



# Linguagens de Programação

## 1. Preliminares

# Por que estudar LP?

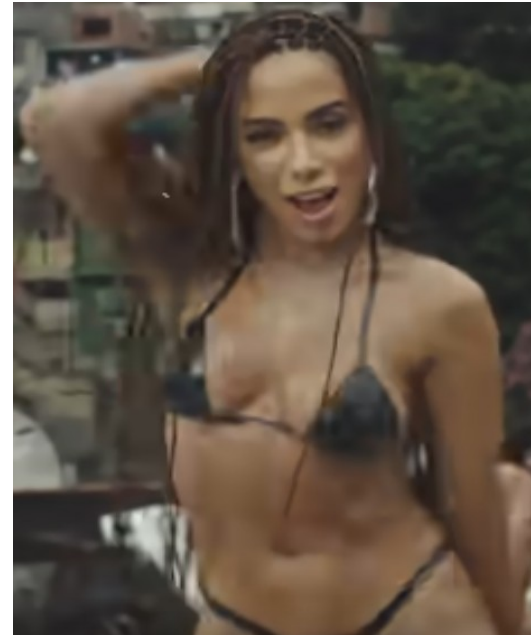
- Se precisa explicar, alguma coisa está errada...



<https://www.youtube.com/watch?v=yS3zSb2u-BU>

# Por que estudar LP?

- **Aumentar capacidade de expressar idéias:** a profundidade com a qual as pessoas pensam (e o poder de abstração) é influenciada pelo poder de expressividade da linguagem com a qual elas comunicas suas idéias.



# Por que estudar LP?



<https://www.youtube.com/watch?v=b-stLY6GVIw>

Oh amigos, mudemos de tom!  
Entoemos algo mais agradável  
E cheio de alegria!

Alegria, mais belo fulgor divino  
Filha de Elíseo  
Ébrios de fogo entramos  
Em teu santuário celeste!  
Tua magia volta a unir  
O que o costume rigorosamente dividiu  
Todos os homens se irmanam  
Onde pairar teu voo suave

A quem a boa sorte tenha favorecido  
De ser amigo de um amigo  
Quem já conquistou uma doce companheira  
Rejuble-se conosco!  
Sim, mesmo se alguém conquistar apenas  
uma alma  
Uma única em todo o mundo  
Mas aquele que falhou nisso  
Que fique chorando sozinho!

Alegria bebem todos os seres  
No seio da Natureza  
Todos os bons, todos os maus  
Seguem seu rastro de rosas  
Ela nos deu beijos e vinho e  
Um amigo leal até a morte  
Deu força para a vida aos mais humildes  
E ao querubim que se ergue diante de Deus!

Alegres, como voam seus sóis  
Através da esplêndida abóbada celeste  
Sigam irmãos sua rota  
Gozosos como o herói para a vitória

Alegria, mais belo fulgor divino  
Filha de Elíseo  
Ébrios de fogo entramos  
Em teu santuário celeste!  
Abracem-se milhões de seres!  
Enviem este beijo para todo o mundo!  
Irmãos! Sobre a abóbada estrelada  
Deve morar o Pai Amado  
Vos prosternais, Multidões?  
Mundo, pressentes ao Criador?  
Buscais além da abóbada estrelada!  
Sobre as estrelas Ele deve morar

# Por que estudar LP?



[https://www.youtube.com/watch?v=kDhptBT\\_-VI](https://www.youtube.com/watch?v=kDhptBT_-VI)

Vai, malandra, an an  
Ê, tá louca, tu brincando com o bumbum  
An an, tutudum, an an  
Vai, malandra, an an  
Ê, tá louca, tu brincando com o bumbum  
An an, tutudum, an an

Tá pedindo, an, an  
Se prepara, vou dançar, presta atenção  
An, an, tutudum an, an  
Cê aguenta an, an  
Se eu te olhar  
Descer, quicar até o chão

Desce, rebola gostoso  
Empina me olhando  
Te pego de jeito  
Se eu começar embrazando contigo  
É taca, taca, taca, taca

AAAARRRRRRGGGGGHHHHHHH



# Por que estudar LP?

- **Embasamento para escolher linguagens adequadas:** se você conhece várias, sabe escolher e combinar.



