

# Introdução às Linguagens de Programação

**Mauro Hemerly Gazzani**

mauro.hemerly@gmail.com

Universidade Estadual de Minas Gerais  
Câmpus de Ituiutaba

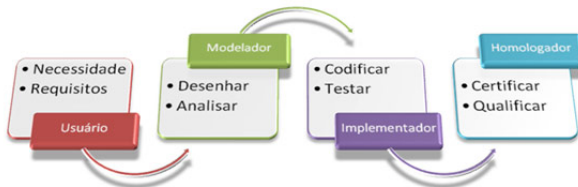
<https://github.com/mauro-hemerly/UEMG-2019-1>

# O que é uma Linguagem de Programação?

- Na programação de computadores, uma linguagem de programação serve como **meio de comunicação** entre o indivíduo (**desenvolvedor**) que deseja resolver um determinado problema e o computador.
- A linguagem de programação deve fazer a ligação entre o **pensamento humano** (muitas vezes de natureza não estruturada) e a **precisão requerida para o processamento** pelo computador.

# O que é uma Linguagem de Programação?

- Uma linguagem de programação auxilia o desenvolvedor (programador) no processo de desenvolvimento de software:
  - Projeto;
  - Implementação;
  - Teste;
  - Verificação;
  - Manutenção do software.



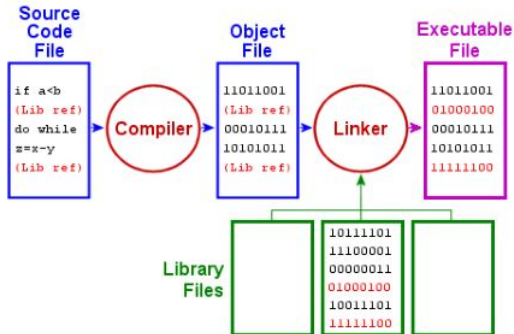
Ciclo de Vida de Desenvolvimento de Software (SDLC - Software Development Life Cycle)

# O que é uma Linguagem de Programação?

- Uma linguagem de programação é uma **linguagem** destinada para ser usada por uma pessoa (**desenvolvedor**) para **expressar um processo (algoritmo)** através do qual um computador possa resolver um problema.
- Os **modelos/paradigmas** de linguagens de programação correspondem a diferentes **pontos de vista** dos quais processos podem ser expressados.
  - Exemplos: **Imperativo, orientado a objetos, funcional, e lógico.**

# O que é uma Linguagem de Programação?

- Para que se tornem operacionais, os programas escritos em **linguagens de alto nível** devem ser traduzidos para **linguagem de máquina**.



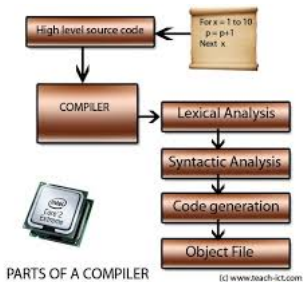
# O que é uma Linguagem de Programação?

- A conversão de um código em **linguagem alto nível para linguagem de máquina** é realizada através de sistemas especializados:




## Compiladores ou Interpretadores

- Esses sistemas recebem como entrada uma representação textual da solução de um problema (expresso em uma **linguagem fonte**) e produzem uma representação do mesmo algoritmo expresso em uma **linguagem de máquina**.

# O que é uma Linguagem de Programação?



# O que é uma Linguagem de Programação?

High Level Language	Assembly Language	Machine Language
		
<code>i = j + k;</code>	1    ILOAD j    // i = j + k	
<code>if (i == 3)</code>	2    ILOAD k	
<code>k = 0;</code>	3    IADD	10111001    00000000
<code>else</code>	4    ISTORE i	11010010    10100001
<code>j = j - 1;</code>	5    ILOAD i    // if (i < 3)	00000100    00000000
	6    BIPUSH 3	10001001    00000000
	7    IF_ICMPEQ L1	00001110    10001011
	8    ILOAD j    // j = j - 1	00000000    00011110
	9    BIPUSH 1	00000000    00000010
	10   ISUB	10111001    00000000
	11   ISTORE j	11100001    00000011
	12   GOTO L2	00010000    11000011
	13 L1:        BIPUSH 0	10001001    10100011
	14   ISTORE k	00001110    00000100
	15 L2:	00000010    00000000



