



# PROGRAMAÇÃO DE COMPUTADORES I

## PRIMEIRA PARTE

---

“As máquinas me surpreendem muito frequentemente.

*Alan Turing*

CAPÍTULO I

INTRODUÇÃO À LÓGICA  
DE PROGRAMAÇÃO E  
ALGORITMOS

# Programação de Computadores I

prof. Marco Villaça

# Programa e Linguagem de Programação

---

- O processador de um computador (CPU – Central Processing Unit) executa uma lista de instruções armazenadas na memória. Esta lista de instruções é chamada de **programa**.
- Os programas são elaborados em uma Linguagem de Programação (LP)

# Programa e Linguagem de Programação

---

- Uma linguagem de programação é um conjunto de regras (sintáticas e semânticas) que permite a comunicação entre uma pessoa que necessita resolver um problema e um computador escolhido para ajudá-lo na solução.
  - Sintaxe é a forma como as instruções de uma linguagem são escritas, mas sem atender ao seu significado (= semântica).
  - Por exemplo, a **sintaxe** da instrução 'se' na linguagem C é: `if (expressão) {instrução}` e sua **semântica** é: “se o valor da expressão for verdadeiro, as instruções entre as chaves serão executadas pelo programa”

# Programa e Linguagem de Programação

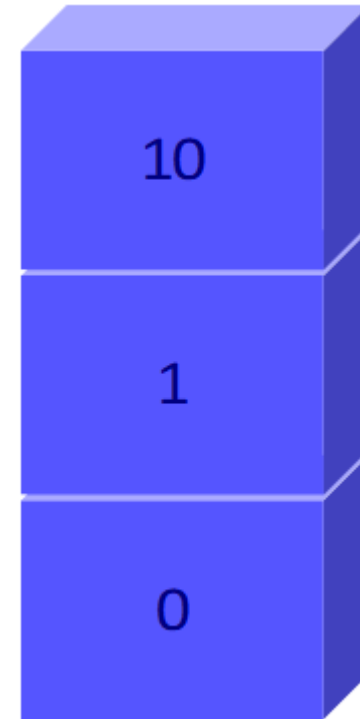
---

- Entretanto, uma linguagem de programação não se resume a este conjunto de regras.
  - Uma linguagem de programação é, sobretudo, um meio de exprimir ideias segundo uma metodologia.

# Mémoria do Computador


---

- Imagine um conjunto de caixas empilhadas e em cada caixa um número pintado ao seu lado em números binários.
- A primeira caixa é pintada com o número 0, a segunda como o número 1, a terceira, com 10 (2 em decimal) e, assim, sucessivamente.



11111111111111111111111111111111

- A última caixa da pilha é reconhecida pelo número 11111111111111111111111111111111 (4.294.967.295 em decimal ou  $2^{32}-1$ ).
- Imagine que, ao invés de objetos, estas caixas armazenam bytes.
- Um byte pode conter um número que varia de 00000000 a 11111111 (255 em decimal ou  $2^8-1$ ).

- 
- A blue 3D box with the number '10' on its front face. A blue ballot slip with the binary string '0100011' is being inserted into a slot on top of the box. The slip is tilted upwards and to the right, with a trail of small blue dots behind it.





- No final dos anos 70, Cleve Moler inventou a linguagem Matlab (Matrix Laboratory), voltada para o tratamento de matrizes.
- O Matlab foi lançado comercialmente em 1984 pela MathWorks, tornando-se um sucesso entre engenheiros e cientistas.
- O [Scilab](#), criado pelo instituto francês INRIA (Institut National de Recherche en Informatique et Automatique), é um software de código aberto para computação numérica inspirado no Matlab.



- O sucesso de programas como o Matlab e o Scilab se deve a facilidade que oferecem para a elaboração de pequenos programas voltados para as engenharias e ciências.
- A versão atualizada do Scilab pode ser baixada para GNU/LINUX, Windows e Mac OS X em:  
<http://www.scilab.org/download/latest>.



# Linguagens imperativas

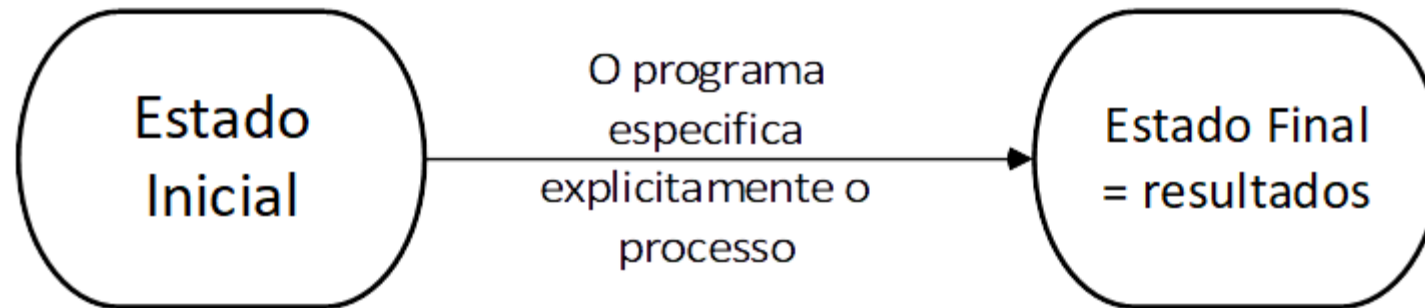
---

- 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)
- A maioria das linguagens imperativas são **procedurais**, paradigma que utiliza sub-rotinas ou procedimentos como mecanismo de estruturação.
- Este paradigma trata dados e procedimento(s) como duas entidades diferentes.
- O paradigma imperativo foi o primeiro a surgir e ainda mantém a sua importância.

