INTRODUÇÃO: CARACTERÍSTICAS GERAIS

Melina Mongiovi
Everton L. G. Alves

Linguagens de Programação

- Características:
  - Expressividade
  - Universalidade
  - Suporte a Abstração
  - Simplicidade
  - Eficiência
  - **Implementável**

Implementável

- Uma LP deve ser passível de tradução para um formalismo que seja executável em uma máquina
  - Linguagem de máquina
  - Linguagem Intermediária (e.g., bytecode)

- Deve ser possível construir um **compilador** para a linguagem

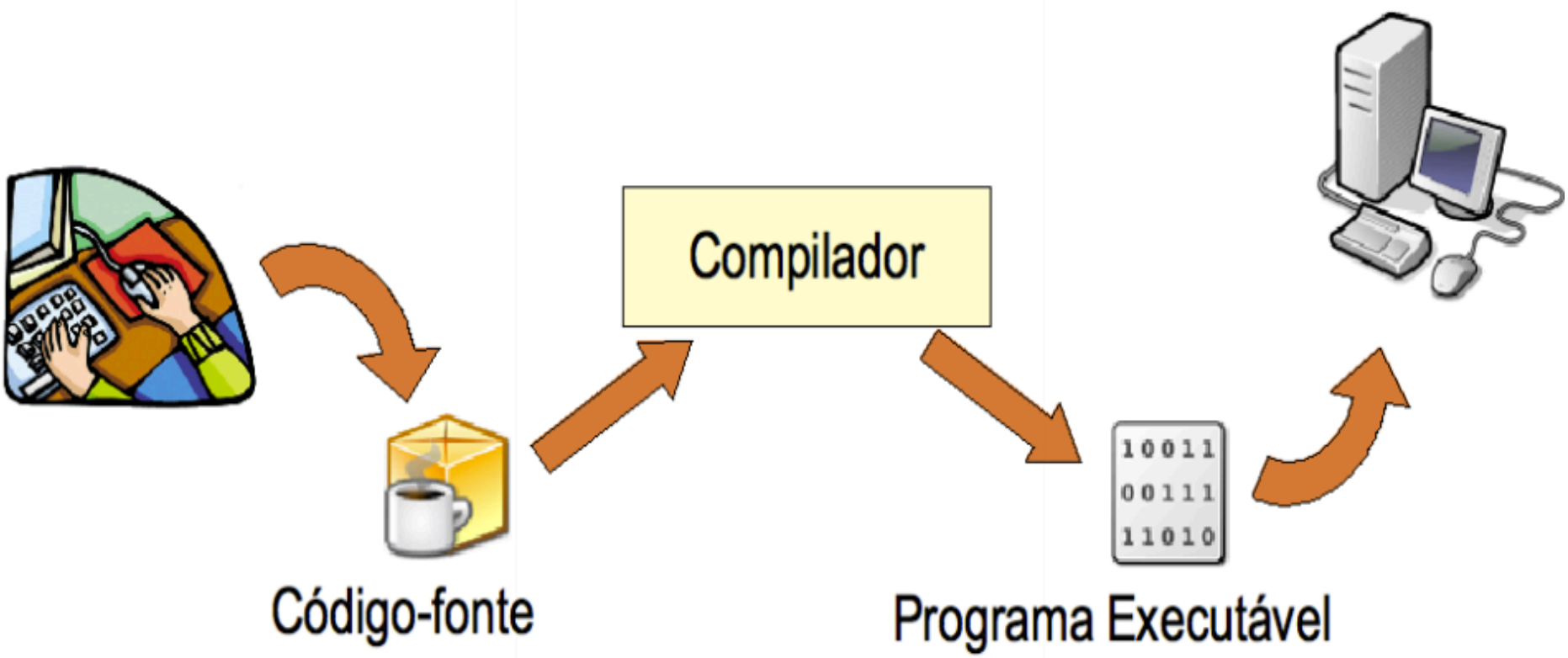
- Com a popularização de linguagens como Java, C, C#, Python, e sua forte adoção no mercado de TI, é comum nos depararmos com debates sobre as diferenças entre linguagens **interpretadas** e linguagens **compiladas**

- Qual a diferença?
- Decisão importante quando projetando uma LP

LP Compiladas x LP Interpretadas

- São linguagens que requerem compiladores
- Do ponto de vista do código fonte, toda linguagem de programação é compilada
- LP como C e C++ são compiladas **estaticamente**, e seus códigos fontes são transformados diretamente em linguagem de máquina