# TIOBE Index for February 2019

O índice **TIOBE Programming Community** é um indicador da popularidade das linguagens de programação. O índice é atualizado uma vez por mês. As avaliações são baseadas no número de engenheiros especializados em todo o mundo, cursos e fornecedores e parceiros . Motores de busca populares como **Google, Bing, Yahoo!, Amazon, YouTube** e **Baidu** são usados para calcular as classificações. É importante notar que o índice **TIOBE não é sobre a melhor linguagem de programação e nem também aquela que a maioria das linhas de código foram escritas.**

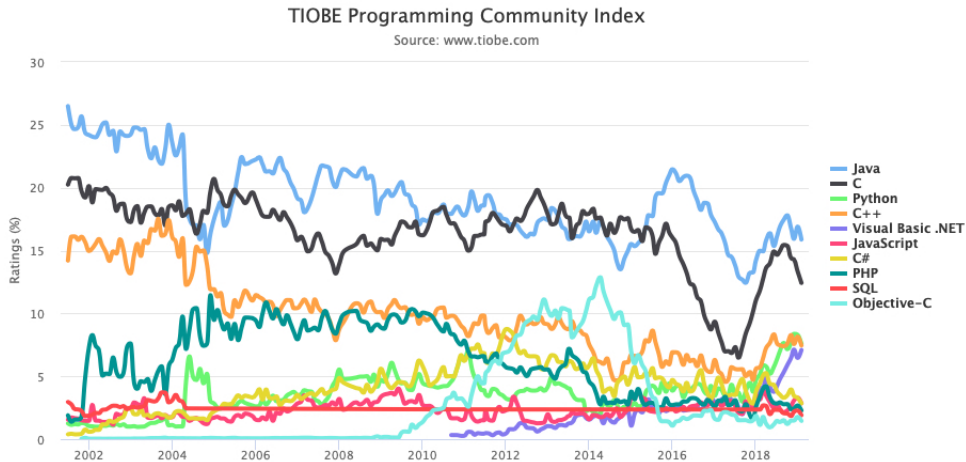
## Onde encontrar o Índice TIOBE

<http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html>

# TIOBE Index for February 2019

Feb 2019	Feb 2018	Change	Programming Language	Ratings	Change
1	1		Java	15.876%	+0.89%
2	2		C	12.424%	+0.57%
3	4	↗	Python	7.574%	+2.41%
4	3	↘	C++	7.444%	+1.72%
5	6	↗	Visual Basic .NET	7.095%	+3.02%
6	8	↗	JavaScript	2.848%	-0.32%
7	5	↘	C#	2.846%	-1.61%
8	7	↘	PHP	2.271%	-1.15%
9	11	↗	SQL	1.900%	-0.46%
10	20	↗	Objective-C	1.447%	+0.32%
11	15	↗	Assembly language	1.377%	-0.46%
12	19	↗	MATLAB	1.196%	-0.03%
13	17	↗	Perl	1.102%	-0.66%

# TIOBE Index for February 2019



## Classificação das Linguagens

- As linguagens de programação podem ser classificadas em relação a **três critérios**:
  - **Em relação ao nível:**
    - Baixo nível, Médio nível, ou Alto nível;
  - **Em relação à geração:**
    - 1ª Geração, 2ª Geração, 3ª Geração, 4ª Geração, ou 5ª Geração;
  - **Em relação ao paradigma:**
    - Imperativo, Funcional, Lógico, Orientado a Objetos, ...;

## Classificação das Linguagens: Nível

- **Baixo Nível:**

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

# Classificação e Paradigmas de LP

## Assembly – Exemplo

```
section .text
    global _start
_start:
    mov     eax, '3'
    sub     eax, '0'
    mov     ebx, '4'
    sub     ebx, '0'
    add     eax, ebx
    add     eax, '0'
    mov     [sum], eax
    mov     ecx, msg
    mov     edx, len
    mov     ebx, 1
    mov     eax, 4
    int     0x80
    mov     ecx, sum
    mov     edx, 1
    mov     ebx, 1
    mov     eax, 4
    int     0x80
    mov     eax, 1
    int     0x80
section .data
    msg db "Resultado:", 0xA, 0xD
```

