



# Modelos de Linguagem de Programação

## Unidade 3 – Classificação e Paradigmas de Linguagens de Programação



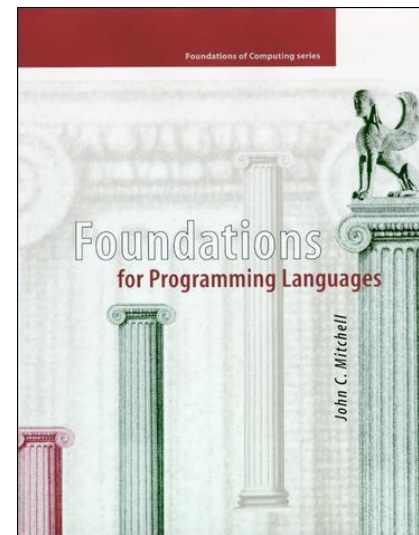
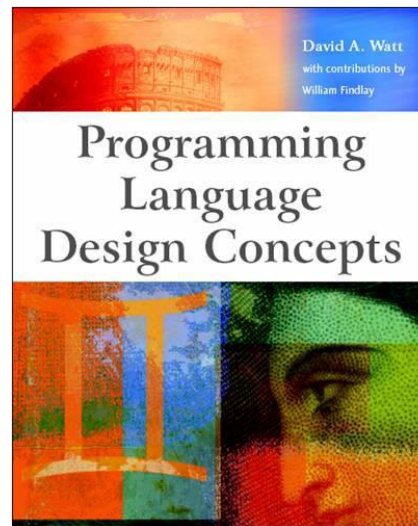
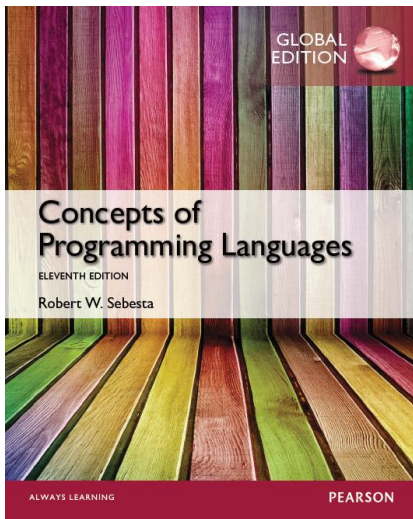
Prof. Aparecido V. de Freitas  
Doutor em Engenharia  
da Computação pela EPUSP  
[aparecidovfreitas@gmail.com](mailto:aparecidovfreitas@gmail.com)





# Bibliografia

- Sebesta, Robert W. Concepts of Programming Languages – Eleventh Edition
- Watt, D. Programming Language Design Concepts. John Wiley and Sons, 2004.
- Mitchell, J. Foundations for Programming Languages, MIT Press, 1996.






# Classificação das Linguagens: Nível

## 1. Baixo Nível:

- As linguagens de Baixo Nível são aquelas **voltadas para a máquina**, ou seja as que são escritas utilizando as instruções do microprocessador do computador.
- São genericamente chamadas de linguagens **Assembly**. Os programas escritos com Alto Nível geralmente podem ser convertidos com programas especiais para Baixo Nível.



# Exemplo – Assembly



**codingground**  
SIMPLY EASY CODING

COMPILE AND EXECUTE ASSEMBLY ONLINE

New Project-20170815 [Save] [Undo] [Redo] [Close] [Back]

root  
main.asm

Compile | Execute | Share Code main.asm x

```

1 section .text
2     global _start           ;must be declared for using gcc
3     _start:                 ;tell linker entry point
4     mov edx, len            ;message length
5     mov ecx, msg            ;message to write
6     mov ebx, 1              ;file descriptor (stdout)
7     mov eax, 4              ;system call number (sys_write)
8     int 0x80                ;call kernel
9     mov eax, 1              ;system call number (sys_exit)
10    int 0x80                ;call kernel
11
12    section .data
13
14    msg db 'Hello, world!',0xa ;our dear string
15    len equ $ - msg          ;length of our dear string
16

```





# Classificação das Linguagens: Nível

## 1. Baixo Nível:

### – Vantagens:

- Os programas são executados com maior **velocidade de processamento**;
- Os programas ocupam **menos espaço na memória**;

### – Desvantagens:

- Em geral, programas em Assembly **tem pouca portabilidade**, isto é, um código gerado para um tipo de processador não serve para outro;
- Códigos Assembly não são estruturados, tornando a **programação mais difícil...**

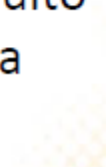




# Classificação das Linguagens: Nível

## 2. Médio Nível:

- São linguagens voltadas ao **ser humano e a máquina**;
- Estas linguagens são uma mistura entre as linguagens de Alto Nível e as de Baixo Nível;
- Estas linguagens de programação contêm comandos muito simples e outros mais complicados, o que pode tornar a programação um pouco "complicada".