<https://www.youtube.com/watch?v=63jv0JcDNUg>

Mozart - Requiem in D minor, K. 626

2. Dies irae

Allegro assai

Tutti

Soprano

Alto

Tenore

Basso

Piano

Di - es i - rae, di - es

il - la solvet aedum in fa - vil - la te - ste Da - vid cum Sy -

il - la solvet aedum in fa - vil - la te - ste Da - vid cum Sy -

il - la solvet aedum in fa - vil - la te - ste Da - vid cum Sy -

il - la solvet aedum in fa - vil - la te - ste Da - vid cum Sy -

46:54 • Dies irae (choir) >

Scroll for details

# Por que estudar LP?

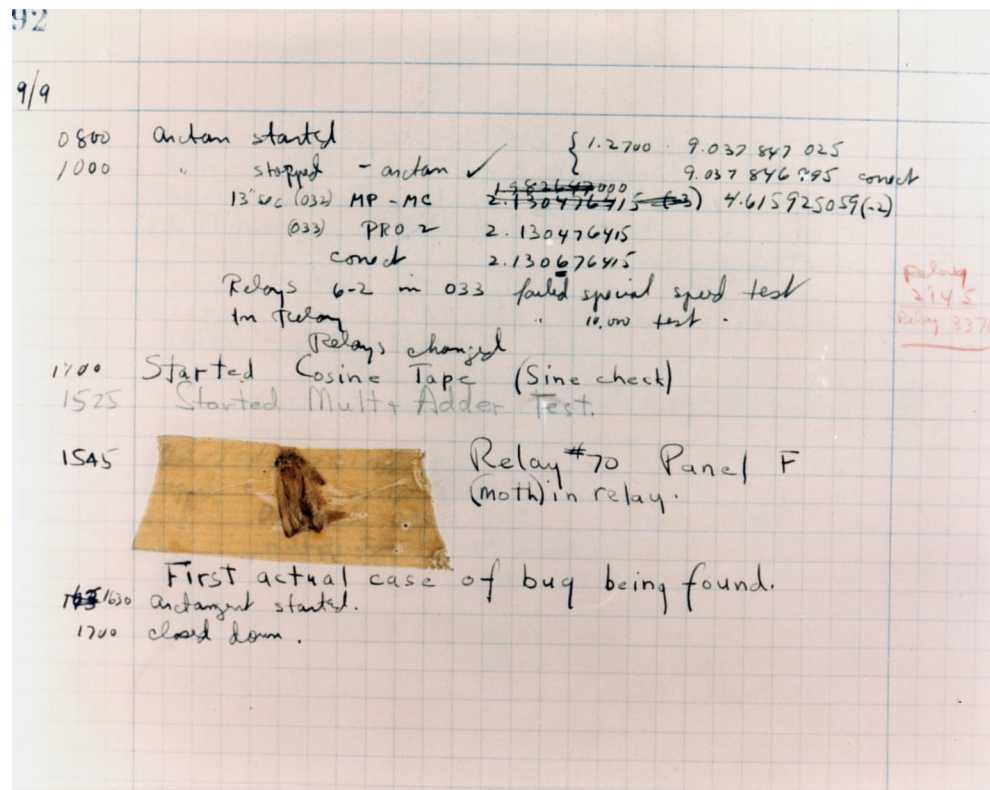
- **Aumento da habilidade em aprender novas linguagens:** “As mais difíceis são as 10 primeiras” (Carlos Amaral Freire).



# Por que estudar LP?

- **Entender a importância da implementação:** torna claro porque certas linguagens foram projetadas de certa forma; nos habilita a usar a linguagem da maneira que foi planejada; correção de bugs.