## Classificação das Linguagens: Nível

- **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 e Paradigmas de LP

## 8086 Microprocessor Emulator

The screenshot displays an 8086 Microprocessor Emulator interface. The main window is titled "emulator: add-sub.com\_". It features a menu bar (file, math, debug, view, external, virtual devices, virtual drive, help) and a toolbar with buttons for Load, reload, step back, single step, and run. Below the toolbar, the registers section shows the state of various registers: AX (00 00), BX (00 00), CX (00 26), DX (00 00), CS (0700), IP (0100), SS (0700), SP (FFFE), BP (0000), SI (0000), and DI (0000). The memory window displays two segments: 0700:0100 and 0700:0102. The 0700:0100 segment shows memory addresses from 07100 to 0710F with corresponding hex values and decimal offsets. The 0700:0102 segment shows assembly instructions: MOV AL, 05h, MOV BL, 0Ah, ADD BL, AL, SUB BL, 01h, MOV CX, 00008h, MOV AH, 02h, MOV DL, 030h, TEST BL, 080h, JZ 0117h, MOV DL, 031h, INT 021h, SHL BL, 1, LOOP 010Ch, MOV DL, 062h, and INT 021h. The original source code window on the right shows the assembly code for "add-sub". It starts with "name 'add-sub'" and "org 100h". The code includes instructions for moving values into AL and BL, adding them, subtracting 1 from BL, and printing the result in binary. Comments explain the binary and hexadecimal values used.

```
01 name "add-sub"
02
03 org 100h
04
05 mov al, 5 ; bin=00000101b
06 mov bl, 10 ; hex=0ah or bin=00001010b
07
08 ; 5 + 10 = 15 (decimal) or hex=0fh
09 add bl, al
10
11 ; 15 - 1 = 14 (decimal) or hex=0eh
12 sub bl, 1
13
14 ; print result in binary:
15 mov cx, 8
16 print: mov ah, 2 ; print function
17 mov dl, '0'
18 test bl, 10000000b ; test first bit
19 jz zero
20 mov dl, '1'
21 zero: int 21h
22 shl bl, 1
```



## Classificação das Linguagens: Nível

- **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 e Paradigmas de LP

## Classificação das Linguagens: Nível

- **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

- **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

- **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 e Paradigmas de LP

## Classificação das Linguagens: Nível

- **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

- **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)**

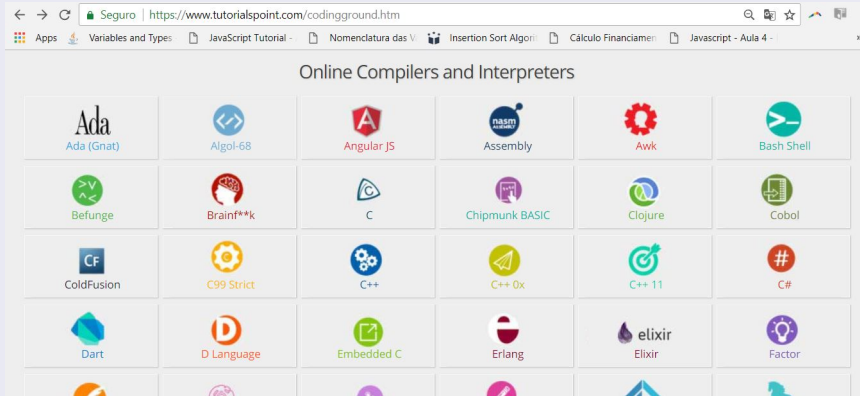
- Compreende as linguagens simbólicas ou de montagem (Assembly), projetadas para **minimizar as dificuldades** da programação em **notação binária**.
- Códigos de operação e endereços binários foram substituídos por mnemônicos (abreviações).

**Exemplo:** `mov mul add label goto`

**Compilador Online:** [https://www.tutorialspoint.com/compile\\_assembly\\_online.php](https://www.tutorialspoint.com/compile_assembly_online.php)

# Classificação e Paradigmas de LP

## Online Compilers and Interpreters



# Classificação e Paradigmas de LP

## 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 e Paradigmas de LP

## 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**, onde:

**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 e lógicos;
  - \* Instruções de controle de fluxo de execução (desvios condicionais, incondicionais e processamento iterativo (loop)).