# Classificação das Linguagens: Nível

## 2. Médio Nível:

- **Exemplo – Linguagem C:** pode-se acessar registros do sistema e trabalhar com endereços de memória (características de linguagens de baixo nível) e ao mesmo tempo realizar operações de alto nível (if...else; while; for).

```
int x, y, *p;  
y = 0;  
p = &y;  
x = *p;  
x = 4;  
(*p)++;  
x--;  
(*p) += x;
```

```
int vet[6] = {1, 2, 3, 4, 5};  
  
printf("%d\n", vet);  
printf("%d\n", *vet);  
printf("%d\n", *(vet + 2));
```







# Classificação das Linguagens: Nível

## 2. Médio Nível:

### – Vantagens:

- Geralmente são linguagens poderosas, permitindo a criação de diversos softwares, desde jogos a programas de alta performance.

### – Desvantagens:

- Alguns comandos têm uma sintaxe um pouco difícil de compreender.







# Classificação das Linguagens: Nível

## 3. Alto Nível:

- São linguagens voltadas para o **ser humano**. Em geral utilizam sintaxe mais estruturada, tornando o seu código **mais fácil de entender**.
- São linguagens independentes de arquitetura.
  - Um programa escrito em uma linguagem de alto nível, pode ser migrado de uma máquina a outra sem nenhum tipo de problema.
- Permitem ao programador se esquecer completamente do funcionamento interno da máquina.
  - Sendo necessário um tradutor que entenda o código fonte e as características da máquina.





# Classificação das Linguagens: Nível

## 3. Alto Nível:

– Exemplos: Lua, Java, C#, C++...

```
nota = io.read()

if nota < 3.0 then
    io.write("Reprovado")
elseif nota >= 5.0 then
    io.write("Aprovado")
else
    io.write("Prova final")
end
```

```
Scanner entrada = new Scanner(System.in);
mes = entrada.nextInt();
switch (mes)
{
    case 1: System.out.println("Janeiro");
            break;
    case 2: System.out.println("Fevereiro");
            break;
    case 3: System.out.println("Marco");
            break;
    default:
        System.out.println("Outro");
        break;
}
```





# Classificação das Linguagens: Nível

## 3. Alto Nível:

### – Vantagens:

- Por serem compiladas ou interpretadas, têm maior **portabilidade**, podendo ser executados em várias plataformas com pouquíssimas modificações.
- Em geral, a programação é **mais fácil**.

### – Desvantagens:

- Em geral, as rotinas geradas (em linguagem de máquina) são mais genéricas e, portanto, mais complexas e por isso são **mais lentas e ocupam mais memória**.





# Classificação das Linguagens: Geração

## 1ª Geração: linguagens em nível de máquina:

- Os primeiros computadores eram programados em linguagem de máquina, em notação binária.
- Exemplo:

```
0010 0001 0110 1100
```

- Realiza a soma (código de operação 0010) do dado armazenado no registrador 0001, com o dado armazenado na posição de memória 108 (0110 1100)





# Classificação das Linguagens: Geração

## **1ª Geração: linguagens em nível de máquina:**

- Cada instrução de máquina é, em geral, formada por um código de operação e um ou dois endereços de registradores ou de memória;
- As linguagens de máquina permitem a comunicação direta com o computador em termos de "bits", registradores e operações de máquina bastante primitivas;
- Um programa em linguagem de máquina nada mais é que uma sequência de zeros e uns, a programação de um algoritmo complexo usando esse tipo de linguagem é complexa, cansativa e fortemente sujeita a erros.





# Classificação das Linguagens: Geração

## 2ª Geração: linguagens de montagem (Assembly):

- Compreende as linguagens simbólicas ou de montagem (Assembly), projetadas para **minimizar as dificuldades** da programação em **notação binária**.
- Códigos de operação e endereços binários foram substituídos por mnemônicos (abreviações).

mov mul add label goto





# Classificação das Linguagens: Geração

## 2ª Geração: linguagens de montagem (Assembly):

– Exemplo de tradução de IF para Assembly:

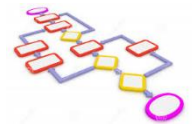
```
if (a == b)
{
    c = d;
}
d = a + c;
```



```
_start:
    cmp eax, ebx
    jne .L7
    mov edx, ecx
.L7:
    mov eax, edx
    add ecx, edx
```







# Classificação das Linguagens: Geração

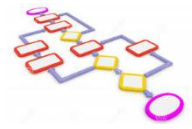
## 2ª Geração: linguagens de montagem (Assembly):

- Códigos de operação e endereços binários foram substituídos por abreviações. Assim, a instrução de máquina 0010 0001 0110 1100 evoluiu para:

```
add r1 total
```

- R1 representa o registrador 1 e TOTAL é o nome atribuído ao endereço de memória 108.
- Nas linguagens de montagem, a maioria das instruções são representações simbólicas de instruções de máquina. Porém, os programas requerem tradução para linguagem de máquina.

