáveis são mapeadas em registradores (ou localizações de memória)
 - Exemplo:
LDF R2, id3
- Traduz código de máquina para instruções de máquina
 - Assembly → Instructions Set

Geração de Código

- Primeiro operando
 - Destino (registrador)
- Operações
 - Load, Multiply, Add, Store
 - **F** indica que é um número de ponto flutuante
- Operadores
 - Registradores
 - **#** indica que é uma constante



Evolução das Linguagens de Programação

- Década de 1940
 - Instruções (binárias) de máquinas
- Operações simples
 - Operações aritméticas
 - Movimentação de dados
 - Comparar valores
- Programação difícil e tediosa

Evolução das Linguagens de Programação

- A partir da década de 1950
 - Linguagens mnemônicas de montagem (assembly)
 - Referências alfabéticas para as operações do processador
 - Exemplo: LDF R2, id3
- Adição de macros
 - Parametrização de instruções mais utilizadas
 - Primeiras “bibliotecas”
- Meados da década de 1950
 - Linguagens de mais alto nível: Fortran, Cobol, Lisp

Evolução das Linguagens de Programação - Fortran

- Fortran: *Formula Translating System*
 - Aplicações científicas

```
C AREA OF A TRIANGLE - HERON'S FORMULA
C INPUT - CARD READER UNIT 5, INTEGER INPUT
C OUTPUT - LINE PRINTER UNIT 6, REAL OUTPUT
C INPUT ERROR DISPLAY ERROR OUTPUT CODE 1 IN JOB CONTROL LISTING
      INTEGER A,B,C
      READ(5,501) A,B,C
501  FORMAT(3I5)
      IF(A.EQ.0 .OR. B.EQ.0 .OR. C.EQ.0) STOP 1
      S = (A + B + C) / 2.0
      AREA = SQRT( S * (S - A) * (S - B) * (S - C))
      WRITE(6,601) A,B,C,AREA
601  FORMAT(4H A= ,I5,5H B= ,I5,5H C= ,I5,8H AREA= ,F10.2,12HSQUARE UNITS)
      STOP
      END
```


Evolução das Linguagens de Programação - **Cobol**

- Cobol: *C*OMMON *B*USINESS *O*RIENTED *L*ANGUAGE
 - Aplicações comerciais

```
000100 IDENTIFICATION DIVISION.  
000200 PROGRAM-ID.      HELLOWORLD.  
000300  
000400*  
000500 ENVIRONMENT DIVISION.  
000600 CONFIGURATION SECTION.  
000700 SOURCE-COMPUTER.  RM-COBOL.  
000800 OBJECT-COMPUTER.  RM-COBOL.  
000900  
001000 DATA DIVISION.  
001100 FILE SECTION.  
001200  
100000 PROCEDURE DIVISION.  
100100  
100200 MAIN-LOGIC SECTION.  
100300 BEGIN.  
100400      DISPLAY " " LINE 1 POSITION 1 ERASE EOS.  
100500      DISPLAY "Hello world!" LINE 15 POSITION 10.  
100600      STOP RUN.  
100700 MAIN-LOGIC-EXIT.  
100800      EXIT.
```

Evolução das Linguagens de Programação - Lisp

```
DEFINE ((
(MEMBER (LAMBDA (A X) (COND ((NULL X) F)
      ( (EQ A (CAR X) ) T) (T (MEMBER A (CDR X))) )))
(UNION (LAMBDA (X Y) (COND ((NULL X) Y) ((MEMBER
      (CAR X) Y) (UNION (CDR X) Y)) (T (CONS (CAR X)
      (UNION (CDR X) Y)))) ))
(INTERSECTION (LAMBDA (X Y) (COND ((NULL X) NIL)
      ( (MEMBER (CAR X) Y) (CONS (CAR X) (INTERSECTION
      (CDR X) Y))) (T (INTERSECTION (CDR X) Y)) )))
))
INTERSECTION ((A1 A2 A3) (A1 A3 A5))
UNION ((X Y Z) (U V W X))
```

```
(defun factorial (n)
  (if (= n 0)
      1
      (* n (factorial (- n 1)))))
```