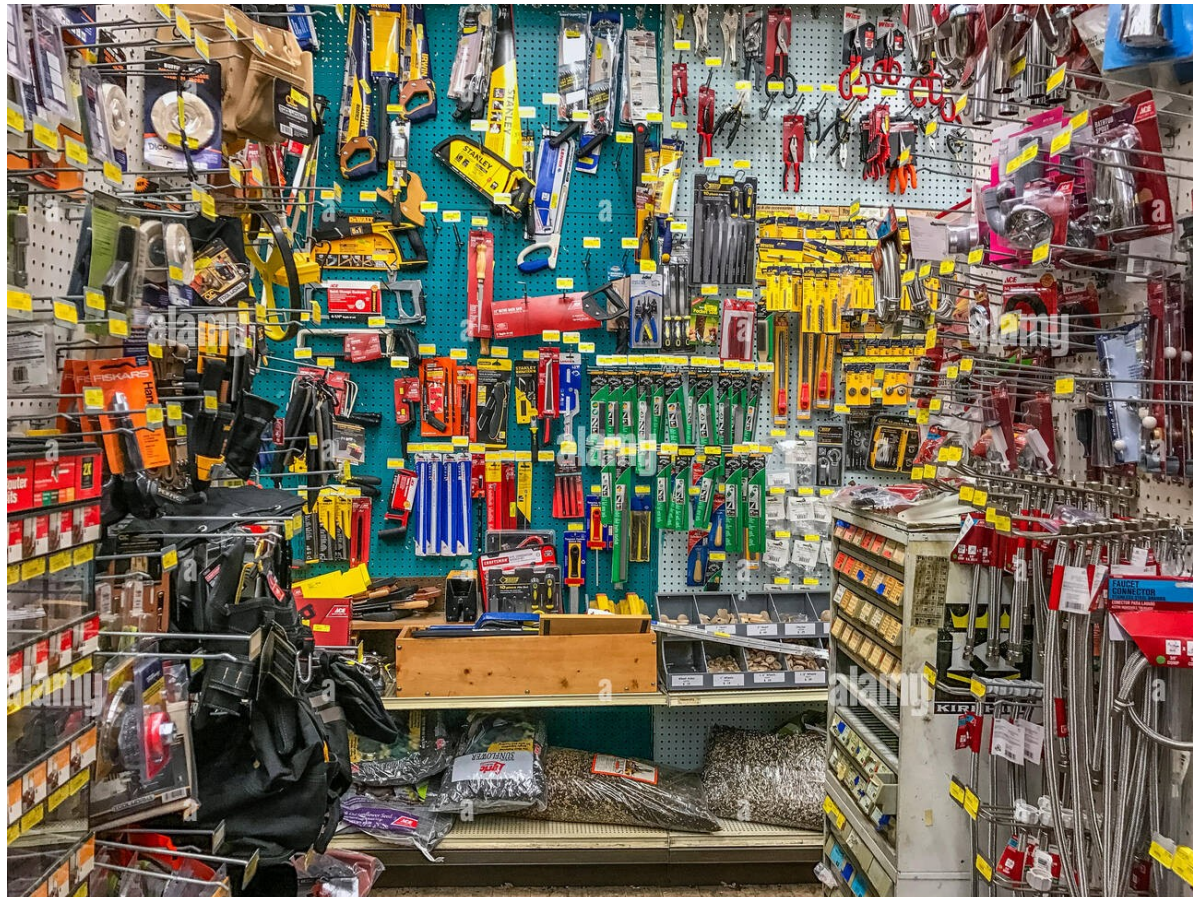
Grace Hooper





# Por que estudar LP?

- **Melhorar o uso de linguagens já conhecidas:** você conhece todos os recursos de sua linguagem preferida? Você domina a gramática do português?



# Por que estudar LP?

- **Avanço geral da computação:** se você só tem martelo, tudo é prego! As linguagens populares não são sempre as melhores disponíveis (Algol 60 x Fortran).



Se a única ferramenta que você tem é um martelo,  
tudo começa a parecer com um prego.

(Abraham Maslow)

# Domínios da programação

- **Aplicações científicas:**
  - Fortran, ALGOL 60, R, MATLAB, MATHEMATICA
- **Aplicações comerciais:**
  - COBOL
- **Inteligência artificial:**
  - LISP, Scheme, Prolog
- **Geral/Web:**
  - C, Java, Python, JavaScript, PHP, ...

# Qual linguagem é melhor?

- Critérios de avaliação das linguagens:



- Legibilidade, Escrita, Confiabilidade e Custo

Característica	CRITÉRIOS		
	Legibilidade	Facilidade de escrita	Confiabilidade
Simplicidade	•	•	•
Ortogonalidade	•	•	•
Tipos de dados	•	•	•
Projeto de sintaxe	•	•	•
Suporte para abstração		•	•
Expressividade		•	•
Verificação de tipos			•
Tratamento de exceções			•
Apelidos restritos			•



# Qual linguagem é melhor?

- **Legibilidade:** o que esse programa faz?

```
#include <stdio.h>
/* Ian Phillipps, International Obfuscated C Code Contest */
int main(t,_,a)
char *a;
{return!0<t?t<3?main(-79,-13,a+main(-87,1-_,
main(-86,0,a+1)+a)):1,t<_?main(t+1,_,a):3,main(-94,-27+t,a
)&&t==2?<13?main(2,_,+1,"%s %d %d\n"):9:16:t<0?t<-72?main(
t,"@n'+,#'/*{ }w+/w#cdnr/+,{ }r/*de}+,/*{ *+,/w{ %+,/w#q#n+,/#{ l,+,/n{n+ \
,/+ #n+,/ #;#q#n+,/+k#; *+,/'r : 'd* '3,}{w+K w'K: '+}e#';dq# 'l q#'+d'K#!/\
+k#;q# 'r}eKK#}w' r}eKK{nl}' / #;#q#n' ) { )#}w' ) { ) {nl}' / + #n' ;d}rw' i;# ) {n \
l}! / n {n#'; r{#w' r nc{nl}' / # {l, + 'K {rw' iK{ ; [{nl}' / w#q# \
n'wk nw' iwk{KK{nl}! / w{ % 'l##w# ' i; : {nl}' / * {q# 'ld;r' } {nlwb! / *de}' c \
; ; {nl' - { }rw}' / +, } ## ' * } #nc, ', #nw}' / +kd' +e} +; \
# 'rdq#w! nr' / ' ) } + } {rl# ' {n' ' ) # } ' + } ## ( ! ! / " )
:t<-50?_==*a ?putchar(a[31]):main(-65,_,a+1):main((*a == '/' )+t,_,a \
+1 ):0<t?main ( 2, 2 , "%s"): *a=='/' ||main(0,main(-61,*a, "!ek;dc \
i@bK' (q) - [w] *%n+r3#l, { } : \nuwloca-0;m .vpbks,fxntdCeghiry"),a+1);}
```

# Qual linguagem é melhor?

- **Facilidade de escrita:** qual é mais fácil de escrever?

```
#include <boost/iterator/counting_iterator.hpp>
#include <algorithm>

int factorial(int n)
{
    return std::accumulate(boost::counting_iterator<int>(1),
                           boost::counting_iterator<int>(n+1), 1, std::multiplies<int>());
}
```

```
(defun factorial (n)
  (if (zerop n) 1 (* n (factorial (- n 1)))))
```

# Qual linguagem é melhor?

- **Confiabilidade:** tem recursos para testar e detectar erros durante a compilação e/ou execução?

```
int main()
{
    const int SIZE = 10;
    double *doubleVals;

    if ((doubleVals = (double*)malloc(sizeof(double)*SIZE)) == NULL)
    {
        exit(EXIT_FAILURE);
    }

    doubleVals[SIZE - 1] = 20.21;
    printf("%lf\n", doubleVals[SIZE - 1]);

    doubleVals[SIZE] = 25.99; // ERRO! Você está escrevendo em um pedaço
                             // de memória que não pertence ao seu
                             // programa!
    printf("%lf\n", doubleVals[SIZE]);

    return 0;
}
```