# Linguagens imperativas

## Modelo computacional

---



# Linguagens imperativas

---

- Paradigma dominante e bem estabelecido que apresenta como problema o relacionamento indireto entre entrada e saída de dados:
- Difícil legibilidade
- Erros introduzidos durante manutenção
- Descrições demasiadamente operacionais
- Exemplos: Fortran, Pascal e C

# Linguagens declarativas

---

- Ao invés de o programa estipular a maneira de chegar à solução passo a passo, como acontece nas linguagens procedurais, ele fornece uma descrição do problema que se pretende computar utilizando uma **coleção de fatos e regras** (lógica) que indicam como deve ser resolvido o problema proposto.
- Em resumo, define a lógica do programa, mas não o fluxo de controle detalhado

# Linguagens declarativas

## Prolog

---

- Dados os fatos:

```
pai(mario,marco) .  
pai(mario,sergio) .  
pai(marco,pedro) .  
pai(sergio,michel) .
```

- E a seguinte regra:

```
avo(X,Z) :- pai(X,Y), pai(Y,Z) .
```

- Que significa que **se (-)** alguém é pai de uma pessoa, que por sua vez é pai de outra pessoa, então ele é avô.

# Linguagens declarativas

## Prolog

---

- Qual a resposta da seguinte questão:

`?- avo(mario,pedro), avo(mario,michel) .`

- A questão retornará a saída: "YES"

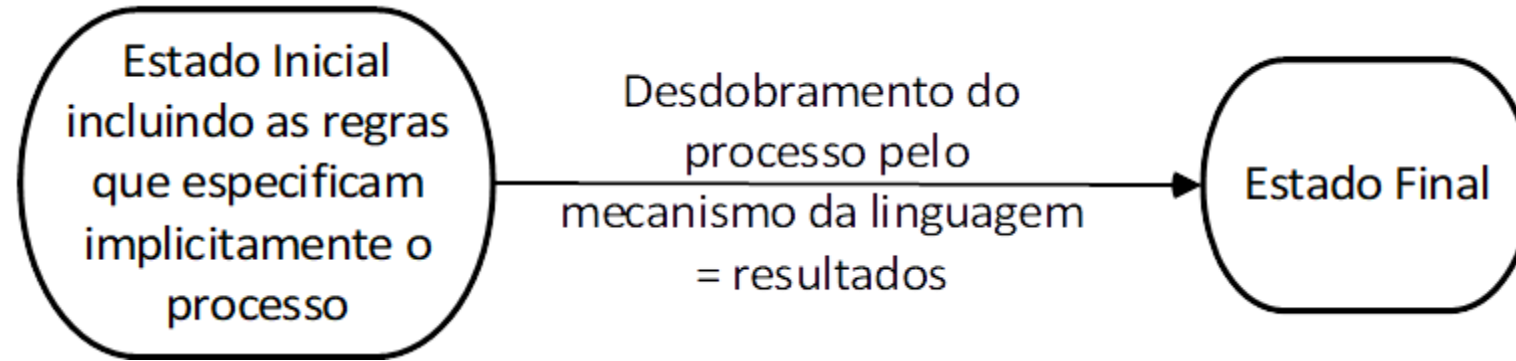
Pois, "mario" é avô de "pedro" e "michel", pois "mario" é pai de "marco" e "sergio", que por sua vez são pais de "pedro" e "michel".



# Linguagens declarativas

## Modelo computacional

---



# Linguagens declarativas

## Prolog

---

Interface de desenvolvimento *online* para Prolog – SWI:

<https://swish.swi-prolog.org>



# Linguagens orientadas a objeto

---

- O modelo Orientado a Objeto focaliza mais o problema. Um programa OO é equivalente a objetos que mandam mensagens entre si. Os objetos do programa equivalem aos objetos da vida real (problema).
- Os objetos são compostos de Atributos e Métodos:
  - Atributos: características que descrevem o objeto (seu estado)
  - Métodos: definem o comportamento de um objeto

# Exemplo de objeto

---

- Objeto gato
  - Atributos: nome, idade, raça, cor dos pelos

Gato	Gato
Fernando	Ivan
8 anos	12 anos
Persa	Siamês
Fumaça	Amarelo
Aniversario()	Aniversario()
Correr( )	Correr( )

# Exemplo de objeto

---

- Objeto gato

- Definindo, por exemplo:

- ```
fernando = Gato("Fernando", 8, "Persa", "Fumaça")
```

- Poderíamos acessar seus atributos utilizando

- ```
fernando.nome, fernando.idade, ...
```

- Chamando o método

- ```
fernando.aniversario()
```

- Sua idade (`fernando.idade`) passaria a ser 9

# Encapsulamento

---

- O princípio mais importante da orientação a objetos é o encapsulamento: a ideia de que os dados dentro do objeto devem ser acessados apenas através de uma interface pública - ou seja, os métodos do objeto.
- No entanto, os métodos de um objeto podem acessar os métodos de outros objetos.
- Ele vincula os dados às funções que operam nele e os protege de modificações acidentais de funções externas.

