# Paradigmas de Programação

## Linguagens de Programação

## Introdução

Este material contém notas de aula sobre *Linguagens de Programação*. O objetivo é apresentar características, classificações, métodos de implementação e recursos de linguagens de programação.

## Linguagem de Programação

Como já visto, linguagens de programação são notações utilizadas para instruir computadores sobre como realizar tarefas. Embora linguagens diferentes possuam características diferentes, existe um conjunto de características desejáveis a toda linguagem de programação.

### Características Desejáveis em Linguagens de Programação (Critérios de Avaliação)

Sebesta (2012) afirma que os principais critérios são:

- **Legibilidade**: facilidade com a qual os programas podem ser lidos e entendidos.
- **Facilidade de Escrita**: medida do quão facilmente uma linguagem pode ser usada para criar programas para um domínio.
- **Confiabilidade**: característica atingida quando o programa está de acordo com suas especificações em todas as condições.

As características mais importantes que influenciam os critérios citados estão na Tabela 1:

**Tabela 1**. Critérios de avaliação de linguagens e as características que os afetam (SEBESTA, 2011).

| | Critérios | | |
|---|---|---|---|
| **Características** | **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 | | | ✓ |

Como um quarto critério, Sebesta (2011) cita o **custo**, que não foi incluído na tabela "*porque é apenas superficialmente relacionado aos outros três critérios e às características que os influenciam*".

### Para pensar

Que outros aspectos podem ser considerados ao se avaliar linguagens de programação?

- Robustez?
- Tolerância a Falhas?
- Facilidade de Aprendizado?
- Portabilidade?
- Reusabilidade?

Algum desses aspectos está associado aos critérios da Tabela 1?

## Influência em Projeto de Linguagens de Programação

O que influencia o projeto de linguagens de programação? Considera-se *influência* aquilo que produz algum efeito, seja ele positivo ou negativo. Segundo Sebesta (2011), além dos aspectos citados anteriormente (presentes na Tabela 1), os mais importantes são:

- Arquitetura e Organização de Computadores (Modelo de von Neumann)
- Metodologias de Projeto de Programação

### Arquitetura de Computadores

A Arquitetura de von Neumann, cujo projeto está ilustrado na Figura 1, é uma arquitetura de computador que prevê a interação entre a unidade central de processamento com dispositivos de entrada e saída (E/S) e a memória, que armazena dados e instruções.