# Qual linguagem é melhor?

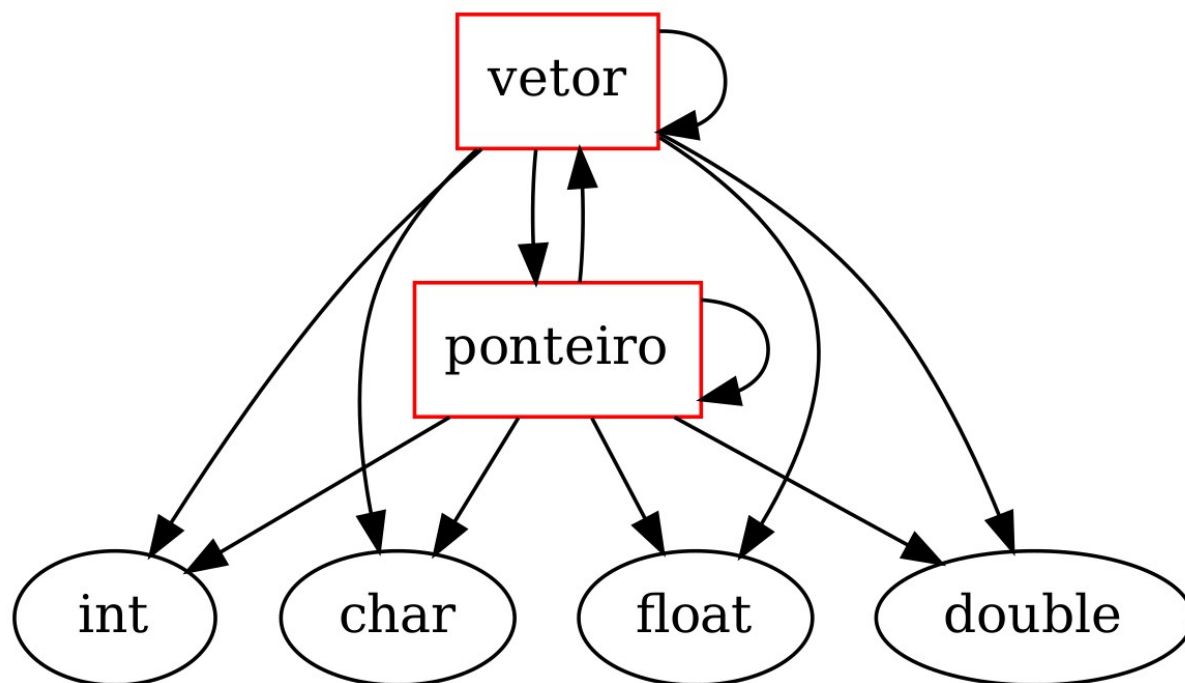
- **Simplicidade:** nem 8 nem 80!
  - Quantidade de primitivos
  - Multiplicidade de recursos
  - Sobrecarga de operadores

```
count = count + 1  
count += 1  
count++  
++count
```



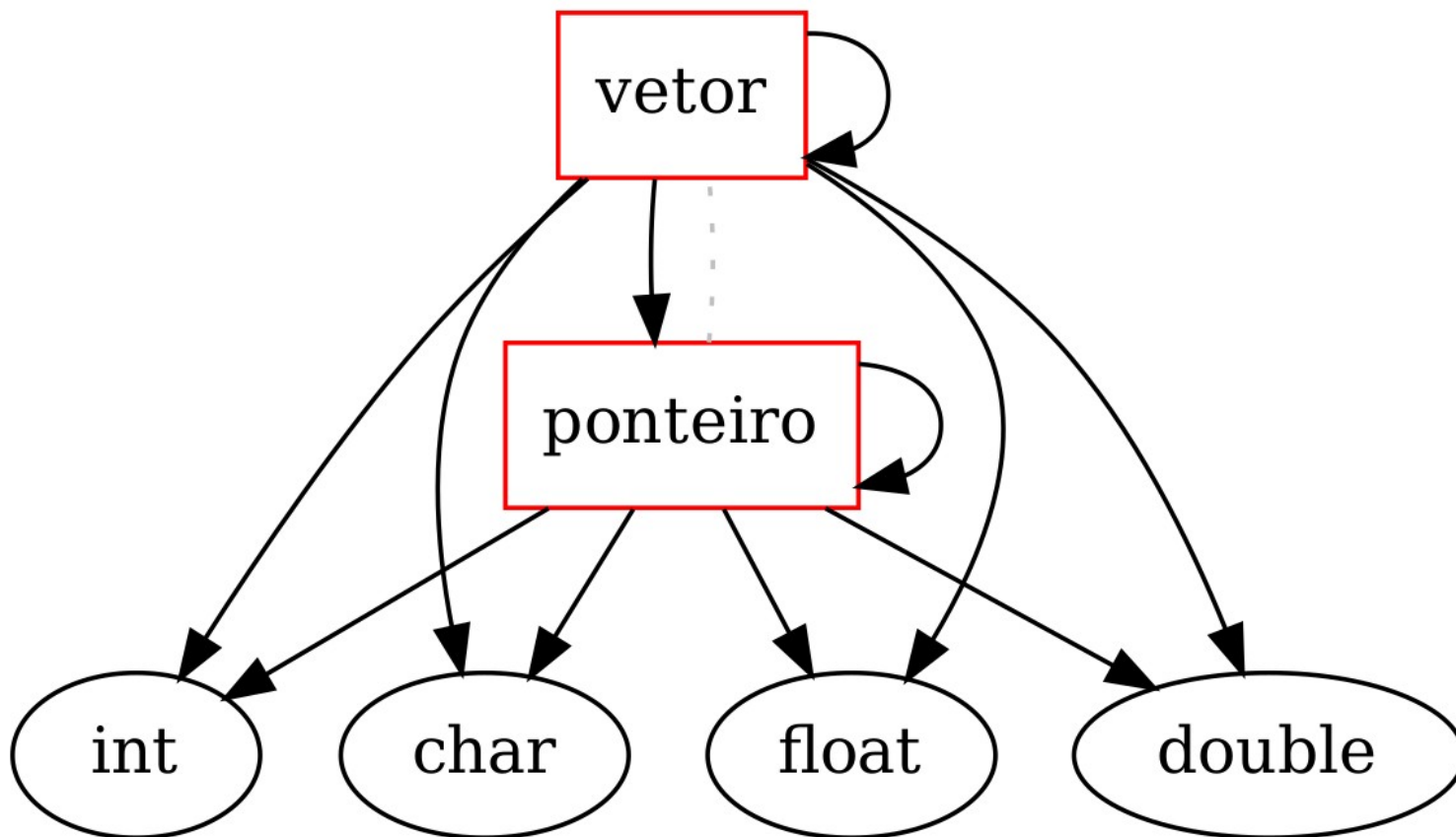
# Qual linguagem é melhor?