# Classes

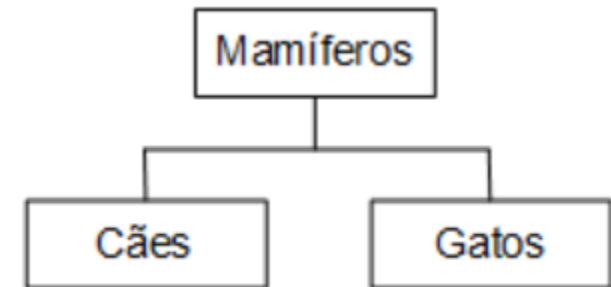
---

- Olhando os 2 gatos do exemplo anterior, percebe-se que eles possuem exatamente o mesmo conjunto de atributos.
- Isto acontece porque são 2 objetos da mesma classe, ou seja, os objetos são instância da classe.
- Ou seja, a classe é um modelo, e todos objetos daquela classe têm os mesmos atributos e os mesmos métodos
- A classe é compartilhável, portanto, os códigos podem ser reutilizados

# Hierarquia de classes

---

- Imagine um objeto de outra classe: cães
- Sabe-se que as classes gato e cães apresentam características comuns, uma vez que são membros de uma categoria maior: os mamíferos.
- Isso significa que na descrição de um cão ou gato pode-se omitir tudo que foi apresentado na descrição de mamífero (presença de pelos, glândulas mamárias, diafragma, dentes diferenciados, etc.)



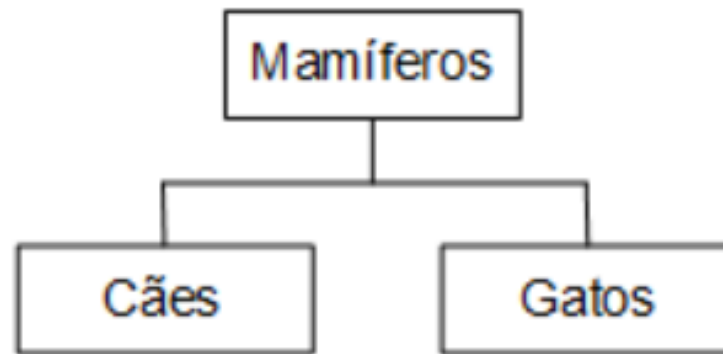


# Hierarquia de classes

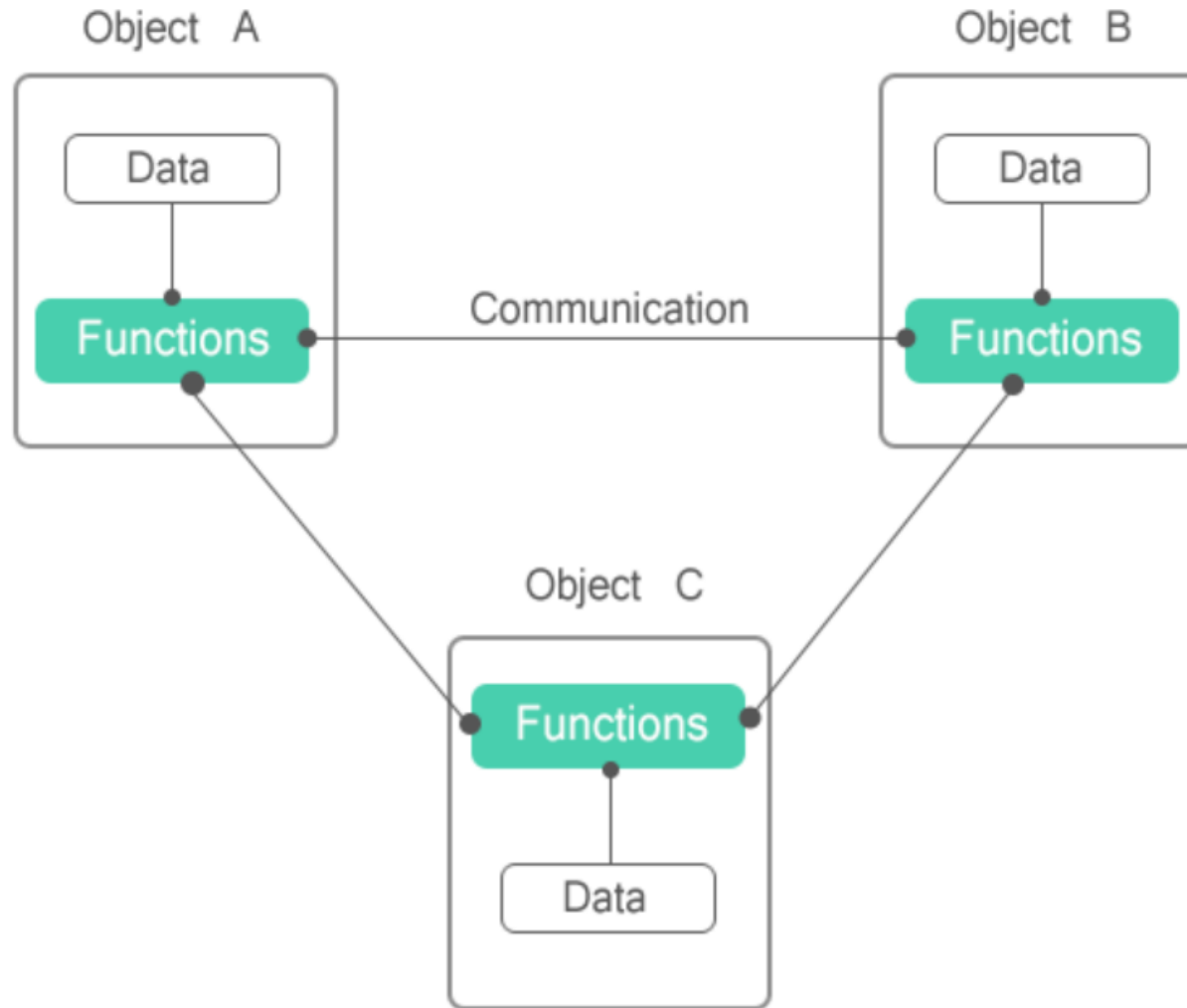
---

- Ou enxergando de outro modo:

A classe derivada herda recursos da classe base, adicionando novos recursos a ela. Isso resulta na reutilização do código.