# Classificação e Paradigmas de LP

## 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 em **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
```

## Compilador Basic Online

[http://www.tutorialspoint.com/compile\\_freebasic\\_online.php](http://www.tutorialspoint.com/compile_freebasic_online.php)

# Classificação e Paradigmas de LP

## 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.
```

## Compilador Pascal Online

[http://www.tutorialspoint.com/compile\\_pascal\\_online.php](http://www.tutorialspoint.com/compile_pascal_online.php)

## Classificação das Linguagens: Geração

- **3ª Geração: linguagens orientadas ao usuário**

- 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 e Paradigmas de LP

## 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, Clarion, Scilab, Strata, ...

## Classificação das Linguagens: Geração

- **5ª Geração: linguagens do conhecimento**

- Linguagens de programação para resolução de problemas a partir de **restrições** dadas para o programa em vez de desenvolver algoritmos.
- São usadas principalmente na área de **Inteligência Artificial**.
- Facilitam a representação do conhecimento, o que é essencial para a simulação de comportamentos inteligentes.

## Classificação das Linguagens: Geração

- **5ª Geração: linguagens do conhecimento**

- Linguagens de programação lógica e linguagens baseadas em restrições geralmente pertencem a 5ª geração. Exemplo: **Prolog**
- Na década de 80, as linguagens de 5ª geração eram **consideradas o futuro da computação**. Sendo surgido que elas substituiriam as linguagens de gerações anteriores.
- Problema: desenvolver algoritmos **genéricos e eficientes** é um problema complexo.

## 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 e Paradigmas de LP

## 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.

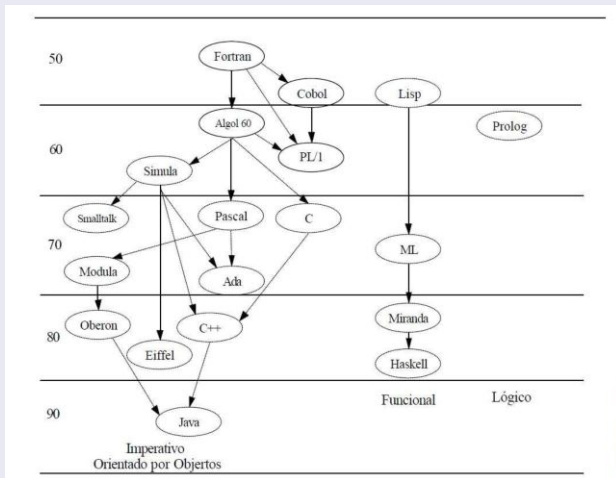
# Classificação e Paradigmas de LP

## 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 e Paradigmas de LP

## Classificação das Linguagens: Paradigma



## Classificação das Linguagens: Paradigma

### ● Paradigma Imperativo

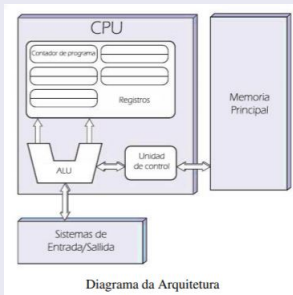
- As linguagens imperativas são orientadas a **ações**, onde a computação é vista como uma **sequência de instruções** que manipulam valores de **variáveis** (leitura e atribuição).
- Os programas são centrados no conceito de um **estado** (modelado por variáveis) e **ações** (comandos) que manipulam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

# Classificação e Paradigmas de LP

## Classificação das Linguagens: Paradigma

### ● 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;
  - \* Ciclo de **busca-decodifica-executa** de uma instrução.



# Classificação e Paradigmas de LP

## Classificação das Linguagens: Paradigma

- **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 e Paradigmas de LP

## Classificação das Linguagens: Paradigma

- **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

### • 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

- **Paradigma Imperativo**

- Exemplos de Linguagens Imperativas:

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

## Classificação das Linguagens: Paradigma

### ● Paradigma Imperativo

#### - Vantagens:

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

#### - Desvantagens:

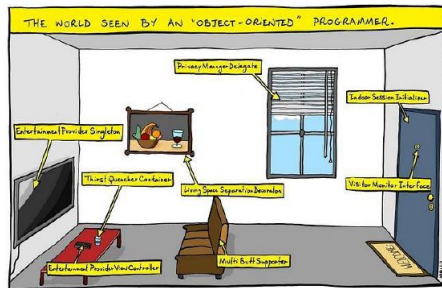
- Possui difícil legibilidade e facilita introdução de erros em sua manutenção;
- Descrições demasiadamente profissional focaliza o "como" e não o "quê";
- Tende a gerar códigos confusos, onde tratamento dos dados são misturados com o comportamento do programa;