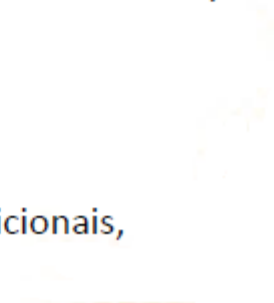




# Classificação das Linguagens: Geração

## 3ª Geração: linguagens orientadas ao usuário:

- Surgiram na década de 60.
- Algumas delas são orientadas à solução de problemas científicos, já outras são usadas para aplicações comerciais.
- As instruções oferecidas por essas linguagens pertencem, em geral, a três classes:
  - Instruções entrada/saída;
  - Instruções de cálculos aritméticos ou lógicos;
  - Instruções de controle de fluxo de execução (desvios condicionais, incondicionais e processamento iterativo);





# Classificação das Linguagens: Geração

## 3ª Geração: linguagens orientadas ao usuário:

- Exemplos de linguagens orientadas ao usuário: BASIC, ALGOL, PL/I, PASCAL, ADA, C, etc.

- **Exemplo: BASIC**

```
VAR number = 8

IF number < 0 THEN
    PRINT "Number is negative"
ELSEIF number > 0 THEN
    PRINT "Number is positive"
ELSE
    PRINT "Number is zero"
END IF
```





# Classificação das Linguagens: Geração

## 3ª Geração: linguagens orientadas ao usuário:

- Exemplos de linguagens orientadas ao usuário: BASIC, ALGOL, PL/I, PASCAL, ADA, C, etc.
- **Exemplo: PASCAL**

```
program teste;  
var  
  i : byte;  
begin  
  writeln('Digite um numero = ');  
  readln(i);  
  if i <= 10 then  
    writeln('É menor ou igual a 10!')  
  else  
    writeln('É maior que 10!');  
end.
```



- Nesta geração surgiram também linguagens declarativas, as quais dividem-se, basicamente, em duas classes:
  - **Funcionais:** as quais se baseiam na teoria das funções recursivas. Exemplo: LISP;
  - **Lógicas:** cuja base é a lógica matemática. Exemplo: Prolog.
- As linguagens dessa geração requerem um **tradutor** para transformar os seus comandos em linguagem de máquina.
  - **Compiladores;**
  - **Interpretadores.**





# Classificação das Linguagens: Geração

## **4ª Geração: linguagens orientadas à aplicação:**

- As linguagens de 3ª geração foram projetadas para programadores experientes e não para usuários finais.
- A depuração de programas escritos nessas linguagens consome tempo, e a modificação de sistemas complexos é relativamente difícil.
- As linguagens de 4ª geração foram projetadas em resposta a esses problemas.





# Classificação das Linguagens: Geração

## 4ª Geração: linguagens orientadas à aplicação:

- Os programas escritos em linguagens de 4ª geração necessitam de menor número de linhas de código do que os programas correspondentes codificados em linguagens de programação convencionais.
- Exemplo: ler e exibir uma imagem em MATLAB:

```
A = imread('image.jpg');  
image(A);
```







# Classificação das Linguagens: Geração

## 4ª Geração: linguagens orientadas à aplicação:

– Exemplo: ler uma imagem em C:

```
BYTE* LoadBMP(int* width, int* height, long* size, LPCTSTR bmpfile)
{
    BITMAPFILEHEADER bmpheader;
    BITMAPINFOHEADER bmpinfo;
    DWORD bytesread;
    HANDLE file = CreateFile (bmpfile , GENERIC_READ, FILE_SHARE_READ,
                             NULL, OPEN_EXISTING, FILE_FLAG_SEQUENTIAL_SCAN, NULL);
    if (NULL == file )
        return NULL;
    ...
    *size = bmpheader.bfSize - bmpheader.bfOffBits;
    BYTE* Buffer = new BYTE[ *size ];
    SetFilePointer(file, bmpheader.bfOffBits, NULL, FILE_BEGIN);
    if ( ReadFile(file, Buffer, *size, &bytesread, NULL ) == false)
    {
        ...
    }
}
```





# Classificação das Linguagens: Geração

## **4ª Geração: linguagens orientadas à aplicação:**

- As linguagens de 4ª geração variam bastante no número de facilidades oferecidas ao usuário.
- Algumas são, meramente, geradores de relatórios ou pacotes gráficos. Outras são capazes de gerar aplicações completas.
- Em geral, essas linguagens são projetadas para atender a classes específicas de aplicações.