Evolução das Linguagens de Programação

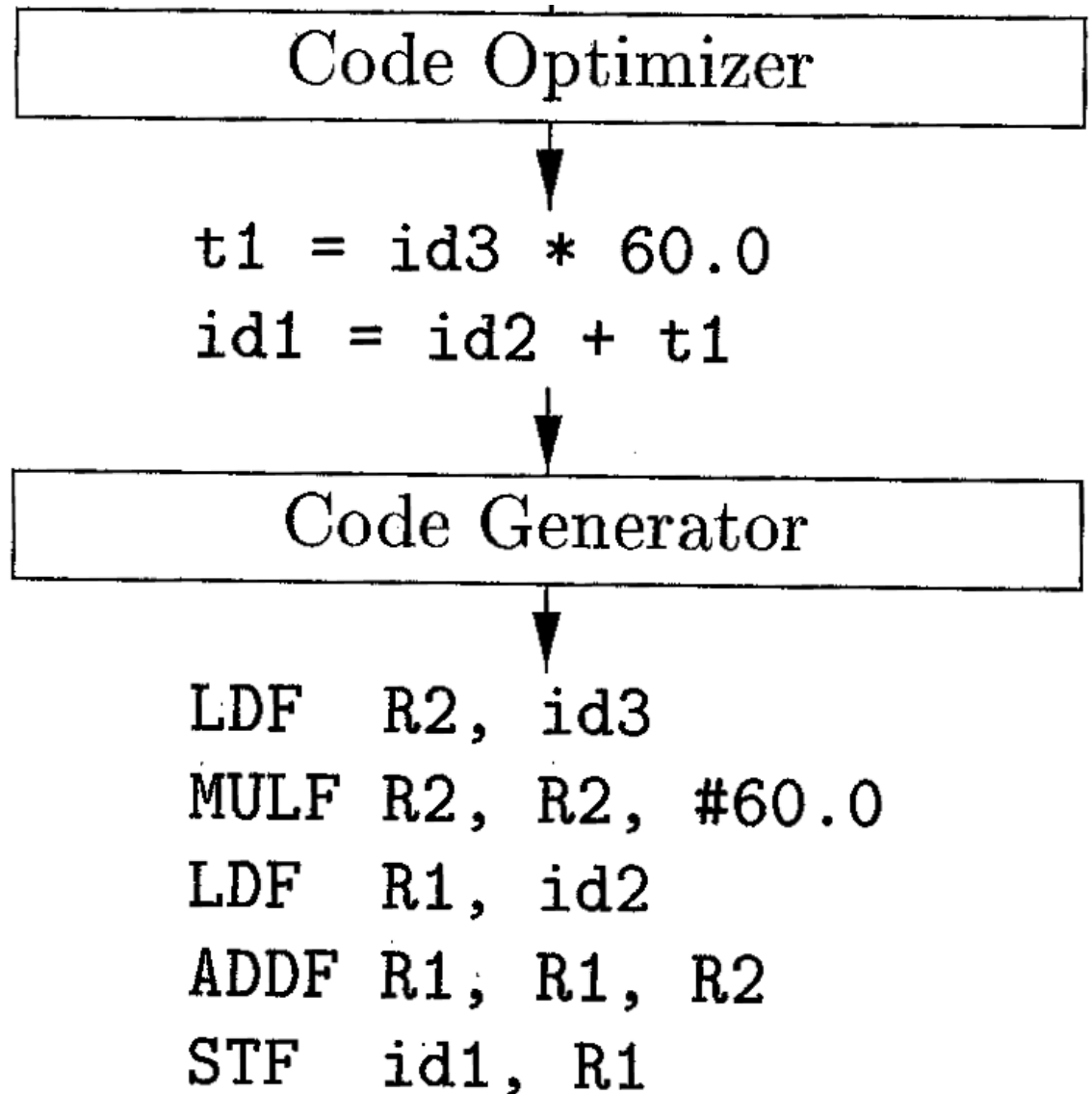
- Resumindo: objetivo principal:
 - Criar notações de mais alto nível para **facilitar a programação**
- Taxonomia (classificação) das linguagens
 - Princípios de projeto da linguagem
 - Aplicações
 - Organização das ideias (programa) para resolver problemas

Evolução das Linguagens de Programação – 1ª Geração

- Linguagens de máquina
- Conjuntos de instruções
 - Instructions Set
- Exemplo:
 - Arquitetura i386, Sparc

Evolução das Linguagens de Programação – 2ª Geração

- Linguagens de montagem
 - Assembly



Evolução das Linguagens de Programação – 3ª Geração

- Linguagens de alto nível
 - Fortran, Cobol, Lisp, C, C++, C#, **Java**

```
class SomeNumbers
{
    static int square (int x)
    {
        return x*x;
    }

    public static void main (String[] args)
    {
        int n=20;
        if (args.length > 0) // change default
            n = Integer.parseInt(args[0]);
        for (int i=0; i <= n; i++)
        {
            System.out.print("The square of " + i + " is ");
            System.out.println(square(i));
        }
    }
}
```