- **Ortogonalidade:** um conjunto pequeno de instruções primitivas pode ser combinado a um número pequeno de formas para construir as estruturas de controle e de dados da linguagem, todas as combinações possíveis de primitivos devem ser válidas e independentes do contexto!



# Qual linguagem é melhor?

- **Ortogonalidade:** falta de ortogonalidade leva a exceções nas regras da linguagem e estruturas úteis não poderão ser definidas!



# Qual linguagem é melhor?

- **Ortogonalidade:** falta de combinações diminui ortogonalidade e exceções!

```
ADDL operand_1, operand_2
```

```
operand_2 ← contents(operand_1) + contents(operand_2)
```

Memória + Memória  
Memória + Registrador  
Registrador + Memória  
Registrador + Registrador

```
A Reg1, memory_cell
```

```
AR Reg1, Reg2
```

```
Reg1 ← contents(Reg1) + contents(memory_cell)
```

```
Reg1 ← contents(Reg1) + contents(Reg2)
```

Registrador + Memória  
Registrador + Registrador

# Qual linguagem é melhor?

- **Ortogonalidade:** exemplos problemáticos em C:
  - Funções podem retornar struct mas não array
  - Qualquer tipo de dado pode ser membro de um struct, exceto void ou uma struct do mesmo tipo
  - Elementos de vetores podem ser de qualquer tipo, exceto void
  - Parâmetros são passados por valor, exceto de forem vetores
  - $A + B$ : se A for ponteiro para float e B for inteiro, o contexto de A afeta o significado de B



# Qual linguagem é melhor?

- **Tipos de dados:** adequados para a situação

```
timeOut = 1
```

```
timeOut = true
```

# Qual linguagem é melhor?

- **Sintaxe:** nem 8 nem 80!
  - Palavras especiais (if, while, for, ...)
  - Métodos para formar blocos: { }, begin/end, ...
  - Forma e significado: a aparência deve indicar o propósito
- **Abstração:** fundamental!
  - Esconde detalhes internos
  - Controle de complexidade

# Qual linguagem é melhor?

- **Expressividade:** abstrato, depende de experiência para reconhecer
  - Linguagens funcionais geralmente são reconhecidas como as de maior poder expressivo
- **Verificação de tipos:** testes para erros de tipo durante a compilação e/ou execução do programa
  - Desejável na compilação
  - Na execução tem alto custo

# Qual linguagem é melhor?

- **Tratamento de exceções:** habilidade do programa interceptar erros durante a execução, executar ações corretivas e então continuar a execução
- **Apelidos:** dois ou mais nomes diferentes para acessar a mesma célula de memória (por exemplo, uma variável). Cuidado!



# Qual linguagem é melhor?

- **Custo:** da linguagem em si e ocultos:

- Codificação
- Manutenção
- Confiabilidade
- Treinamento
- Compilação/Otimização
- Execução/Interpretação
- Implementação
- Portabilidade, Generabilidade, “Padronização”