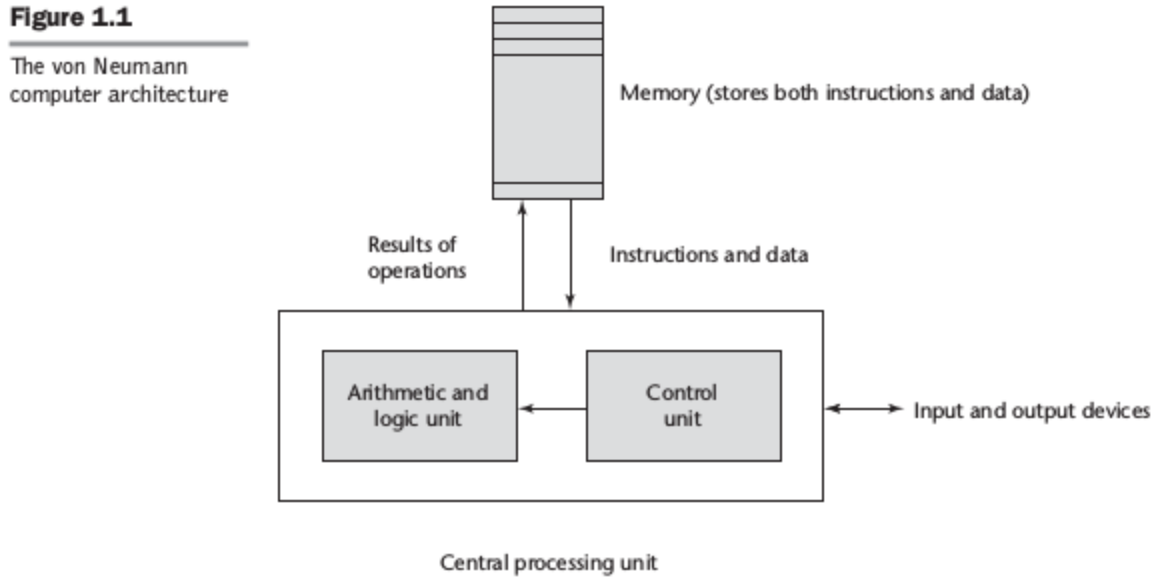


**Figura 1**. Arquitetura de von Neumann. Fonte: Sebesta (2012).

Como dito anteriormente, a arquitetura de computadores influencia o projeto de linguagens de programação. Essa influência se dá através de

- modelagem de células de memória por meio de variáveis;
- transmissões de dados, por meio das atribuições;
- repetições iterativas, porque as instruções ficam em posições contíguas de memória;

- dados e expressões transmitidos entre CPU e memória;
- ação denominada *ciclo de busca e execução*, carregando o programa da memória na CPU, instrução por instrução, armazenando o endereço de memória da próxima instrução e executando a instrução atual.

## Classificação de Linguagens de Programação

Sebesta (2012) menciona uma frequente classificação em quatro categorias de linguagens de programação:

- Imperativa
- Orientada a Objetos
- Funcional
- Lógica

> *Alguns autores se referem às linguagens de scripting como uma categoria separada de linguagens de programação. Entretanto, linguagens nessa categoria são mais unidas entre si por seu método de implementação, interpretação parcial ou completa, do que por um projeto de linguagem comum. As linguagens de scripting, dentre elas Perl, JavaScript e Ruby, são imperativas em todos os sentidos (SEBESTA, 2011).*

## Ferramentas

No contexto do estudo da disciplina de Paradigmas de Programação, o aprendizado de novas linguagens de programação requer a prática de programar. Para isso, recomenda-se a familiarização com algumas ferramentas, tais como:

- `clang`
- `DevC++`
- `gcc`
- `gdb`
- `ghci`
- `javac`
- `java`
- `nasm`

- `objdump`
- `prolog`
- `python`
- `sh, zsh, bash`

Outra possibilidade é utilizar serviços *online*, tais como:

- https://www.tutorialspoint.com/codingground.htm
- https://www.codechef.com/ide
- https://www.onlinegdb.com/
- https://www.jdoodle.com/
- https://repl.it/languages/

**Observação**: outras ferramentas podem ser necessárias.

## Métodos de implementação (compilação, interpretação, híbrido)

A linguagem de máquina do computador é o seu conjunto de instruções. O projeto da linguagem de máquina implementa em hardware uma linguagem de baixo nível que fornece as operações primitivas mais comumente necessárias (as mais básicas) e requer software de sistema para criar uma interface para programas em linguagens de nível superior. Essa interface é conhecida como Sistema Operacional.

O Sistema Operacional se responsabiliza por gerenciar recursos do sistema, operações de E/S, oferecer um sistema de arquivos, editores de texto e diversas outras ferramentas. Como os sistemas de implementação de linguagem precisam de muitos recursos do Sistema Operacional, eles se comunicam com o Sistema Operacional em vez de fazê-lo diretamente com o processador (em linguagem de máquina).

Tanto o Sistema Operacional quanto as implementações de linguagens de programação representam camadas de abstração que podem ser vistas, neste contexto, como computadores virtuais. Cada camada mais externa tem nível de abstração mais alto, como mostra a Figura 2 (Figura 1.2 em Sebesta, 2012).