# Classificação e Paradigmas de LP

## Classificação das Linguagens: Paradigma

### ● Paradigma Orientado a Objetos

- Trata 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 e Paradigmas de LP

## Classificação das Linguagens: Paradigma

- **Paradigma Orientado a Objetos**

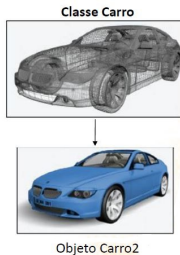
- Sugere a diminuição da distância entre a modelagem computacional e o **mundo real**:
  - O ser humano se relaciona com o mundo através de conceitos de **objetos**;
  - Estamos sempre **identificando** qualquer objeto ao nosso redor;
  - Para isso lhe damos **nomes**, e de acordo com suas **características** 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 e Paradigmas de LP

## Classificação das Linguagens: Paradigma

### • 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.

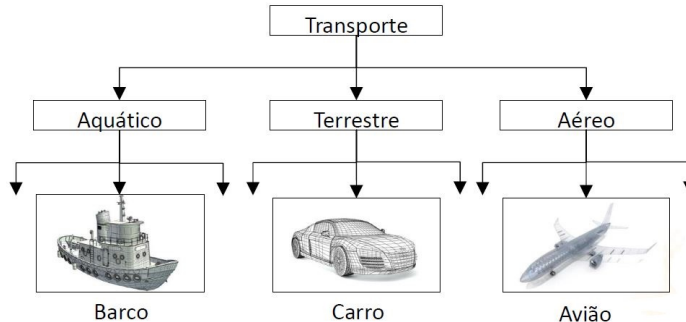


# Classificação e Paradigmas de LP

## Classificação das Linguagens: Paradigma

- **Paradigma Orientado a Objetos**

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



# Classificação e Paradigmas de LP

## Classificação das Linguagens: Paradigma

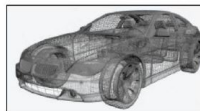
- Paradigma Orientado a Objetos

- Exemplo **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

- **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

- **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;

## Classificação das Linguagens: Paradigma

### • 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

### • 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 e Paradigmas de LP

## Classificação das Linguagens: Paradigma

- **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 e Paradigmas de LP

## Classificação das Linguagens: Paradigma

- **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;

## Compilador Haskell Online

[https://www.tutorialspoint.com/compile\\_haskell\\_online.php](https://www.tutorialspoint.com/compile_haskell_online.php)

## Classificação das Linguagens: Paradigma

- **Paradigma Funcional**

- Exemplos de Linguagens Funcionais:
  - Haskell;
  - Scheme;
  - Common LISP;
  - CLOS (Common LISP Object System);
  - Miranda;
  - ML;
  - Erlang;
  - Ocaml;
  - ...

# Classificação e Paradigmas de LP

## Linguagem Funcional Haskell

### 1º Exemplo

Execute   Embed	main.hs	STDIN	Result
<pre>1 dobrar::Int-&gt;Int 2 dobrar x = 2 * x 3 4 quadruplicar::Int-&gt;Int 5 quadruplicar x = dobrar (dobrar x) 6 7 quadrado::Int-&gt;Int 8 quadrado x = x * x 9 10 11 main = do 12     print (dobrar 4) 13     print (quadruplicar 5) 14     print (quadrado 7)</pre>			<pre>\$ghc -O2 --make *.hs -o main -threaded -rtsopts [1 of 1] Compiling Main             ( main.hs, main.o ) Linking main ... \$main 8 20 49</pre>

# Classificação e Paradigmas de LP

## Linguagem Funcional Haskell

- Um programa em **linguagem funcional** é constituído de uma série de definições de funções.
- **Definição de Funções em Haskell:** em geral uma simples definição de função terá a seguinte forma: **nome x1 x2 x3 ... xn = e**  
**nome:** nome da função que está sendo definida.  
**xi:** os parâmetros formais.  
**e:** o resultado, definido em termos dos parâmetros formais.
- **Declaração dos Tipos em Haskell:** acompanha a definição da função e tem a seguinte forma: **nome :: t1 -> t2 ->...-> tk -> t**  
**nome:** nome da função que está sendo definida.  
**ti:** os tipos dos parâmetros formais.  
**t:** o tipo do resultado da aplicação da função.