# OO – Modelo computacional



# Linguagens OO

---

- Vantagens
  - Classes permitem uma melhor organização do projeto: **modularidade, reusabilidade e extensibilidade**;
  - Aceitação comercial crescente.
- Problemas
  - Semelhantes aos do paradigma imperativo, mas amenizadas pelas facilidades de estruturação
- Exemplos: Java e C++.

# Algoritmo

---

- Definição:

Sequência de passos que transformam uma informação de entrada em uma informação de saída.

- Três questões:

- Especificação: qual é exatamente o problema que queremos resolver?
- Correção: o algoritmo resolve mesmo o problema proposto na especificação?
- Eficiência: com que consumo de recursos computacionais (essencialmente, tempo e memória) o algoritmo executa a sua função?

# Algoritmo

---

- Existem várias formas de representar um algoritmo.
- O aprendizado de algoritmos não se consegue a não ser através de muitos exercícios.
- Para a definição de um bom algoritmo é necessário desenvolver um raciocínio lógico.
- Um algoritmo é, sob certo aspecto, um programa abstrato ou, dizendo de outro modo, um programa é a concretização de um algoritmo

# Formas de representar um algoritmo

---

- Linguagem natural
- Fluxograma
- Diagrama de Chapin
- Pseudocódigo

# Linguagem natural

---

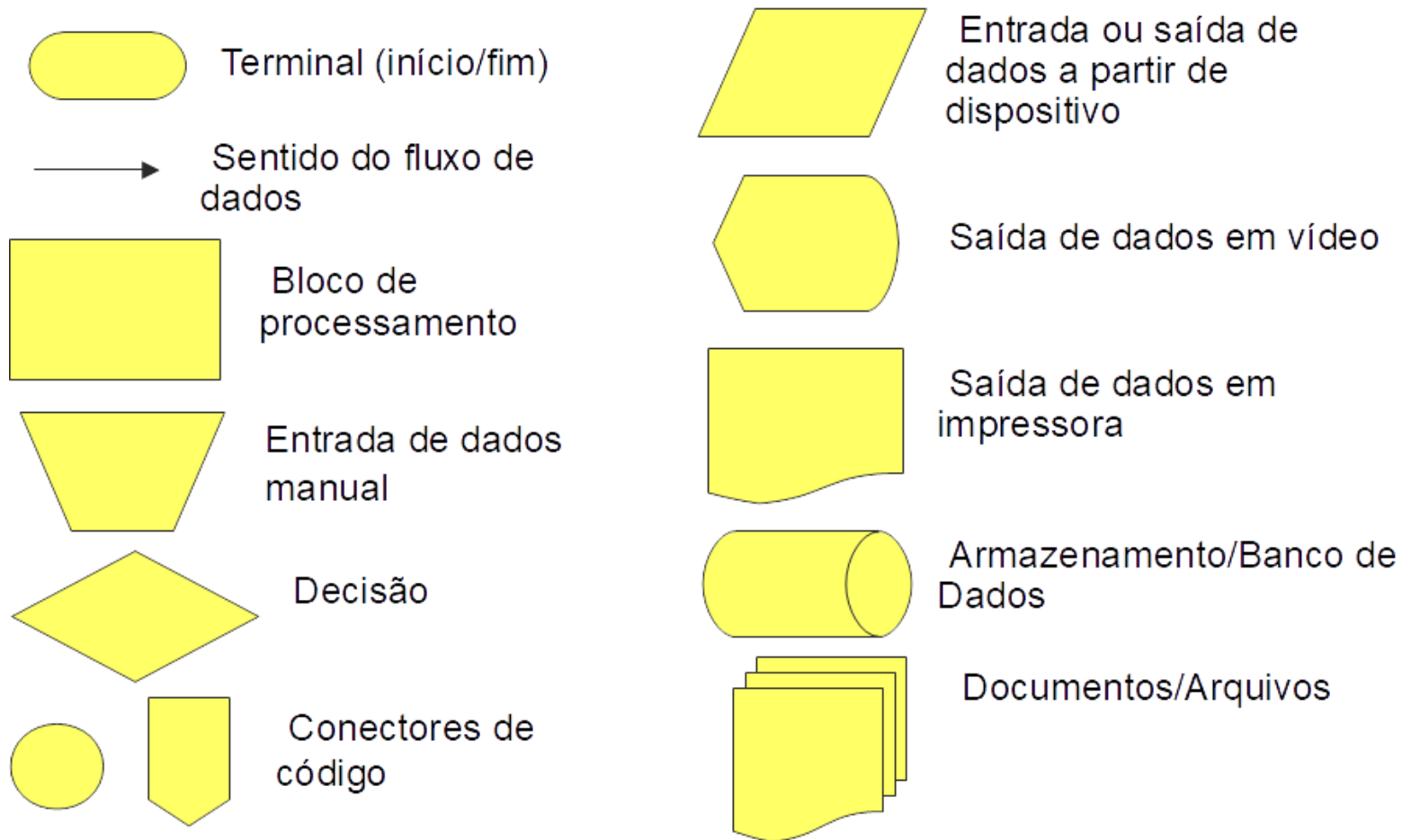
- Como trocar uma lâmpada?
  - Acionar o interruptor;
  - SE a lâmpada não acender:
    - Buscar lâmpadas novas;
    - Pegar uma escada;
    - Colocar a escada embaixo da lâmpada;
    - Subir na escada;
    - retirar a lâmpada queimada
    - colocar a lâmpada nova
    - ENQUANTO a lâmpada não acender
      - Retirar a lâmpada velha;
      - Colocar uma lâmpada nova.