Evolução das Linguagens de Programação – 4ª Geração

- Linguagens para fins específicos
 - Nomad (relatórios), **SQL**, Postscript (text formatting)

```
CREATE TABLE STATION (ID INTEGER PRIMARY KEY,  
CITY CHAR(20), STATE CHAR(2), LAT_N REAL,  
LONG_W REAL);
```

```
INSERT INTO STATION VALUES (13, 'Phoenix', 'AZ', 33, 112);
```

```
INSERT INTO STATION VALUES (44, 'Denver', 'CO', 40, 105);
```

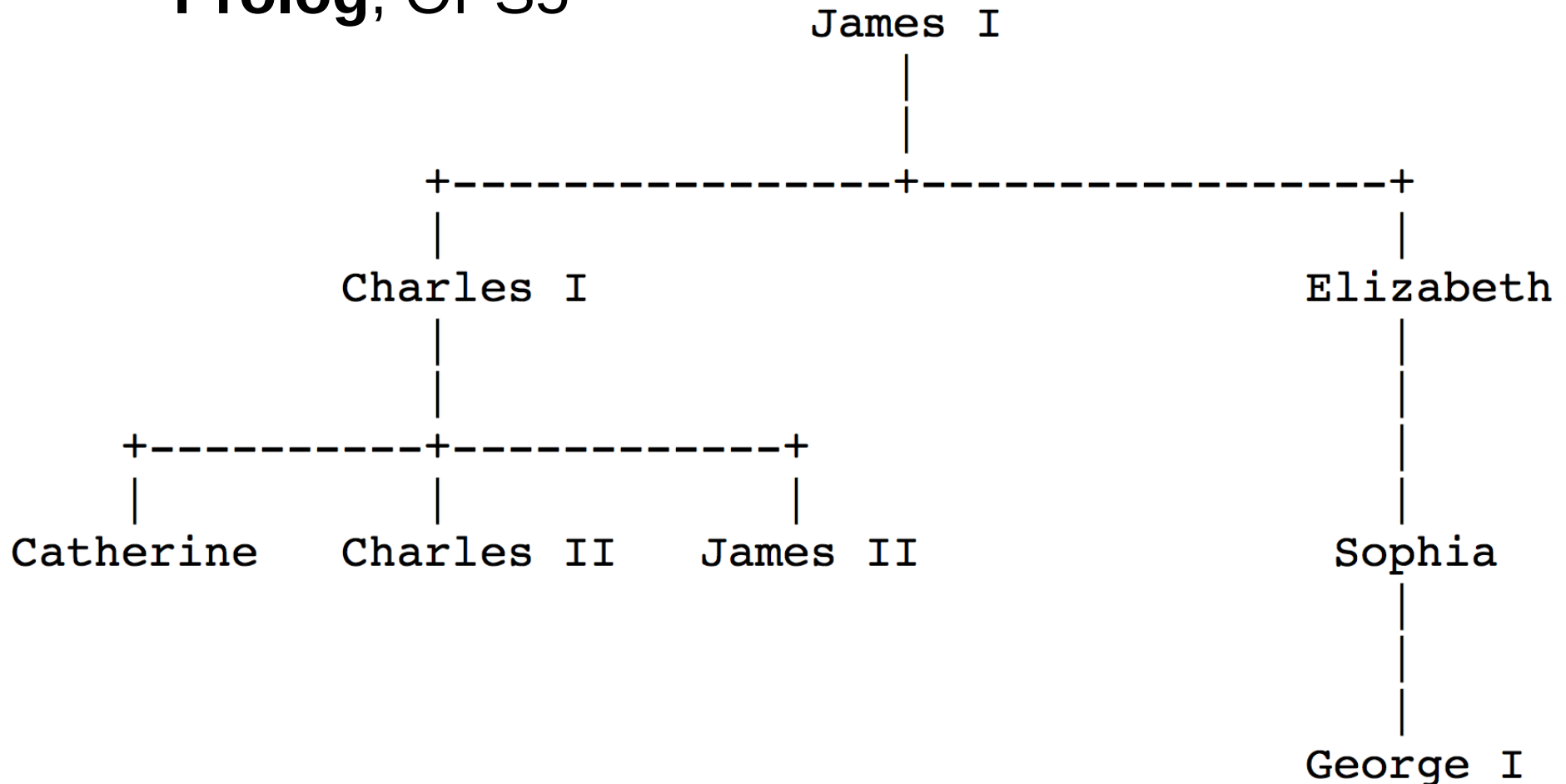
```
INSERT INTO STATION VALUES (66, 'Caribou', 'ME', 47, 68);
```

```
SELECT * FROM STATION WHERE LAT_N > 39.7;
```