**Figure 1.2**

Layered interface of virtual computers, provided by a typical computer system

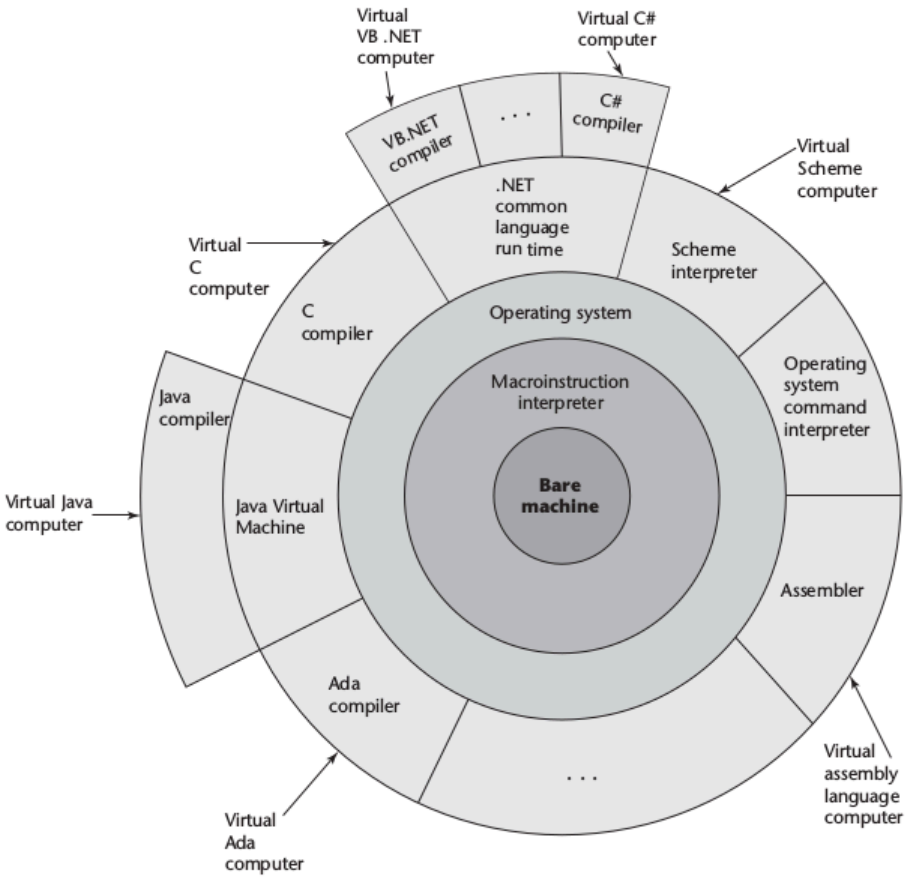


**Figura 2**. Interface em camadas de computadores virtuais, fornecida por um sistema de computador típico. Fonte: Sebesta (2012).

Cada sistema de implementação de linguagens de programação tem suas próprias características. Em geral os sistemas de implementação costumam ser classificados em ***compilados***, ***interpretados*** e ***híbridos***, explicados de forma resumida na sequência do texto.

**Observação importante**: embora seja costume classificar as linguagens desta forma, o correto é dizer que estas são propriedades das implementações. Muitas linguagens possuem somente uma forma de implementação, simplificando a discussão. No entanto, o correto é referir-se à implementação, e não à linguagem.

## Compilação

É o processo de tradução do código-fonte para a linguagem de máquina. Na compilação, se

traduz uma vez e um executável é gerado. Diz-se que para uma tradução pode-se realizar *n* execuções.

Em geral, o processo de compilação passa por otimização, o que costuma trazer melhorias de desempenho na execução e no uso de recursos.

O exemplo disponível no quadro a seguir (e no arquivo `aula2.c`) contém um código simples em linguagem C.

```
#include <stdio.h>
int main() {
      printf("Hello, world!\n");
      return 0;
}
```

Para compilar o código do arquivo `aula2.c`, isto é, traduzi-lo para a linguagem de máquina, usa-se um compilador. Para realizar esta ação usando o compilador `gcc`, basta abrir o terminal no diretório em que o arquivo `aula2.c` está e digitar:

```
gcc aula2.c
```

Ao digitar o código acima, o arquivo executável nomeado a.out será gerado no mesmo diretório, bastando digitar `./a.out` para executá-lo. Para definir um nome diferente para o arquivo executável, basta utilizar o parâmetro `-o`, como no exemplo a seguir:

```
gcc aula2.c -o aula2
```

Neste caso, o arquivo executável gerado se chama `aula2`, bastando digitar `./aula2` para executá-lo. Em ambos os casos, após a etapa de compilação, basta executar o arquivo (`a.out` ou `aula2`), não sendo necessário compilar novamente, a menos que o código-fonte seja novamente alterado.