LP Compiladas



Compilador

- O que é?
  - É um programa que **traduz** um programa de uma **linguagem textual** facilmente entendida por um ser humano para uma **linguagem de máquina**, específica para um **processador** e **sistema operacional**

  - É um programa que tem a finalidade de **traduzir** um programa fonte $\mathbf{P_f}$, escrito em uma linguagem $\mathbf{L_f}$, para um programa objeto $\mathbf{P_o}$ semanticamente equivalente, porém em outra linguagem, $\mathbf{L_o}$

Compilação – Um Exemplo

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

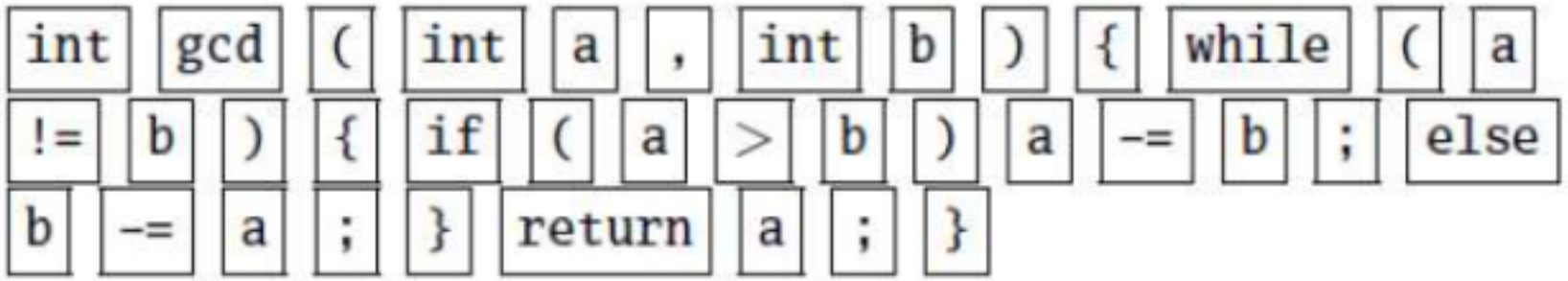
- **Fase 1**: o compilador vê o código fonte como uma sequência de caracteres

```
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
```

Compilação – Um Exemplo

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

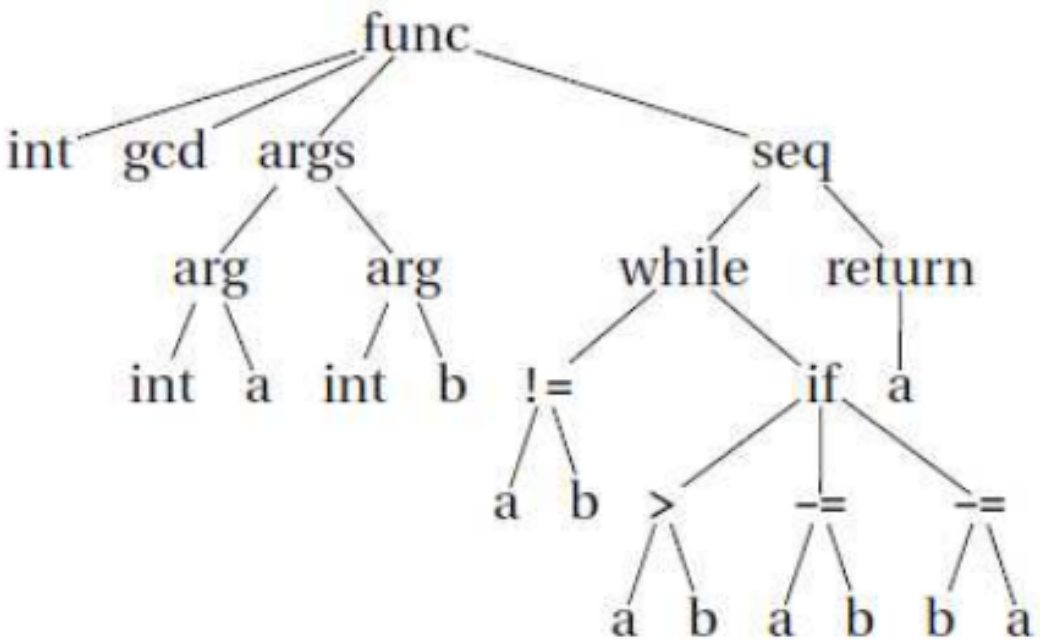
- **Fase 2**: Análise Léxica
  - Transforma o conjunto de caracteres em um conjunto de tokens