ID	CITY	STATE	LAT_N	LONG_W
44	Denver	CO	40	105
66	Caribou	ME	47	68

Evolução das Linguagens de Programação – 5ª Geração

- Linguagens lógicas ou baseada em regras
 - **Prolog**, OPS5



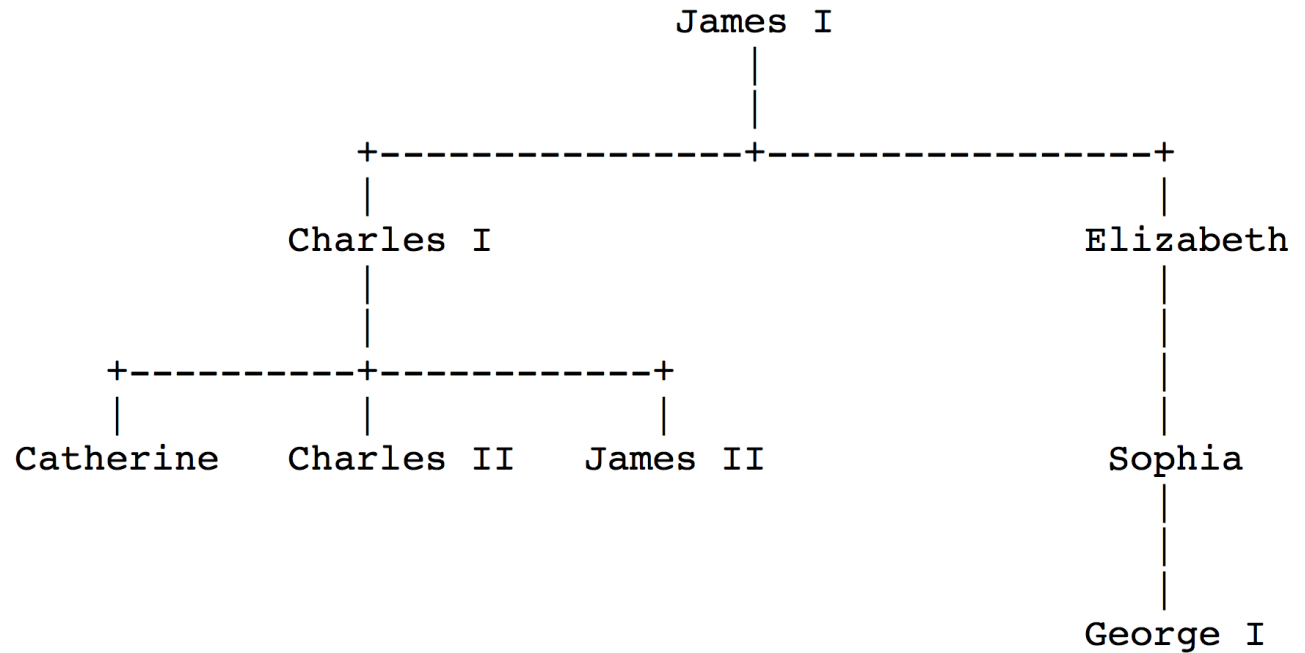
Prolog – Exemplo

<http://www.cs.toronto.edu/~sheila/384/w11/simple-prolog-examples.html>

```
male(james1).  
male(charles1).  
male(charles2).  
male(james2).  
male(george1).
```

```
female(catherine).  
female(elizabeth).  
female(sophia).
```

```
parent(charles1, james1).  
parent(elizabeth, james1).  
parent(charles2, charles1).  
parent(catherine, charles1).  
parent(james2, charles1).  
parent(sophia, elizabeth).  
parent(george1, sophia).
```



Prolog - Exemplo

Here is how you would formulate the following queries:

Was George I the parent of Charles I?

Query: **parent(charles1, george1).**

Who was Charles I's parent?

Query: **parent(charles1,X).**

Who were the children of Charles I?

Query: **parent(X,charles1).**

Now try expressing the following rules:

M is the mother of X if she is a parent of X and is female

F is the father of X if he is a parent of X and is male

X is a sibling of Y if they both have the same parent.

```
cd /Users/alage/Copy/professional/ufal/ensino/2015.1/introducao-computacao/aulas/
```

```
swipl exemplo-prolog.pl
```