# Classificação das Linguagens: Geração

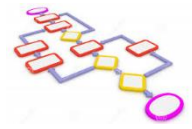
## 4ª Geração: linguagens orientadas à aplicação:

### – Principais objetivos:

- Facilitar a programação de computadores de tal maneira que usuários finais possam resolver seus problemas;
- Apressar o processo de desenvolvimento de aplicações;
- Facilitar e reduzir o custo de manutenção de aplicações;
- Minimizar problemas de depuração;
- Gerar código sem erros a partir de requisitos de expressões de alto nível.

### – Exemplos de linguagens de 4ª geração: SQL, Oracle Reports, MATLAB, PowerBuilder, Scilab, Strata, ...





# Classificação das Linguagens: Paradigma

- Paradigma é um **modelo interpretativo** (ou conceitualização) de uma **realidade**.
- Permite organizar as ideias com vista:
  - Ao entendimento dessa realidade;
  - À determinação de qual a melhor forma de atuar sobre essa realidade.
- Pode dizer-se que um paradigma é um **ponto de vista**: um ponto de vista que determina como uma realidade é entendida e como se atua sobre ela.





# Classificação das Linguagens: Paradigma

- Algumas linguagens criadas durante a história introduziram **novas formas de se pensar sobre programação**, resultando em formas distintas de modelagem de soluções de software.
  - FORTRAN (imperativa);
  - LISP (funcional);
  - Simula (orientadas a objetos);
  - Prolog (lógica).
- Outras linguagens são o resultado da evolução de linguagens mais antigas, muitas vezes **mesclando** características de diferentes linguagens existentes.
  - Por exemplo, C++ é uma evolução do C com características de orientação a objetos, importadas de Simula.

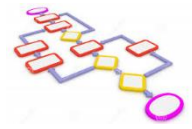




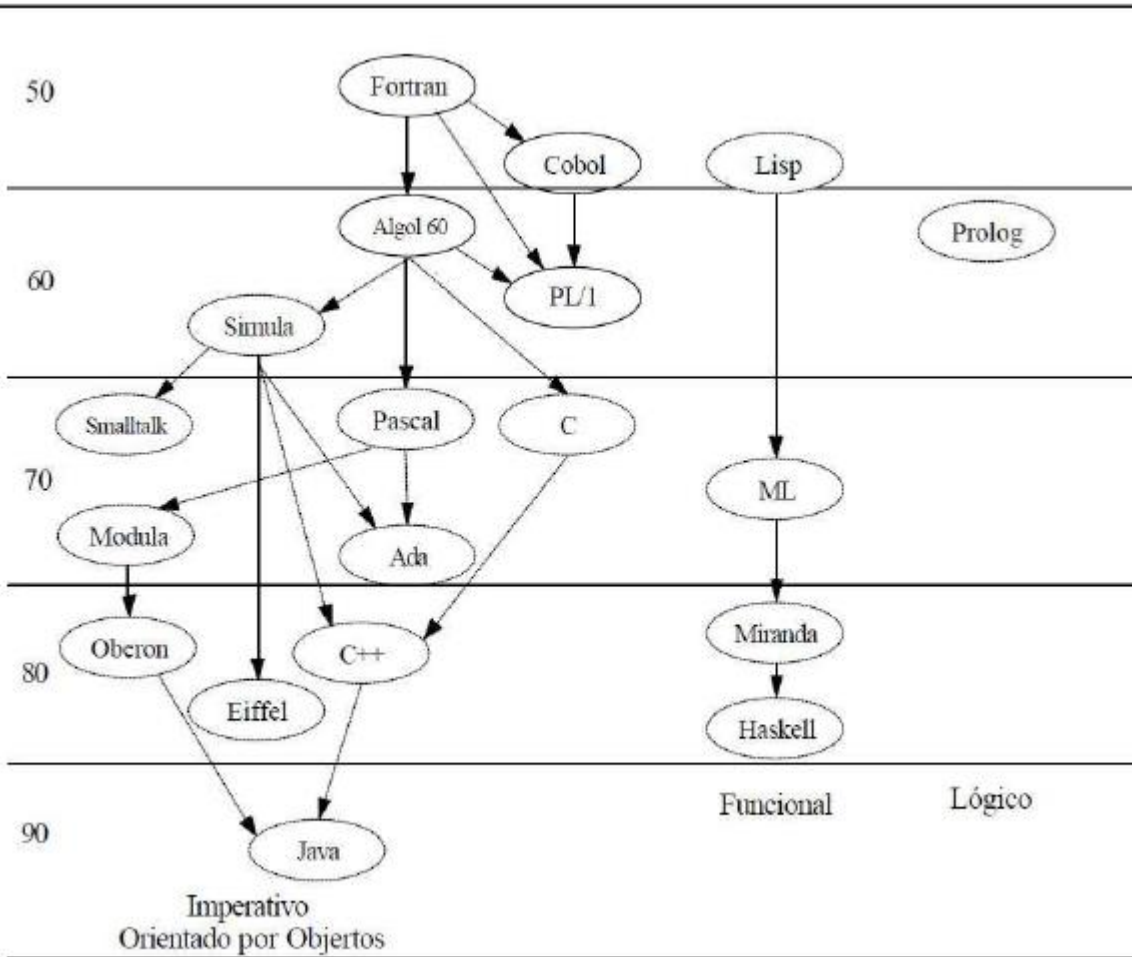
# Classificação das Linguagens: Paradigma