Exemplos de linguagens com implementações compiladas:

- C
- C++

- C#
- Go
- Haskell
- LISP
- Rust

### Interpretação

É o processo de execução no qual a tradução do código-fonte ocorre instrução por instrução. Assim, na interpretação, traduz-se uma vez e executa uma vez.

Características da interpretação:

- É um processo mais flexível.
- A depuração é mais fácil.
- As mensagens de erro são mais precisas.
- Analisa o código mais rápido, mas executa mais lento.

Como exemplo, digite os comandos abaixo no terminal:

```
R
x <- c(1,500,98,800)
mean(x)
sd(x)
```

O primeiro comando servirá para iniciar a interpretação interativa da linguagem de programação R no terminal, permitindo que se interaja com o interpretador através do *console*. Depois, cada um dos três comandos seguintes é executado um a um pelo interpretador.

Sobre as mensagens de erro, é interessante analisar os exemplos a seguir. O código do quadro abaixo, em linguagem C, também está disponível no arquivo array-out-of-bounds.c:

```
#include <stdio.h>
int main() {
      short array[2];
      array[0] = 10;
```

```
    array[1] = 10;
    array[2] = 10;
    printf("%hd\n", array[2]);
    printf("%hd\n", array[3]);
    return 0;
}
```

O código do quadro abaixo, em linguagem Python, também está disponível no arquivo aula2.py. O código pode ser executado linha a linha na ferramenta de linha de comando `python` (`python3`):

```
tamanho = 2
array = [10] * tamanho
print array[0]
print array[1]
print array[2]
```

Os códigos apresentam erro(s)? Como as ferramentas trataram cada situação?

Exemplos de linguagens com implementações interpretadas:

- Haskell
- JavaScript
- Perl
- PHP
- Python
- R
- Ruby

### Modelos Híbridos

Esses modelos misturam vantagens da interpretação e da compilação. Eles podem fazer isso de várias maneiras, como compilando partes do código para melhorar o desempenho, enquanto outras partes são interpretadas para fornecer flexibilidade. Também podem usar

técnicas como a compilação *just in time* – JIT, onde o código é compilado sob demanda durante a execução, permitindo tanto a otimização quanto a flexibilidade. Essa abordagem híbrida é comumente encontrada em ferramentas como PyPy (python), máquinas virtuais JVM (Java) e CLR (C#), e ambientes de desenvolvimento como o .NET Core e o Node.js.

Na abordagem de compilação JIT, o código é compilado para uma forma intermediária não dependente de arquitetura, chamada de *código de bytes* (ou *byte code*). Por não ser dependente de arquitetura, esta representação intermediária fornece portabilidade para qualquer máquina que possua um interpretador de *byte codes* e um sistema de tempo de execução associado (SEBESTA, 2012). No caso da linguagem Java, juntos eles são chamados de Java Virtual Machine. Existem agora sistemas que traduzem código de bytes Java em código de máquina para uma execução mais rápida (SEBESTA, 2012).

A máquina virtual possui um interpretador, que traduz o código instrução por instrução. O compilador JIT só traduz o *byte code* interpretado, realizando otimizações no código e o transformando-o em código de máquina dependente de arquitetura.

Exemplos de linguagens com implementações híbridas:

- C#
- Java
- JavaScript
- Python
- Ruby

## Recursos e Características de Linguagens de Programação

Diversas linguagens diferentes podem resolver os mesmos problemas, mas oferecendo e trabalhando os recursos de formas diferentes. Por exemplo, em programas escritos em C, o gerenciamento de memória é responsabilidade do programador, enquanto programas escritos em C# executam sobre uma máquina virtual que é capaz de gerenciar memória.