# Fluxogramas

---

- Forma gráfica de representar ações e sequência lógica (ou “fluxo”) de instruções em um programa de computador.
- É uma representação gráfica de um algoritmo
- Utiliza uma simbologia padrão
  - Com algumas pequenas variações



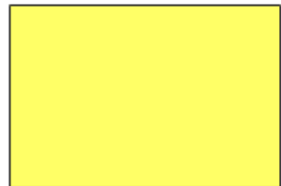


# Fluxogramas Simbologia

---



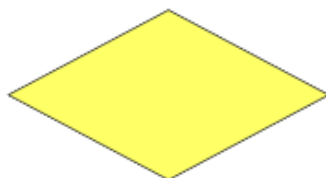
Terminal (início/fim)



Bloco de  
processamento



Entrada ou saída de  
dados



Decisão



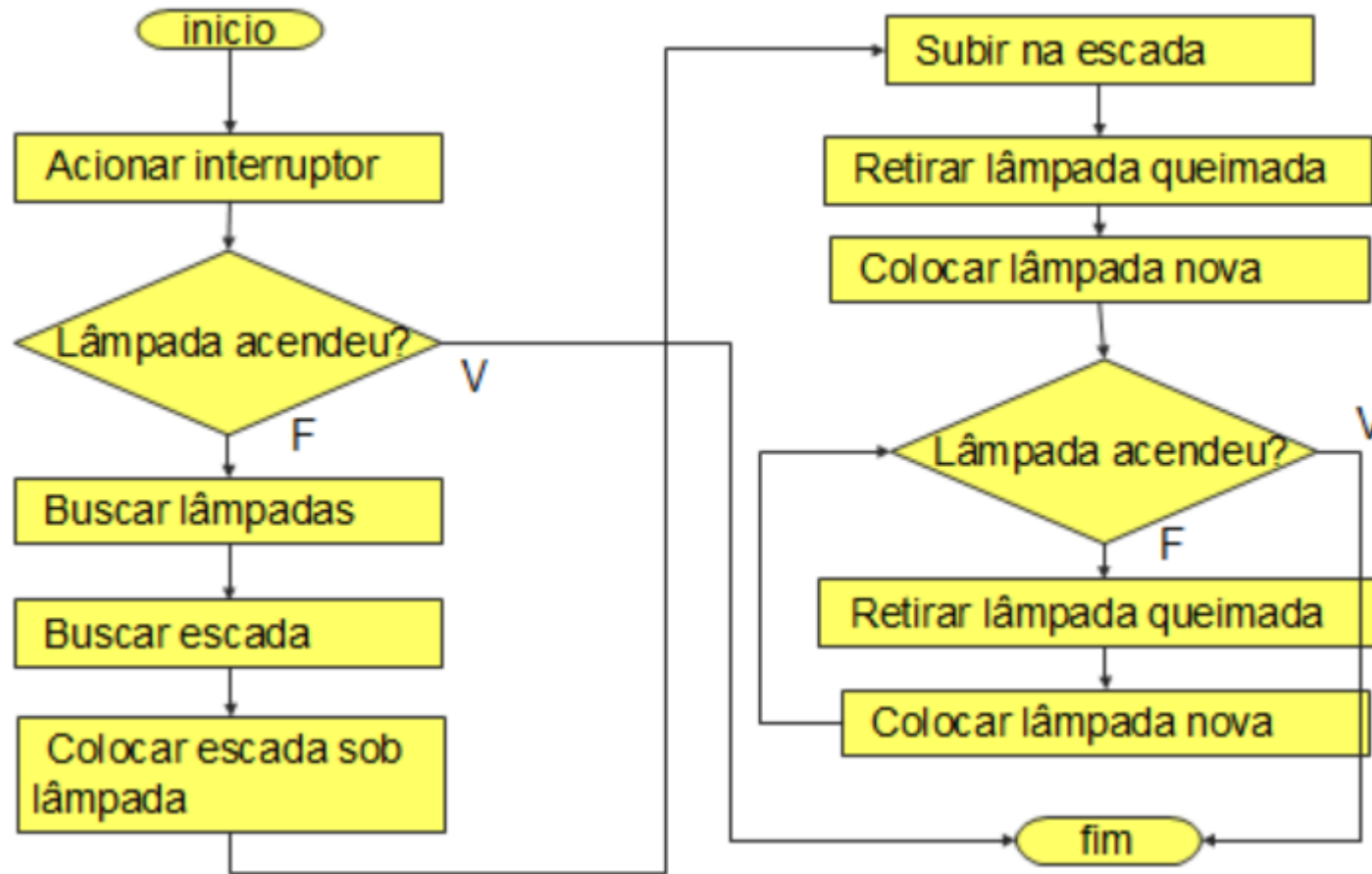
Sentido do fluxo de  
dados

## Fluxogramas — Simbologia básica

---

# Fluxogramas

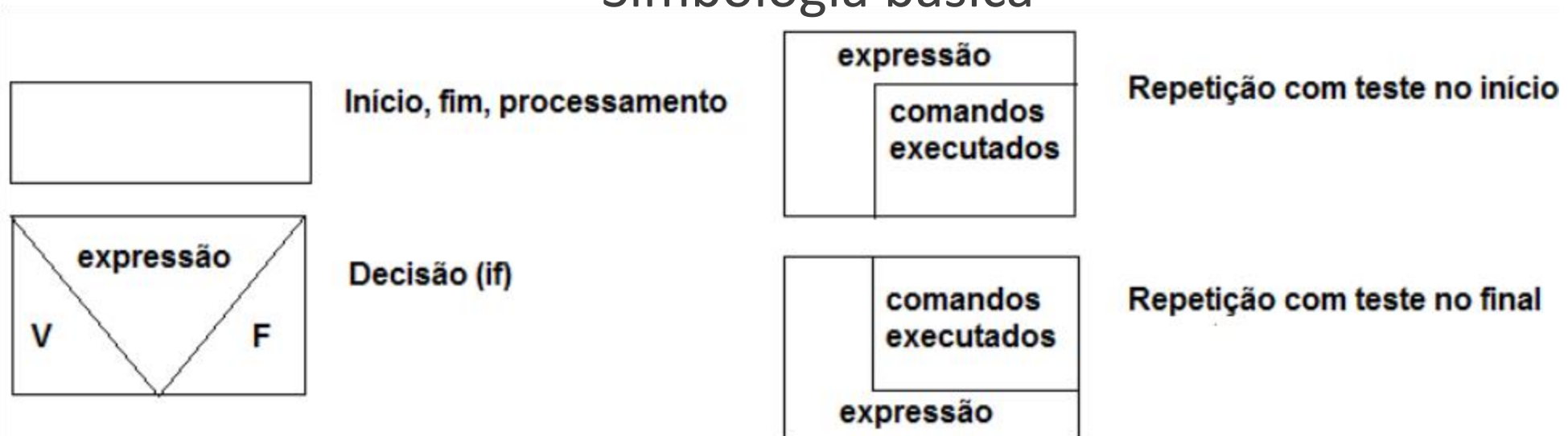
## Exemplo



# Diagrama de Nassi-Shneiderman

- Também conhecido como Diagrama de Chapin, é uma alternativa aos fluxogramas tradicionais.
- Fornece uma visão hierárquica do programa.