- **Paradigma imperativo** (sequência, atribuição, estado): Basic, Pascal, C, Ada, Fortran, Cobol, Assembly...
- **Paradigma funcional** (função, aplicação, avaliação): Lisp, Haskell, Erlang, Scheme...
- **Paradigma lógico** (relação, dedução): Prolog.
- **Paradigma orientado a objetos** (objeto, estado, mensagem): C++, Java, C#, Eiffel, Smalltalk, Python...
- **Paradigma concorrente** (processo, comunicação (síncrona ou assíncrona)): C++, C#, Java...





# Classificação das Linguagens: Paradigma







# Classificação das Linguagens: Paradigma

## 1. Paradigma Imperativo:

- As linguagens imperativas são orientadas a **ações**, onde a computação é vista como uma **sequência de instruções** que manipulam valores de **variáveis** (leitura e atribuição).
- Os programas são centrados no conceito de um **estado** (modelado por variáveis) e **ações** (comandos) que manipulam o estado.
- Paradigma também denominado de **procedural**, por incluir subrotinas ou procedimentos como mecanismo de estruturação.



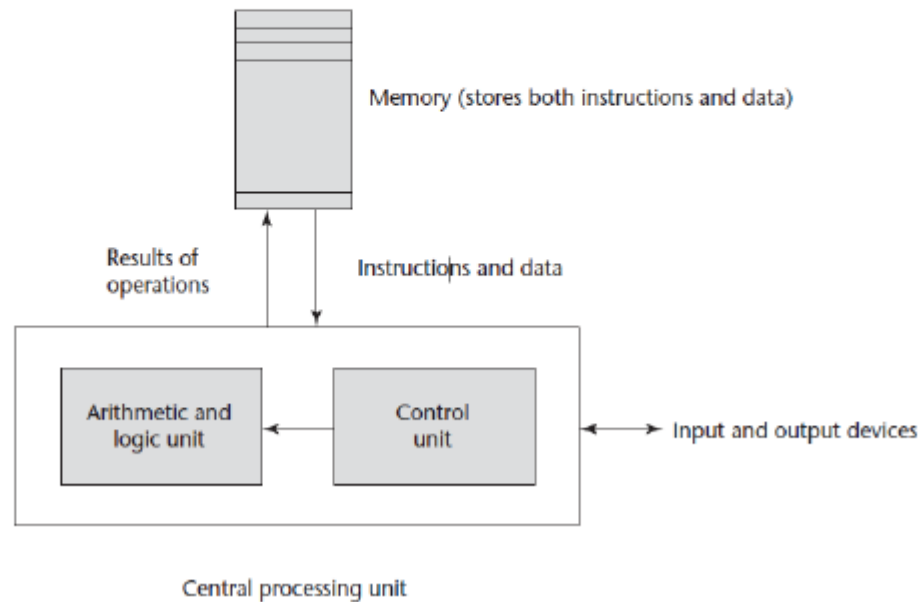


# Classificação das Linguagens: Paradigma

## 1. Paradigma Imperativo:

– Baseia-se na arquitetura de computadores Von Neumann:

- Programas e dados são armazenados na mesma memória;
- Instruções e dados são transmitidos da CPU para a memória, e vice-versa;
- Resultados das operações executadas na CPU são retornadas para a memória.