Prices for Commercial Users in North America, Russia and rest of the world except Europe, India, China (PRC) and Taiwan	Professional Edition	Enterprise Edition
	per user	per user
LispWorks 8.0 (32-bit) for Windows	\$1,500	\$4,500
LispWorks 8.0 (32-bit) for Windows + 1 year <a href="#">maintenance</a>	\$1,700	\$5,100
LispWorks 8.0 (64-bit) for Windows	\$3,000	\$4,500
LispWorks 8.0 (64-bit) for Windows + 1 year <a href="#">maintenance</a>	\$3,400	\$5,100
LispWorks 8.0 (64-bit) for Macintosh	\$3,000	\$4,500
LispWorks 8.0 (64-bit) for Macintosh + 1 year <a href="#">maintenance</a>	\$3,400	\$5,100
LispWorks 8.0 (32-bit) for x86/x86_64 Linux	\$1,500	\$4,500
LispWorks 8.0 (32-bit) for x86/x86_64 Linux + 1 year <a href="#">maintenance</a>	\$1,700	\$5,100
LispWorks 8.0 (64-bit) for x86_64 Linux	\$3,000	\$4,500
LispWorks 8.0 (64-bit) for x86_64 Linux + 1 year <a href="#">maintenance</a>	\$3,400	\$5,100
LispWorks 8.0 (32-bit) for ARM Linux	\$1,500	\$4,500
LispWorks 8.0 (32-bit) for ARM Linux + 1 year <a href="#">maintenance</a>	\$1,700	\$5,100
LispWorks 8.0 (64-bit) for ARM Linux	\$3,000	\$4,500
LispWorks 8.0 (64-bit) for ARM Linux + 1 year <a href="#">maintenance</a>	\$3,400	\$5,100
LispWorks 8.0 (32-bit) for x86/x64 Solaris	\$1,500	\$4,500
LispWorks 8.0 (32-bit) for x86/x64 Solaris + 1 year <a href="#">maintenance</a>	\$1,700	\$5,100
LispWorks 8.0 (64-bit) for x86/x64 Solaris	\$3,000	\$4,500
LispWorks 8.0 (64-bit) for x86/x64 Solaris + 1 year <a href="#">maintenance</a>	\$3,400	\$5,100
LispWorks 8.0 (32-bit) for FreeBSD	\$1,500	\$4,500
LispWorks 8.0 (32-bit) for FreeBSD + 1 year <a href="#">maintenance</a>	\$1,700	\$5,100
LispWorks 8.0 (64-bit) for FreeBSD	\$3,000	\$4,500
LispWorks 8.0 (64-bit) for FreeBSD + 1 year <a href="#">maintenance</a>	\$3,400	\$5,100
<a href="#">Maintenance</a> renewal, per 32-bit license	\$375	\$1,125
<a href="#">Maintenance</a> renewal, per 64-bit license	\$750	\$1,125
<a href="#">Support</a> - 5 incidents pack	\$1,200	\$1,200
<a href="#">Support</a> - 10 incidents pack	\$2,100	\$2,100
<a href="#">Support</a> - 20 incidents pack	\$3,600	\$3,600

# Influências no projetos de LP

- **Arquitetura de computadores:** a arquitetura de **von Neumann** influenciou as LP nos últimos 80 anos! A maioria das LP projetadas são **imperativas**:

- **Variáveis**

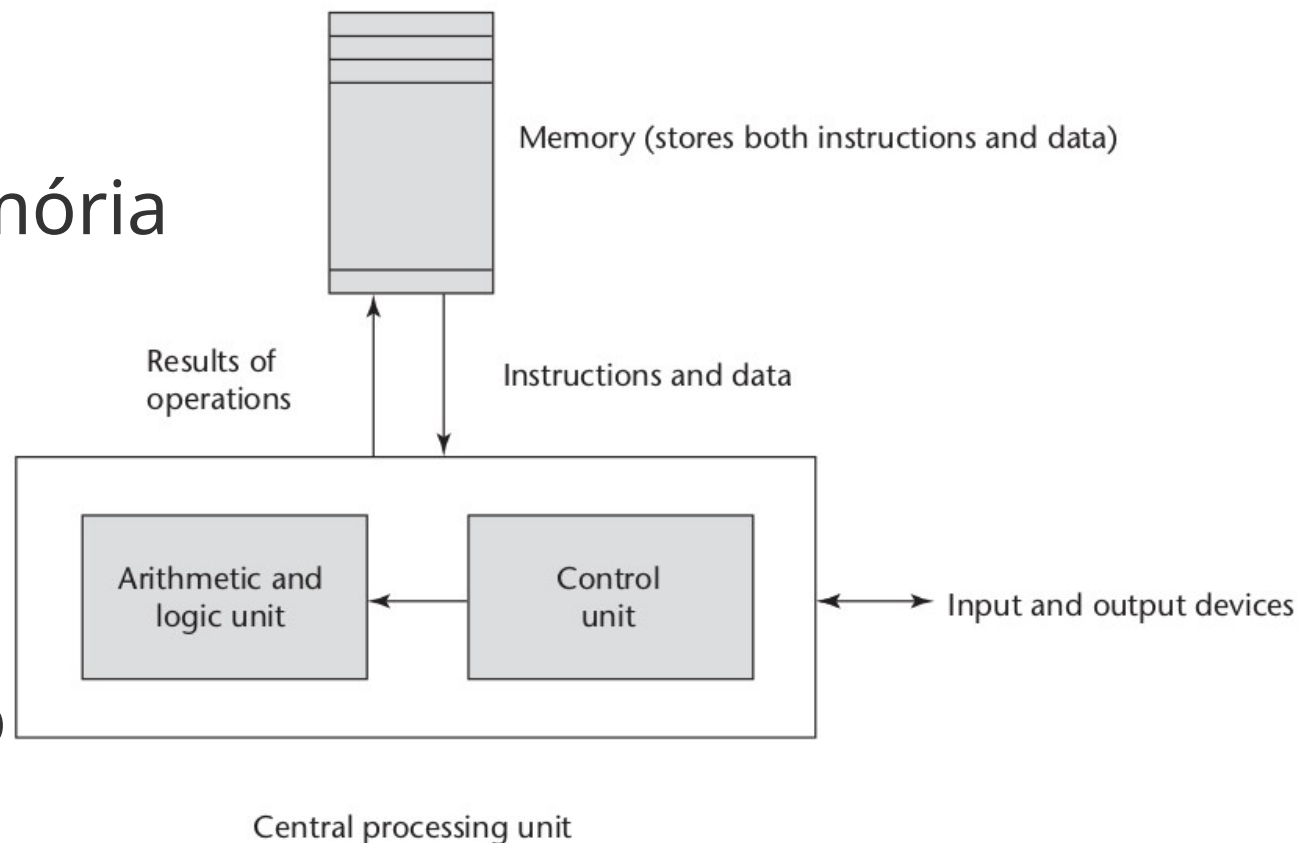
- Modelam memória

- **Atribuição**

- Modem pipe

- **Iteração**

- Para repetição



# Influências no projetos de LP

- **Execução em arquitetura de von Neumann:** ocorre em um ciclo de obtenção e execução, com um contador de programa que aponta para a próxima instrução.

initialize the program counter

**repeat** forever

    fetch the instruction pointed to by the program counter

    increment the program counter to point at the next instruction

    decode the instruction

    execute the instruction

**end repeat**

# Influências no projetos de LP

- **Metodologias de projeto de programas:**
  - Programação estruturada (top-down)
  - Orientação aos dados (modelagem de dados)
  - Orientação a objetos (classes, herança)
  - Outros...

# “Categorias” (paradigmas) de LP

- **Disivão arbitrária:** quatro grandes categorias com áreas de sobreposição e confusão.
  - **Imperativas** (ou procedurais)
  - **Funcionais**
  - **Lógicas**
  - **Orientadas a objetos**
- **Outras “categorias”:**
  - Scripting, Marcação
  - Híbridas (**multiparadigmas**)

Declarativas?

# Trade-offs no projeto de LP

- Não é possível ter tudo ao mesmo tempo: o que priorizar?