Compilação – Um Exemplo

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

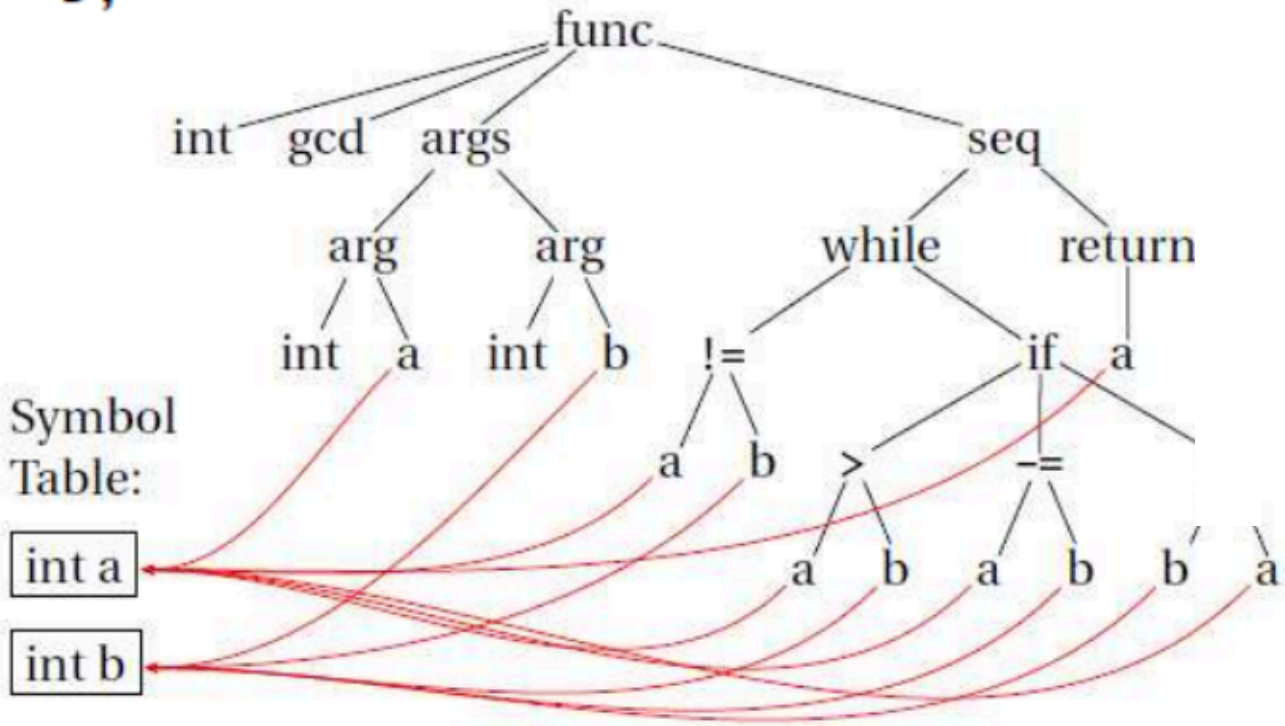
- **Fase 3**: Análise Sintática
  - Cria uma árvore abstrata a partir da gramática da LP



Compilação – Um Exemplo

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

- **Fase 4**: Análise Semântica
  - Resolve os símbolos e faz a checagem dos tipos



Compilação – Um Exemplo

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

- **Fase 5**: Tradução para linguagem intermediária

```
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
```

Compilação – Um Exemplo

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

- **Fase 6**: Tradução para linguagem de montagem (Assembly 80386)

```
gcd:  pushl %ebp
      movl  %esp,%ebp
      movl  8(%ebp),%eax
      movl  12(%ebp),%edx
.L8:  cmpl  %edx,%eax
      je    .L3
      jle   .L5
      subl  %edx,%eax
      jmp   .L8
.L5:  subl  %eax,%edx
      jmp   .L8
.L3:  leave
      ret
```

LPs tipicamente compiladas

- Ada
- ALGOL
- C
- C++
- CLEO
- COBOL
- Cobra
- Common Lisp
- D
- Delphi
- Eiffel
- Fortran
- JOVIAL
- LabVIEW
- Lush
- ML
- Objective-C
- Ocaml
- Pascal
- Sather
- Ubercode
- Urq
- Visual Basic
- Visual Foxpro
- Visual Prolog