# Classificação das Linguagens: Paradigma

## 1. Paradigma Imperativo:

- Subdivide-se em **estruturado** e **não-estruturado**.
- Linguagens não-estruturadas geralmente fazem uso de comandos goto ou jump. Exemplos – Assembly e Basic:

```
_start:
    cmp eax, ebx
    jne .L7
    mov edx, ecx
.L7:
    mov eax, edx
    add ecx, edx
```

```
10 PRINT "Hello"
20 GOTO 50
30 PRINT "This text will not be printed"
40 END
50 PRINT "Goodbye"
```



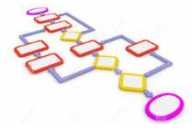


# Classificação das Linguagens: Paradigma

## 1. Paradigma Imperativo:

- As linguagens estruturadas surgiram objetivando facilitar a leitura e execução de algoritmos – **não fazem o uso do goto.**
  - Instruções são **agrupadas em blocos**, os quais podem ser considerados como unidades do programa;
- Blocos de instruções podem ser selecionados para execução através de declarações de seleção como if ... else, ou repetidamente executados através de declarações de repetição (for, while...).





# Classificação das Linguagens: Paradigma

## 1. Paradigma Imperativo:

– Exemplo de linguagem estruturada – C:

```
int busca(int n, int *vet, int elem)
{
    int i;
    for (i = 0; i < n; i++)
    {
        if (elem == vet[i])
        {
            return i;
        }
    }
    return -1;
}
```





# Classificação das Linguagens: Paradigma

## 1. Paradigma Imperativo:

- Linguagens estruturadas permitem a criação de procedimentos (**funções**);
- **Procedimentos criam um nível de abstração**, onde não é necessário conhecer todos os passos de execução de um procedimento, apenas qual sua função e quais os pré-requisitos para sua execução correta;
- **Linguagens estruturadas modulares** criam um outro mecanismo de abstração – módulo: composto de definições de variáveis e procedimentos, agrupados de acordo com critérios específicos.





# Classificação das Linguagens: Paradigma

## 1. Paradigma Imperativo:

### – Exemplos de Linguagens Imperativas:

- FORTRAN
- BASIC
- COBOL
- Pascal
- C
- ALGOL
- Modula
- ...







# Classificação das Linguagens: Paradigma

## 1. Paradigma Imperativo:

### – Vantagens:

- Eficiência;
- Modelagem "natural" de aplicações do mundo real;
- Paradigma dominante e bem estabelecido;

### – Desvantagens:

- Possui difícil legibilidade e facilita introdução de erros em sua manutenção;
- Descrições demasiadamente profissional focaliza o "como" e não o "quê";
- Tende a gerar códigos confusos, onde tratamento dos dados são misturados com o comportamento do programa;





# Classificação das Linguagens: Paradigma

## 2. Paradigma Funcional:

- Trata a computação como um processo de avaliação de **funções matemáticas**, evitando o uso de estados ou dados mutáveis;
- Enfatiza a **aplicação de funções**, em contraste da programação imperativa, que enfatiza mudanças no estado do programa;
- A visão funcional resulta num programa que descreve as operações que devem ser efetuadas para resolver o problema.





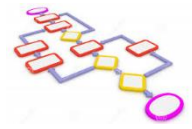
# Classificação das Linguagens: Paradigma

## 2. Paradigma Funcional:

- Programar em uma linguagem funcional consiste em pensar qual função deve ser aplicada para transformar uma entrada qualquer na saída desejada.
- Ao invés dos passos sucessivos do paradigma imperativo, a sintaxe da linguagem é apropriada para definição de funções compostas que denotam aplicações sucessivas de funções:

```
função (... função2 (função1 (dados)) ...)
```





# Classificação das Linguagens: Paradigma

## 2. Paradigma Funcional:

– **Exemplo:** Distância entre dois pontos em C

```
#include <stdio.h>
#include <math.h>
float dist(float PX1, float PY1, float PX2, float PY2)
{
    float res = sqrt((PX2 - PX1)*(PX2 - PX1) +
                     (PY2 - PY1)*(PY2 - PY1));
    return res;
}
int main()
{
    float f = dist(2.0, 4.0, 3.0, 1.0);
    printf("Distance = %f", f);
    return 0;
}
```





# Classificação das Linguagens: Paradigma

## 2. Paradigma Funcional:

– **Exemplo:** Distância entre dois pontos em Haskell

```
dist x1 y1 x2 y2 = sqrt(((x2 - x1)^2) + ((y2 - y1)^2))  
main = print(dist 2.0 4.0 3.0 1.0)
```

– Características da programação funcional:

- Programas são funções que descrevem uma relação explícita e precisa entre E/S;
- Estilo declarativo: não há o conceito de estado nem comandos como atribuição;





# Classificação das Linguagens: Paradigma

## 2. Paradigma Funcional:

### – Exemplos de Linguagens Funcionais:

- Haskell;
- Scheme;
- Common LISP;
- CLOS (Common LISP Object System);
- Miranda;
- ML;
- Erlang;
- Ocaml;
- ...