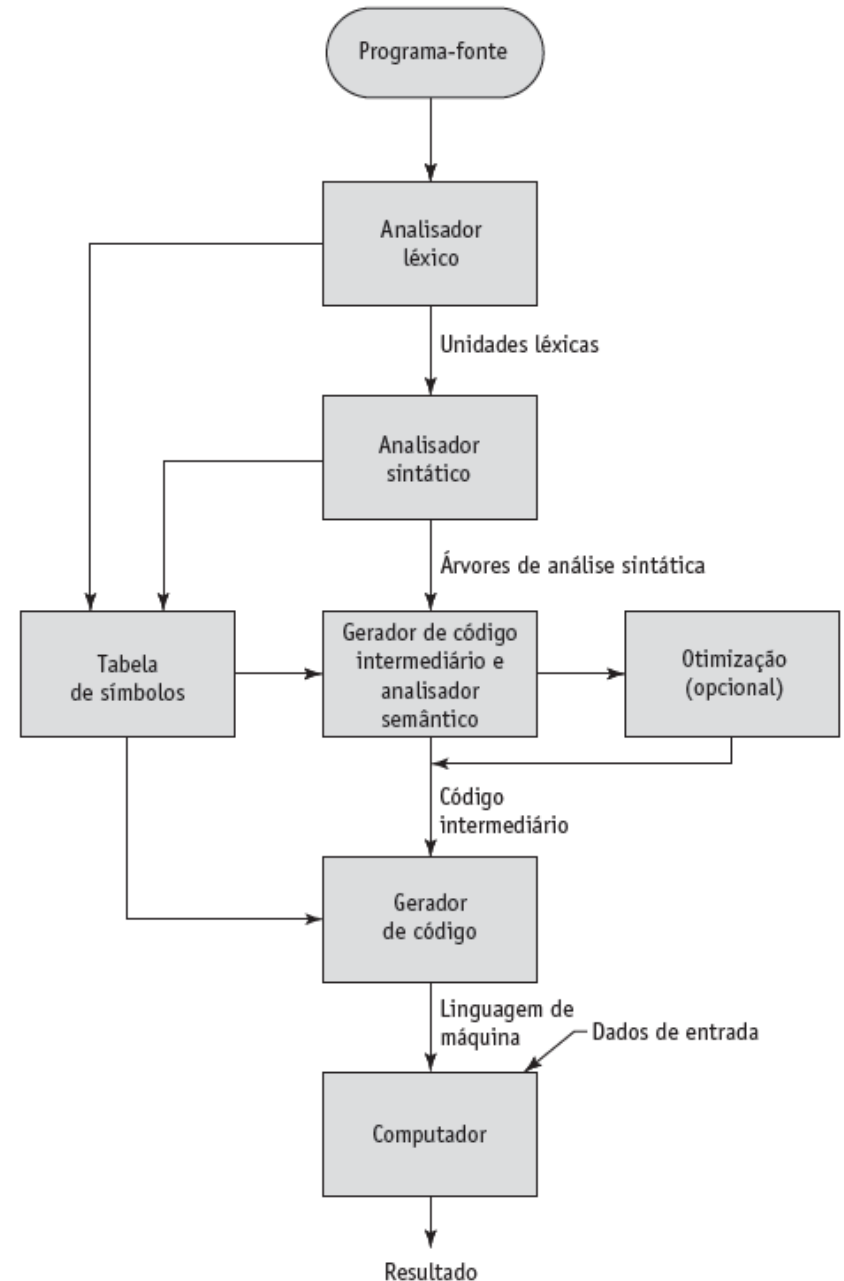


# Métodos de implementação

- **Três métodos gerais:**
  - Compilação
  - Interpretação
  - Híbrido

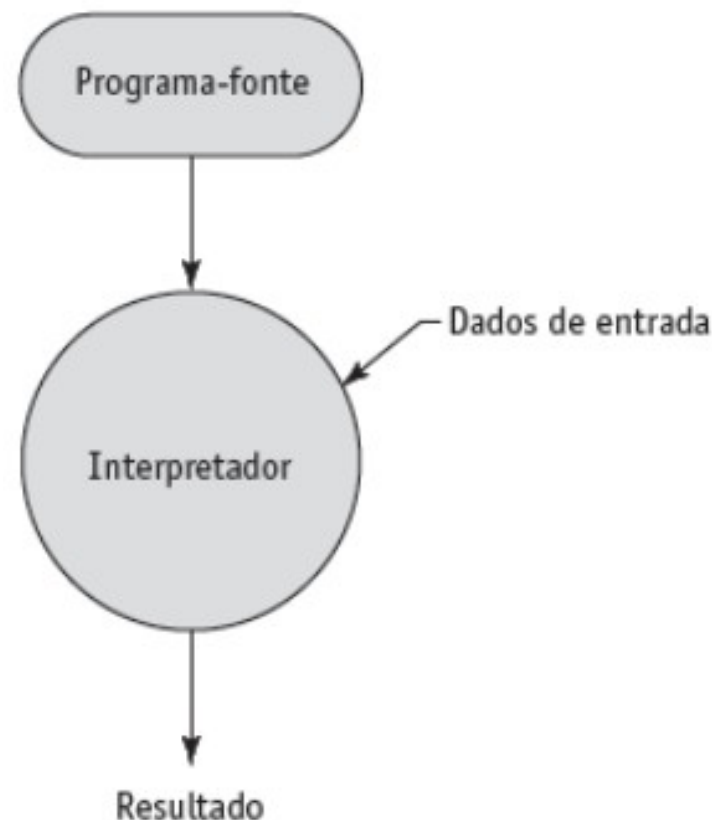
# Métodos de implementação

- **Compilação:** o programa em linguagem de alto nível é traduzido para linguagem de máquina. C, C++, ...
  - Rapidez de execução
  - Gargalo de von Neumann



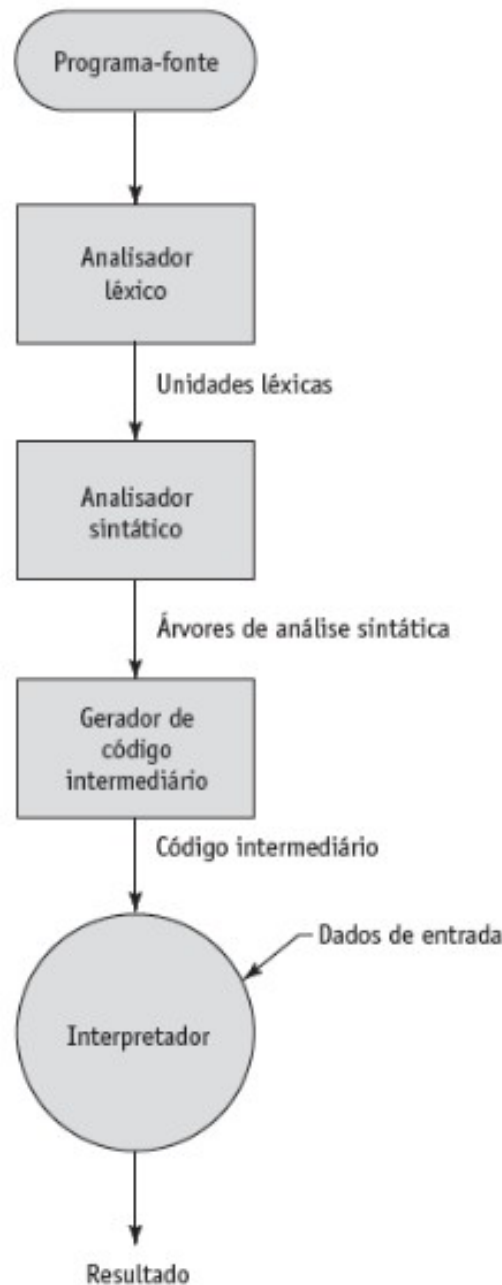
# Métodos de implementação

- **Interpretação:** o programa em linguagem de alto nível não é traduzido para linguagem de máquina, ele é interpretado por outro programa. Python, Lisp, R, JavaScript, PHP, ...
  - Mais lento
  - Mais espaço



# Métodos de implementação

- **Híbridos:** o programa em linguagem de alto nível é traduzido para uma linguagem intermediária projetada para facilitar a interpretação. Perl, Java, .NET, ...
  - Mais rápida que interpretação pura
  - Portabilidade
  - Just-in-Time



**LEIA o capítulo 1 do livro...  
O PSET está chegando!**