LP Puramente Interpretadas

- O programa de interpretação atua como uma simulação de software de uma máquina cujo ciclo de busca-execução lida com instruções de programa de linguagem de alto nível em vez de instruções de máquina

- Implementação fácil de muitas operações de depuração

- Execução é de 10 a 100 vezes mais lenta do que nos sistemas compilados
- Cada vez que uma instrução for executada, ela precisa ser decodificada
- Requer mais espaço

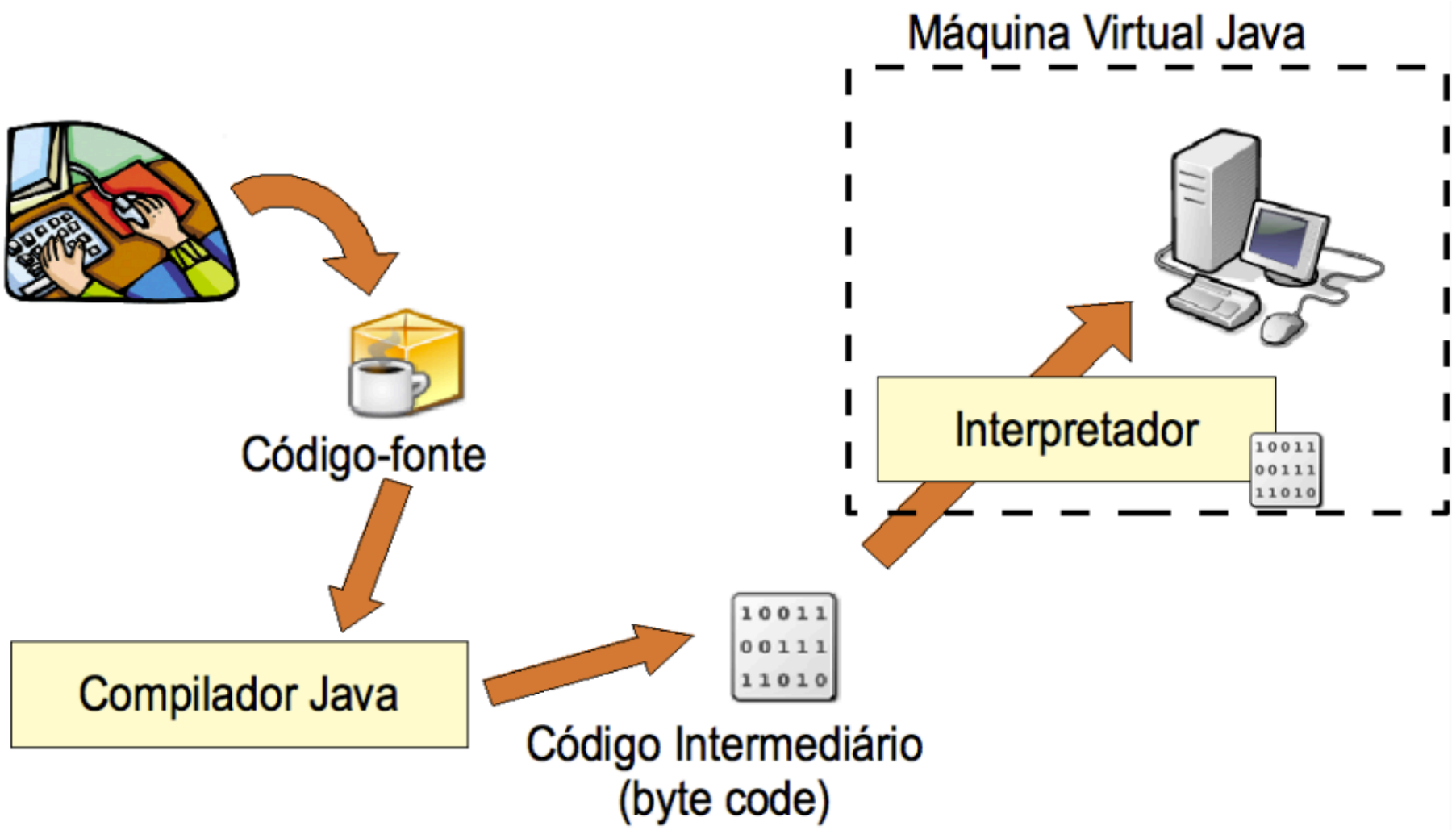
LP Híbridas

- LP mais modernas como Java, C# e Python têm seus códigos fontes transformados em uma **linguagem intermediária**, que será interpretada pela **máquina virtual** da linguagem quando o programa for executado