# Classificação das Linguagens: Paradigma

## 2. Paradigma Funcional:

### – Vantagens:

- Simplifica a resolução de alguns tipos problemas:
  - Prova de propriedades;
  - Resolução de programas de otimização;

### – Desvantagens:

- Problema: o mundo não é funcional!
- Implementações ineficientes;
- Mecanismos primitivos de E/S;



- Paradigma de programação baseado em **lógica formal**;
- Um programa lógico é equivalente à descrição do problema expressa de maneira formal, similar à maneira que o ser humano raciocinaria sobre ele;
- A programação lógica consiste em declarar **fatos**, que podem ser **relações** (associações) ou **regras** que produzem fatos deduzidos a partir de outros.







# Classificação das Linguagens: Paradigma

## 3. Paradigma Lógico:

- Programação em linguagens lógicas requer um **estilo mais descritivo**;
- O programador deve conhecer os **relacionamentos** entre as **entidades** e **conceitos** envolvidos para descrever os fatos relacionados ao problema;
- Programas descrevem um **conjunto de regras** que disparam ações quando suas premissas são satisfeitas;
- Principal linguagem lógica: **Prolog**





# Classificação das Linguagens: Paradigma

## 3. Paradigma Lógico:

– Exemplo Prolog:

```
tropical(caribe).  
tropical(havai).  
praia(caribe).  
praia(havai).  
bonito(havai).  
bonito(caribe).  
paraíso_tropical(X) :- tropical(X), praia(X), bonito(X).  
  
?- paraíso_tropical(X).
```



- Exemplo Prolog:

```
?- avo(X, paulo).
```





# Classificação das Linguagens: Paradigma

## 3. Paradigma Lógico:

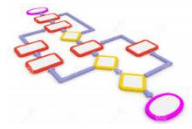
### – Vantagens:

- Permite a concepção da aplicação em alto nível de abstração;
- Linguagem mais próxima do raciocínio humano;

### – Desvantagens:

- Dificuldade em expressar algoritmos complexos;
- Complexidade exponencial;

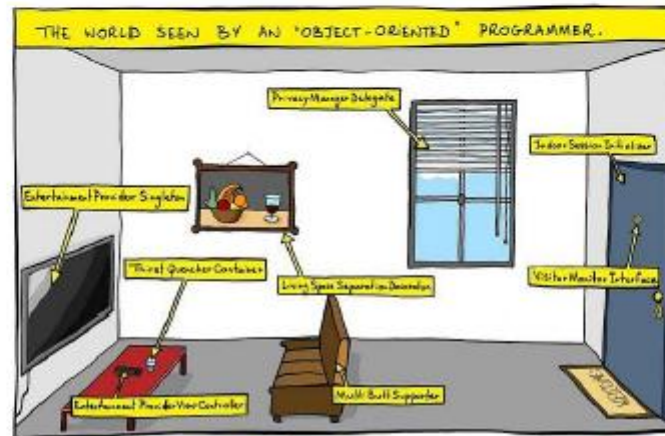




# Classificação das Linguagens: Paradigma

## 4. Paradigma Orientado a Objetos:

- Tratam os elementos e conceitos associados ao problema como **objetos**;
- Objetos são entidades abstratas que embutem dentro de suas fronteiras, as **características e operações** relacionadas com a entidade real;





# Classificação das Linguagens: Paradigma

## 4. Paradigma Orientado a Objetos:

- Sugere a diminuição da distância entre a modelagem computacional e o **mundo real**:
  - O ser humano se relaciona com o mundo através de conceitos de **objetos**;
  - Estamos sempre **identificando** qualquer objeto ao nosso redor;
  - Para isso lhe damos **nomes**, e de acordo com suas **características** lhes classificamos em **grupos**;
- Sistemas são vistos como **coleções de objetos** que se **comunicam**, enviando mensagens, colaborando para dar o comportamento global dos sistemas.