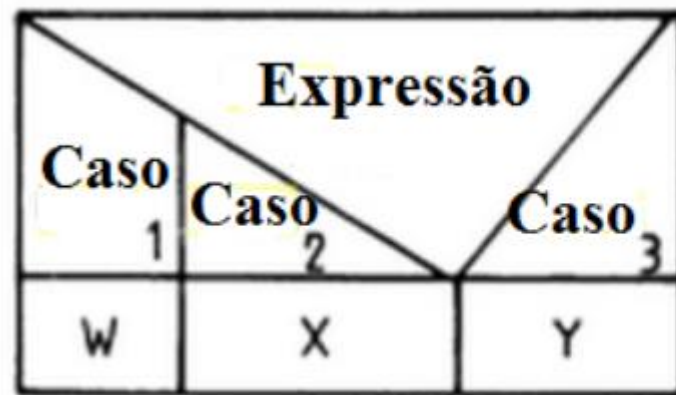
## Simbologia básica

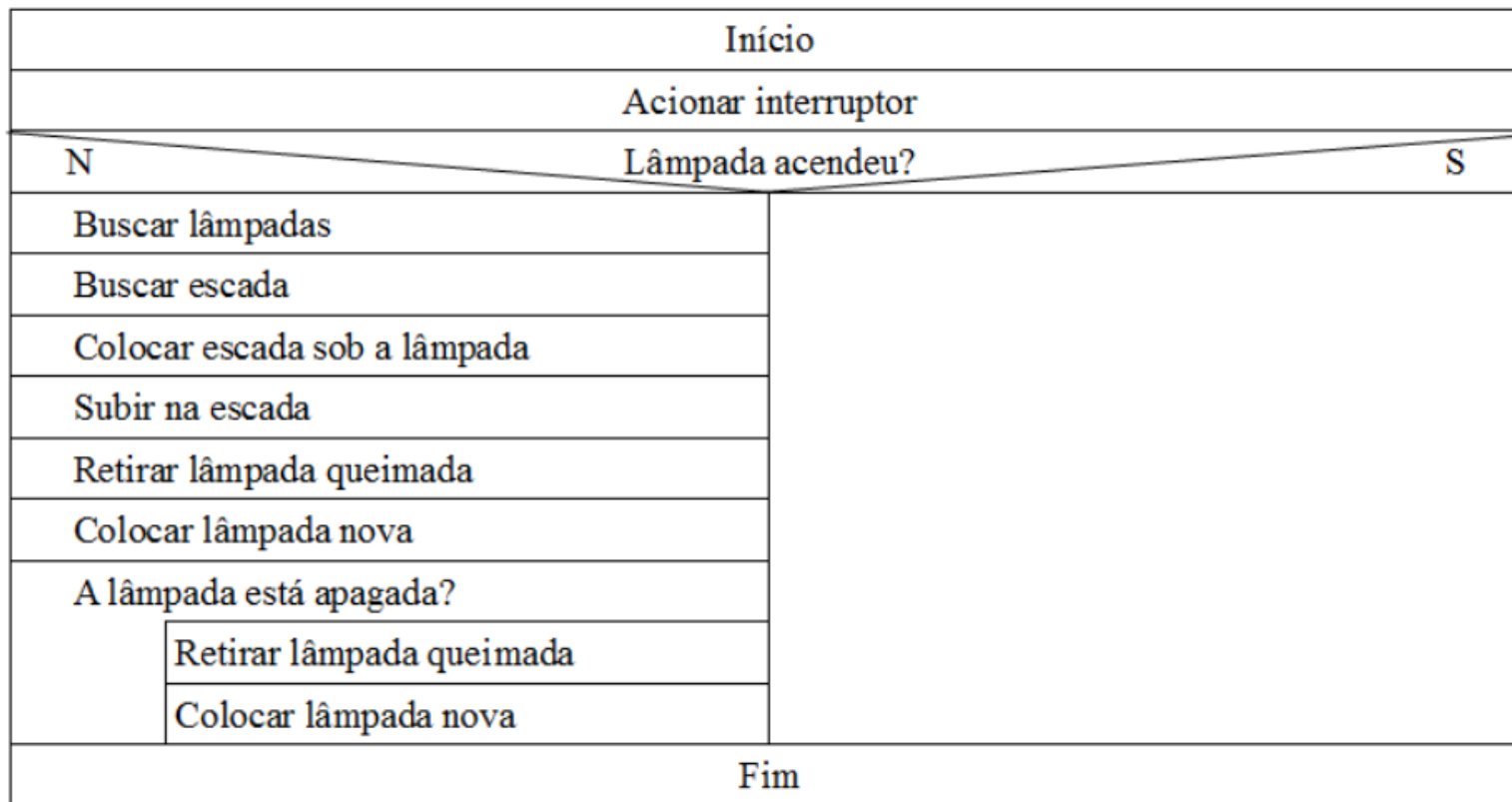


# Diagrama de Nassi-Shneiderman

---

## Simbologia básica





# Diagrama de Nassi-Shneiderman

# Pseudocódigo

- Ou português estruturado
- Rico em detalhes

**algoritmo** <nome\_do\_algoritmo>  
    <variáveis globais>  
    <subalgoritmos>  
    {Programa Principal}  
    **início**  
        <declaração\_de\_variáveis>;  
        <corpo\_do\_algoritmo>;  
    **fim**  