# Classificação e Paradigmas de LP

## Linguagem Funcional Haskell

- 2º Exemplo: Verifica se o número é par

Execute   Embed	main.hs	STDIN	Result
<pre>1 --Verifica se o número é par 2 par::Int-&gt;Bool 3 par x = if mod x 2 == 0 then True else False 4 5 main = print (par 2) 6</pre>			<pre>\$ghc -O2 --make *.hs -o main -threaded -rtsopts [1 of 1] Compiling Main             ( main.hs, main.o ) Linking main ... \$main True</pre>

- 3º Exemplo: Determina o menor entre dois números

Execute   Embed	main.hs	STDIN	Result
<pre>1 --Determina o menor entre dois números 2 menor::Int-&gt;Int-&gt;Int 3 menor x y = if x &lt; y then x else y 4 5 main = print (menor 99 55) 6</pre>			<pre>\$ghc -O2 --make *.hs -o main -threaded -rtsopts [1 of 1] Compiling Main             ( main.hs, main.o ) Linking main ... \$main 55</pre>

# Classificação e Paradigmas de LP

## Linguagem Funcional Haskell

- 4º Exemplo: Determina o fatorial de um número natural

Execute   Embed	main.hs	STDIN	Result
<pre>1 --Fatorial de um número natural 2 fatorial::Int-&gt;Int 3 fatorial 0 = 1 4 fatorial n = n * fatorial (n-1) 5 6 main = print (fatorial 5)</pre>			<pre>\$ghc -O2 --make *.hs -o main -threaded -rtsopts [1 of 1] Compiling Main                ( main.hs, main.o ) Linking main ...  \$main 120</pre>

- 5º Exemplo: Determina o tamanho de uma lista de inteiros

Execute   Embed	main.hs	STDIN	Result
<pre>1 --Determina o tamanho de uma lista de inteiros 2 tamanho::[Int]-&gt;Int 3 tamanho [] = 0 4 tamanho (a:x) = 1 + tamanho x 5 6 main = print (tamanho [11,20,33,4,95,6])</pre>			<pre>\$ghc -O2 --make *.hs -o main -threaded -rtsopts [1 of 1] Compiling Main                ( main.hs, main.o ) Linking main ...  \$main 6</pre>

## Classificação das Linguagens: Paradigma

- **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;

## Classificação das Linguagens: Paradigma

- **Paradigma Lógico**

- Paradigma de programação baseado em **lógica formal** (Lógica de Predicados);
- 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

### • 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 e Paradigmas de LP

## Classificação das Linguagens: Paradigma

- **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).
```

## Compilador Prolog Online

[http://www.tutorialspoint.com/execute\\_prolog\\_online.php](http://www.tutorialspoint.com/execute_prolog_online.php)

## Classificação das Linguagens: Paradigma

- **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;

# Motivos para Estudar os Conceitos de Linguagens de Programação

- **Aumentar a capacidade de expressar ideias:**

- Conhecimento amplo dos **recursos de linguagem** reduz as limitações no desenvolvimento de software;
- A melhor compreensão das funções e implementação das estruturas de uma linguagem de programação nos leva a usar a linguagem de modo a **extrair o máximo** de sua funcionalidade e eficiência.



# Motivos para Estudar os Conceitos de Linguagens de Programação

- **Maior conhecimento para escolha de linguagens apropriadas:**
  - Algumas linguagens são mais apropriadas para resolver determinados problemas;
  - Escolher a melhor linguagem para um problema específico devido ao conhecimento de novos recursos é difícil para:
    - Desenvolvedores antigos;
    - Desenvolvedores sem formação formal.

# Motivos para Estudar os Conceitos de Linguagens de Programação

- **Entender melhor a importância da implementação:**

- Leva a um entendimento do **porquê** das linguagens serem projetadas de determinada maneira;
- Melhora as escolhas que podemos fazer entre as linguagens de programação e as consequências das opções ;
- Nos permite desenvolver programas mais eficientes.
- **Algoritmo recursivo x iterativo**

# Motivos para Estudar os Conceitos de Linguagens de Programação

- **Maior capacidade para aprender novas linguagens:**
  - Na computação, o aprendizado contínuo é fundamental;
  - Compreender os conceitos gerais das linguagens torna mais fácil entender como eles são incorporados na linguagem que está sendo aprendida.

# Motivos para Estudar os Conceitos de Linguagens de Programação

- **Avanço global da computação:**

- Nem sempre as linguagens mais populares são melhores, por quê?

Imposição

- Por que existem várias linguagens de programação?

Resolução específica de problemas.

# Domínios da Programação de Computadores

- Computadores têm sido aplicados a uma infinidade de áreas...



## Aplicações Científicas

- Os primeiros computadores que surgiram na década de 40 foram projetados e utilizados para **aplicações científicas**.
- Nesta categoria se enquadram todos os problemas que necessitam um grande volume de processamento, com operações geralmente feitas em **ponto flutuante**, e com poucas exigências de entrada e saída.
  - Uma das preocupações primárias neste tipo de aplicação é a **eficiência**.
- As aplicações científicas incentivaram a criação de algumas linguagens de alto nível, como por exemplo o **Fortran**.

# Fortran - Exemplo