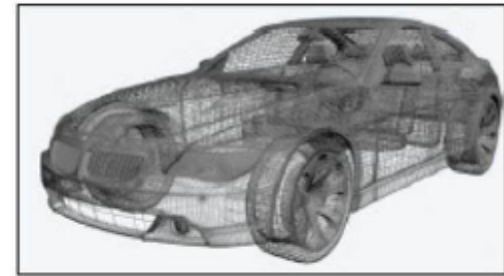


# Classificação das Linguagens: Paradigma

## 4. Paradigma Orientado a Objetos:

- Uma aplicação é estruturada em módulos (**classes**) que agrupam um estado (**atributos**) e operações (**métodos**) sobre este;
- A classe é o **modelo** ou **molde** de construção de **objetos**. Ela define as características e comportamentos que os objetos irão possuir.

Classe Carro



Objeto Carro2

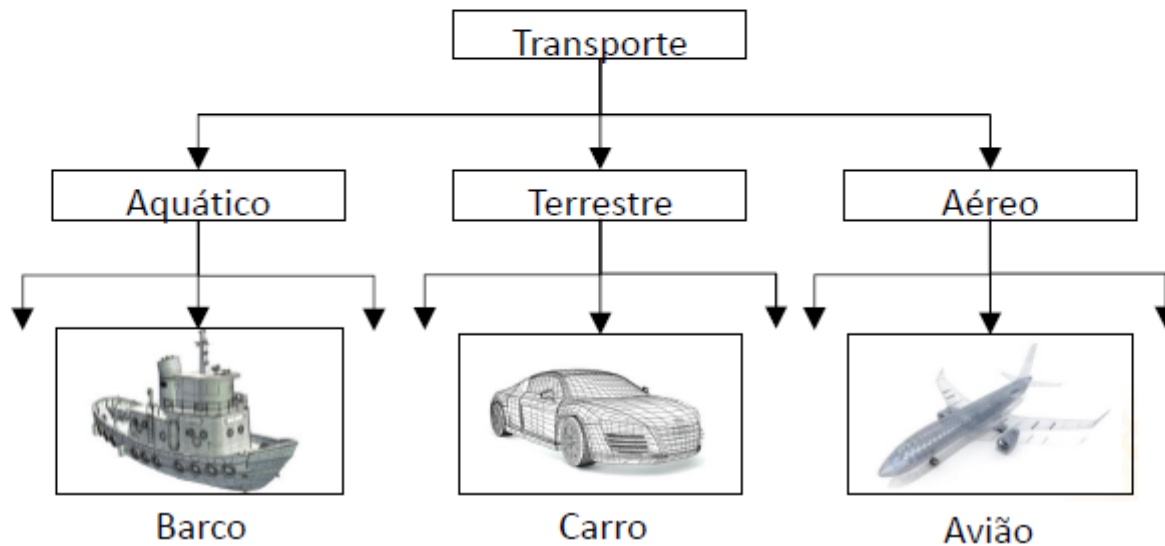




# Classificação das Linguagens: Paradigma

## 4. Paradigma Orientado a Objetos:

- A orientação a objetos permite que classes possam "herdar" as características e métodos de outra classe para expandí-la ou especializá-la de alguma forma.







# Classificação das Linguagens: Paradigma

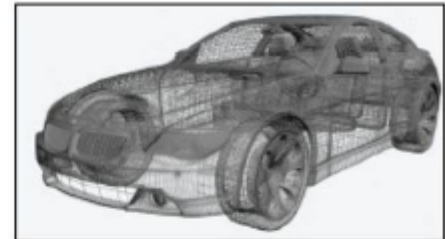
## 4. Paradigma Orientado a Objetos:

– Exemplo em Java:

```
public class Carro {  
    private String marca;  
    private String cor;  
    private String placa;  
    private int portas;  
    private int marcha;  
    private double velocidade;  
  
    public void Acelerar()  
    {  
        velocidade += marcha * 10;  
    }  
    public void Frear()  
    {  
        velocidade -= marcha * 10;  
    }  
}
```

Atributos

Métodos



Carro
- Marca: Texto - Cor: Texto - Placa: Texto - N° Portas: Inteiro ...
+ Acelerar(): void + Frear(): void + TrocarMarcha(x): void + Buzinar(): void ...





# Classificação das Linguagens: Paradigma

## 4. Paradigma Orientado a Objetos:

### – Exemplos de Linguagens Orientadas a Objetos:

- SIMULA 67;
- Smalltalk;
- C++;
- Java;
- C#;
- ADA;
- Eiffel;
- Perl;
- Ruby;
- PHP;
- ...





# Classificação das Linguagens: Paradigma

## 4. Paradigma Orientado a Objetos:

### – Vantagens:

- Organização do código;
- Aumenta a reutilização de código;
- Reduz tempo de manutenção de código;
- Ampla utilização comercial;

### – Desvantagens:

- Menos eficientes;