- O processo de **interpretação** da linguagem intermediária
  - Ocorre durante a **execução** do programa
  - Consiste na tradução dos comandos da linguagem intermediária para linguagem de máquina

- Em tempo de execução, o código intermediário pode ser encarado como um “código fonte” que será compilado **dinamicamente** pelo interpretador da linguagem em código de máquina

LP Interpretada - Java



Compilada X Interpretada

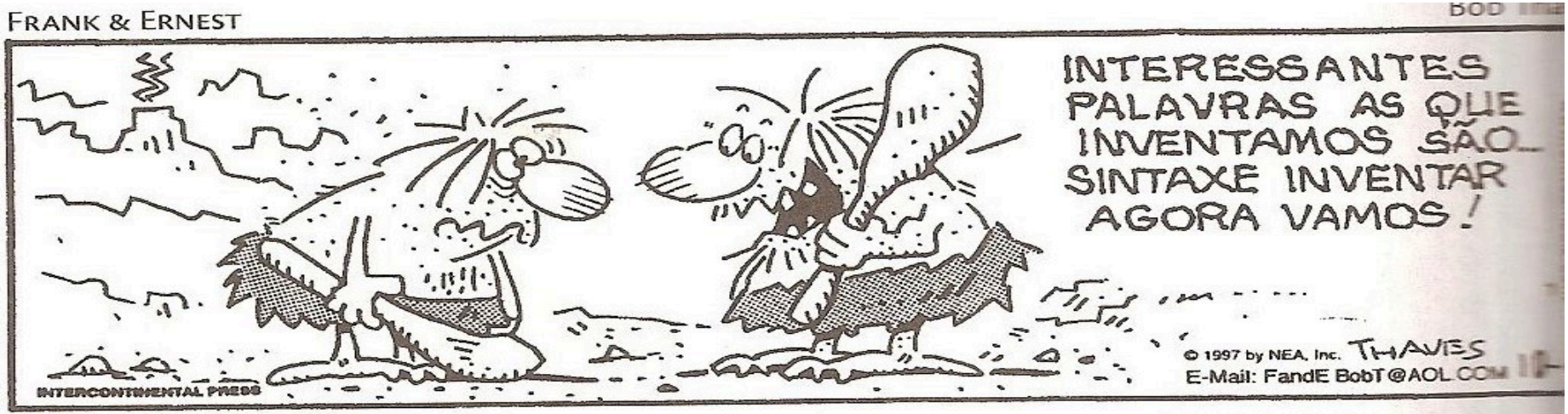
- Compilação:
  - Eficiência
  - Problemas com portabilidade e depuração
  - Ex: Fortran, Pascal, C, C++

- Interpretação:
  - Flexibilidade, portabilidade, facilidade para prototipação e depuração
  - Problemas com eficiência e maior consumo de memória
  - Ex: TACOMA, PHP, Java, C#, Python

Sintaxe X Semântica

- Uma linguagem é um conjunto de **regras sintáticas** e **semânticas** usadas para definir uma forma de comunicação

- Sintaxe:
  - Formato do programa
  - Como expressões, comandos, declarações e outras construções devem ser arranjados para formar programas



Sintaxe X Semântica

- Semântica:
  - Significado do programa
  - Como um programa bem formado deve ser comportar quando executados em uma computador

Sintaxe X Semântica

PASCAL

```
…
var nome: string;
begin
    clrscr;
    writeln('Digite o primeiro nome:');
    readln(nome);
    writeln(nome);
    readkey;
end
```

C

```
…
char nome[40]
int main(){
    system(“cls”);
    printf("Digite o primeiro nome: ”);
    gets(nome);
    printf(nome);
    return 0;
}
```

Sintaticamente diferentes mas semanticamente idênticos!

Sintaxe x Semântica

- Sintaxe influencia a forma como o programa deve ser escrito

- Semântica é a parte mais importante e que realmente caracteriza a linguagem
  - Por que?
    - Determina como o programa será **composto** pelo programador, **entendido** por outros programadores e **interpretado** pelo computador

Sintaxe x Semântica

- Como verificamos a sintaxe de um programa?
  - Compiladores!