**fim algoritmo**

# Pseudocódigo

**algoritmo** troca lâmpada

**início**

acionar interruptor

se a lâmpada não acendeu então

buscar lâmpadas

...

colocar lâmpada nova

enquanto a lâmpada não acender faça

retirar lâmpada queimada

colocar lâmpada nova

fim\_enquanto

fim\_se

**fim**

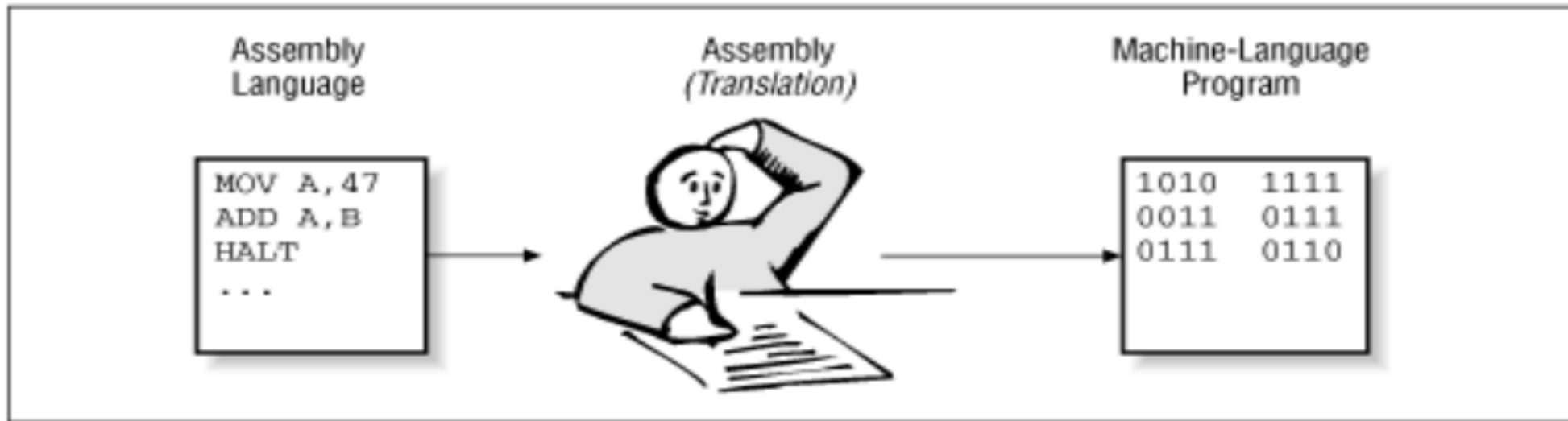
**fim do algoritmo**



# Programação

---

- Nos primórdios, só em Assembly.
- Assembly – Linguagem voltada para a máquina, ou seja, escrita utilizando as instruções do microprocessador do computador