A lista abaixo traz alguns recursos disponíveis em linguagens de programação. Alguns dos itens da lista que contêm link para um diretório com código(s) que exemplifica(m) o uso do

recurso.

- Coletor de lixo (Código em Java)
    - Analise o código e compile o arquivo `Coletor.java` com o comando: `javac Coletor.java`
    - Execute o código com o comando: `java Coletor`
- Goto/jump (Código em C)
    - Analise o código e compile o arquivo `goto.c` com o comando: `gcc goto.c -o goto`
    - Execute o código com o comando: `./goto`
- *Loops* rotulados (*labeled loops*) (Códigos em Java)
    - Analise os códigos e compile os arquivos com o comando: `javac ComLabeledLoop.java Exemplo.java SemLabeledLoop.java`
    - Execute um a um os arquivos `.class` com o comando `java nome-do-arquivo`
    - **Observação**: em linguagens estruturadas, o uso de *loops* rotulados é **altamente desincentivado**.
- Recursividade (Códigos em C, Haskell)
    - (código de exemplo)
- Regex (Códigos em Python e HTML[1])
    - Analise e execute o código do arquivo `regex.py` com o comando: `python3 regex.py`
    - Analise o código e abra o arquivo `regex.html` em um navegador web.
- Ponteiros/Referências
- Polimorfismo
- Herança
- Classe
- Interface
- Inferência de tipos
- Compreensão de Lista/*List comprehension*

[1] HTML é uma linguagem de marcação, e não de programação! O exemplo serve apenas para exemplificar o recurso, que também está disponível em diversas linguagens de programação, tais como Java, JavaScript, PHP e Python.

- Expressões Lambda
- Paralelismo e Concorrência
- Mônadas

**Observação**: os recursos citados serão abordados no decorrer dessa e de outras disciplinas, tais como *Programação Lógica e Funcional* e *Programação de Alto Desempenho*.

# Conclusão

Este texto fornece uma visão geral sobre linguagens de programação, abordando características, critérios de avaliação e influências de arquitetura de computadores e metodologias de projeto de linguagens de programação. Foram apresentados conceitos fundamentais sobre linguagens de programação, destacando sua função como instruções para computadores e discutindo critérios de avaliação como legibilidade, facilidade de escrita e confiabilidade, bem como características que afetam esses critérios.

A influência da arquitetura de computadores no projeto de linguagens de programação foi examinada, destacando a arquitetura de von Neumann e sua interação entre a unidade central de processamento, dispositivos de E/S e memória. O texto aborda como a modelagem de memória, transmissões de dados, repetições iterativas e outras operações são influenciadas pela arquitetura de computadores.

Uma discussão sobre a classificação de linguagens de programação é apresentada, com ênfase em categorias como imperativa, orientada a objetos, funcional e lógica, além da inclusão de linguagens de *scripting* como uma categoria separada. Essa discussão é fundamental para permitir o avanço no estudo sobre paradigmas de programação.

Foram mencionados os métodos de implementação de linguagens de programação: compilação, interpretação e híbrido, apresentando vantagens, desvantagens e exemplos. O texto também abordou recursos e características de linguagens de programação, oferecendo exemplos práticos sobre alguns recursos como coletor de lixo, *loops* rotulados e recursividade.

# Tarefas

1) Resolva os exercícios do capítulo 1 do livro *Concepts of programming languages*, de Robert W. Sebesta.
2) Pesquise sobre os recursos citados na Seção *Recursos e Características de Linguagens de Programação* e elabore exemplos de códigos que os utilizem.

# Referências

SEBESTA, Robert W. Concepts of programming languages. Pearson. 10th ed. 2012.

SEBESTA, Robert W. Concepts of programming languages. Bookman. 9ª ed. 2011.