- Como verificamos a semântica de um programa?
  - Em tempo de execução!!
  - Desentendimentos semânticos levam a programas com comportamentos indesejados

Sintaxe x Semântica

- Todo programa sintaticamente correto é também correto semânticamente?
  - Não!
  - Programas corretos sintaticamente podem resultar em erros de tradução ou execução

- “João é um solteiro casado”
  - Gramaticamente correto
  - Expressa um significado que não pode ser verdadeiro

Sintaxe X Semântica

- A quem interessa?
  - Projetistas de LPs
  - A quem escreve compiladores
  - Programadores

Sintaxe

- Como definir a sintaxe de uma linguagem?
  - Gramáticas

- Entre as gramáticas livres de contexto, a mais utilizada é a BNF– Forma de Backus-Naur
  - Usa abstrações para representar estruturas sintáticas

Sintaxe

```
expressão → expressão '+' termo | expressão '-' termo | termo
termo → termo '*' fator | termo '/' fator | fator
fator → identificador | constante | '(' expressão ')'
```

b*b – 4*a*c

Semântica

- Formas de apresentar a semântica de uma linguagem:
  - Semântica Operacional
    - Como o programa é executado?
    - Que operações são realizadas?
  - Semântica Denotacional
    - O que o programa significa?
    - Que objetos matemáticos ele denota?
  - Semântica Axiomática
    - Quais proposições lógicas são válidas para um programa?
- Detalhes na disciplina de Compiladores!!

Semântica Operacional

- Dá uma indicação de como o programa deve ser executado através da aplicação de um conjunto de **regras**
- Exemplo: Loop em C

C Statement

```
for (expr1; expr2; expr3) {
   ...
}
```

Meaning

```
        expr1;
loop:   if expr2 == 0 goto out
        ...
        expr3;
        goto loop
out:    ...
```

Semântica Operacional

- Dá uma indicação de como o programa deve ser executado através da aplicação de um conjunto de **regras**
- Exemplo: soma de números binários

R1: ε + x → x
R2: x + ε → x
R3: 0x → x (x != ε)
R4: x0 + y0 → (x + y) 0
R5: x1 + y0 → (x + y) 1
R6: x0 + y1 → (x + y) 1
R7: x1 + y1 → (x + y + 1) 0

111 + 101
R7 → ((11 + 10) + 1) 0
R5 → (1 + 1) 1 + 1) 0
R7 e R1 → ((10) 1 + 1) 0
R7 → ((10 + ε) + 1) 0) 0
R6 → (((1 + ε) 1) 0) 0
= 1100

Semântica Denotacional

- Originalmente desenvolvido Scott e Strachey (1970)
- Define o significado de construções da linguagem em um domínio matemático
- Baseado na teoria das **funções recursivas**
- Método mais **abstrato** para descrição semântica

Semântica Denotacional

- Exemplo:
  - Definindo a semântica de um número binário pelo seu decimal equivalente

- Sintaxe Abstrata:

<num_bin> = 0
| 1
| <num_bin> 0
| <num_bib> 1

Semântica Denotacional

- A função de avaliação M_{bin} relaciona os objetos sintáticos com valores decimais

$M_{bin}(\text{‘0’}) = 0$

$M_{bin}(\text{‘1’}) = 1$

$M_{bin}(\text{<num\_bin> ‘0’}) = 2 * M_{bin}(\text{<num\_bin>})$

$M_{bin}(\text{<num\_bin> ‘1’}) = 2 * M_{bin}(\text{<num\_bin>}) + 1$

Semântica Axiomática

- Baleada em **lógica** matemática
- Método usado para **provar** a **exatidão** dos programas
- Cada instrução de um programa tanto é precedida como seguida de uma **expressão lógica** que especifica **restrições** a variáveis
- As expressões lógicas são usadas para especificar o **significado** das instruções
- As restrições são descritas pela notação do cálculo de predicados

Semântica Axiomática

- O problema típico que o programador enfrenta é escrever um programa que irá transformar dados satisfazendo algumas propriedades ou asserções ‘P’ em resultados satisfazendo ‘Q’

- {P} S {Q}
  - P = Predicado (pré-condição)
  - S = Instrução
  - Q = Pós-condição

x = 2 * y – 3 **{x > 25}**

2 * y - 3 > 25
2 * y > 28
y > 14

{y > 14} 2 * y – 3 **{x > 25}**