```
program teste
  real a, b, s
  read *, a, b
  s = a + b
  print *, a, ' + ' , b
  print *, ' = ' , s
end
```

**Compilador Fortran Online**

[http://www.tutorialspoint.com/compile\\_fortran\\_online.php](http://www.tutorialspoint.com/compile_fortran_online.php)

# Domínios da Programação de Computadores

## Aplicações Comerciais

- O desenvolvimento de **aplicações comerciais** teve início na década de 50.
- A primeira linguagem bem sucedida para o desenvolvimento de aplicações comerciais foi o **COBOL** (em 1960).
- As linguagens de programação comerciais se caracterizam pela facilidade de elaborar relatórios e armazenar números decimais e dados de caracteres.



# Cobol - Exemplo

## COBOL- Exemplo

```
IDENTIFICATION DIVISION.
PROGRAM-ID. Teste.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  Num1          PIC 9.
01  Num2          PIC 9.
01  Result        PIC 99.
01  Operador      PIC X.

PROCEDURE DIVISION.
Calcula.
    PERFORM 3 TIMES
        DISPLAY "Numero 1: "
        ACCEPT Num1
        DISPLAY "Numero 2: "
        ACCEPT Num2
        DISPLAY "Operador (+ ou *): "
        ACCEPT Operador
        IF Operador = "+" THEN
            ADD Num1, Num2 GIVING Result
        END-IF
        IF Operador = "*" THEN
            MULTIPLY Num1 BY Num2 GIVING Result
        END-IF
        DISPLAY "Result is = ", Result
    END-PERFORM.
STOP RUN. http://www.tutorialspoint.com/compile\_cobol\_online.php
```

## Inteligência Artificial

- O desenvolvimento de aplicações para inteligência artificial teve início no final da década de 50.
- Essas aplicações caracterizam-se pelo uso de computações simbólicas em vez de numéricas (são manipulados nomes e não números);
- A primeira linguagem desenvolvida para IA foi a funcional **LISP** (1959).
- No início dos anos 70 surge a programação lógica: **Prolog**.

# LISP - Exemplo

```
(defun fatorial (num)
  (cond ((zerop num) 1)
        (t (* num (fatorial (- num 1)))))
)
)
(setq n 6)
(format t "Fatorial ~d = ~d" n (fatorial n))
```

**Compilador LISP Online**

[http://www.tutorialspoint.com/execute\\_lisp\\_online.php](http://www.tutorialspoint.com/execute_lisp_online.php)

# Prolog - Exemplo

```
pai(fred, marcos).  
pai(ricardo, pedro).  
pai(pedro, paulo).  
avo(X,Y) :- pai(X, Z), pai(Z, Y).
```

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

**Compilador Prolog Online**

[http://www.tutorialspoint.com/execute\\_prolog\\_online.php](http://www.tutorialspoint.com/execute_prolog_online.php)

## Software Básico (Plataforma de Software)

- O software básico (**sistema operacional**) deve possuir eficiência na execução por propiciar suporte a execução de outros aplicativos.
- As linguagens de programação para este tipo de sistema devem oferecer execução rápida e ter recursos de baixo nível que permitam ao software fazer interface com os dispositivos externos.
- O sistema operacional **UNIX** foi desenvolvido quase inteiramente em **C** (tornando-o fácil de portar para diferentes máquinas).

# Cr terios de Avalia  o de Linguagens

- Um dos cr terios mais importantes para julgar uma linguagem de programa  o   a facilidade com que os programas s o **lidos e entendidos**.
- Antes dos anos 70: linguagens foram criadas pensando em termos de escrita de c digo.
  - Efici ncia e legibilidade da m quina;
  - As linguagens foram projetadas mais do ponto de vista do computador do que do usu rio.
- Na d cada de 70 foi desenvolvido o conceito de ciclo de vida de software: **manuten  o**.

# Critérios de Avaliação de Linguagens

## Critérios de Avaliação

- Legibilidade
  1. Simplicidade geral
  2. Ortogonalidade
  3. Instruções de controle
  4. Tipos de dados e estruturas
  5. Considerações sobre sintaxe
- Capacidade de Escrita
  1. Simplicidade e Ortogonalidade
  2. Suporte para abstração
  3. Expressividade

# Critérios de Avaliação de Linguagens

## Critérios de Avaliação

- Confiabilidade
  1. Verificação de Tipos
  2. Manipulação de Exceções
  3. Apelidos (Aliasing)
  4. Legibilidade e Facilidade de Escrita
- Custo



# Critérios de Avaliação: Legibilidade

- A facilidade de manutenção é determinada em grande parte, pela **legibilidade** dos programas, dessa forma ela se tornou uma medida importante da qualidade dos programas e das linguagens.
- A legibilidade deve ser considerada no contexto do **domínio do problema**.
  - Um programa escrito em uma linguagem não apropriada para o domínio do problema se mostra antinatural, "enrolado" e difícil de ser lido.

## 1. Simplicidade geral:

- A simplicidade geral de uma linguagem de programação afeta fortemente sua legibilidade;
- Uma linguagem com um **grande número de componentes básicos** é mais difícil de ser manipulada do que uma com poucos desses componentes.
  - Os programadores que precisam usar uma linguagem grande tendem a aprender um subconjunto dela e ignorar seus outros recursos.
  - Isso pode ser um problema quando o leitor do programa aprende um conjunto diferente de recursos daquele que o autor aplicou em seu programa.

# Critérios de Avaliação: Legibilidade

## 1. Simplicidade geral:

- Uma segunda característica que complica a legibilidade é a **multiplicidade de recursos** (mais que uma maneira de realizar uma operação particular);

### Exemplo em C:

```
cont = cont + 1;  
cont += 1;  
cont++;  
++cont;
```

Mesmo significado  
quando usadas em  
expressões separadas!

# Critérios de Avaliação: Legibilidade

## 1. Simplicidade geral:

- Um terceiro problema é a **sobrecarga de operadores**, na qual um único símbolo tem mais que um significado.
- Apesar de ser um recurso útil, pode ser prejudicial a legibilidade se for permitido aos usuários criar suas próprias sobrecargas.

**Exemplo:** sobrecarregar o  $+$  para adicionar inteiros, reais, concatenar strings, somar vetores...

# Critérios de Avaliação: Legibilidade

## 1. Simplicidade geral:

- A simplicidade de linguagens, no entanto pode ser levada ao extremo, por exemplo a forma e o significado da maioria das **instruções em Assembly são modelos de simplicidade**, entretanto torna os programas em Assembly **menos legíveis**.
- Falta instruções de controle mais complexas, o que torna necessário o uso de mais comandos para expressar problemas do que os necessário em linguagens de alto nível.

## Assembly – Exemplo

```
section .text
    global _start
_start:
    mov     eax, '3'
    sub     eax, '0'
    mov     ebx, '4'
    sub     ebx, '0'
    add     eax, ebx
    add     eax, '0'
    mov     [sum], eax
    mov     ecx, msg
    mov     edx, len
    mov     ebx, 1
    mov     eax, 4
    int     0x80
    mov     ecx, sum
    mov     edx, 1
    mov     ebx, 1
    mov     eax, 4
    int     0x80
    mov     eax, 1
    int     0x80
section .data
    msg db "Resultado:", 0xA, 0xD
    len equ $ - msg
segment .bss
    sum resb 1 http://www.tutorialspoint.com/compile\_assembly\_online.php
```