|        |    |                |
|--------|----|----------------|
| 07100: | B0 | MOV AL, 05h    |
| 07101: | 05 | MOV BL, 0Ah    |
| 07102: | B3 | ADD BL, AL     |
| 07103: | 0A | SUB BL, 01h    |
| 07104: | 02 | MOV CX, 00008h |
| 07105: | D8 | MOV AH, 02h    |
| 07106: | 80 | MOV DL, 030h   |
| 07107: | EB | TEST BL, 080h  |
| 07108: | 01 | JZ 0117h       |
| 07109: | B9 | MOV DL, 031h   |
| 0710A: | 08 | INT 021h       |
| 0710B: | 00 | SHL BL, 1      |
| 0710C: | B4 | LOOP 010Ch     |
| 0710D: | 02 | MOV DL, 062h   |
| 0710E: | B2 | INT 021h       |
| 0710F: | 30 | ...            |
| 07110: | F6 |                |
| 07111: | C3 |                |
| 07112: | 80 |                |
| 07113: | 74 |                |
| 07114: | 02 |                |
| 07115: | B2 |                |
| 07116: | 31 |                |
| 07117: | CD |                |
| 07118: | 21 |                |
| 07119: | D0 |                |
| 0711A: | E3 |                |
| 0711B: | E2 |                |
| 0711C: | EF |                |
| 0711D: | B2 |                |
| 0711E: | 62 |                |
| 0711F: | CD |                |
| 07120: | 21 |                |

# Linguagem de baixo nível

EXEMPLO DE CÓDIGO DE MÁQUINA E ASSEMBLY X86.

# Linguagem de baixo nível

---

- Vantagens:
  - Programas são executados com maior velocidade de processamento e ocupam menos espaço na memória.
- Desvantagens:
  - Em geral, programas em Assembly não apresentam 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.

CAPÍTULO I

INTRODUÇÃO À LÓGICA  
DE PROGRAMAÇÃO E  
ALGORITMOS  
(CONTINUAÇÃO)

# Programação de Computadores I

prof. Marco Villaça

# Linguagem de alto nível

---

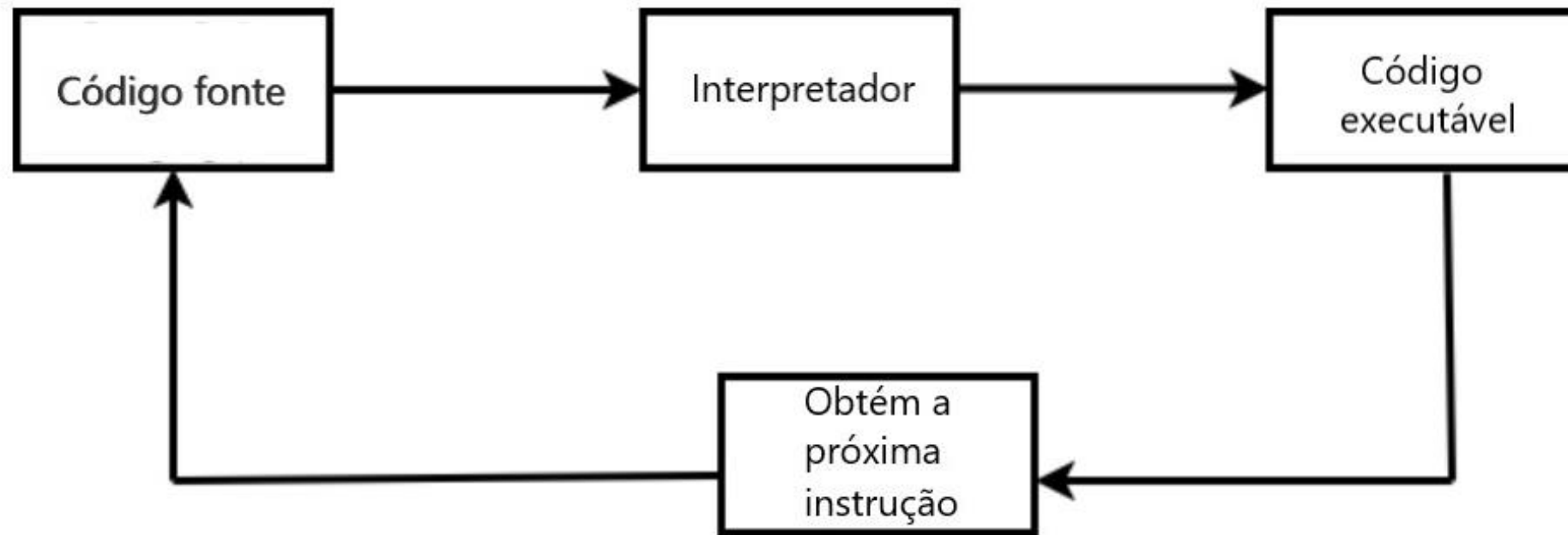
- Linguagens de alto nível
  - Mais fácil de entender (para o programador)
  - “Tradução” da linguagem feita por um compilador
  - FORTRAN, COBOL, PASCAL, C, C++, Python, Java
  - Exemplo

```
area = (base*altura)/2.0;    /*area do triangulo*/.
```

# Interpretador

---

- Em contraste com um compilador, programa que traduz um programa fonte em alto nível para um programa em linguagem assembly, o Scilab é um interpretador, isto é, um ambiente que executa outro programa fonte instrução por instrução.
- O processo de compilação é feito uma única vez.
- No modo interpretado, toda vez que se executar o código-fonte, o interpretador terá que “lê-lo” novamente.



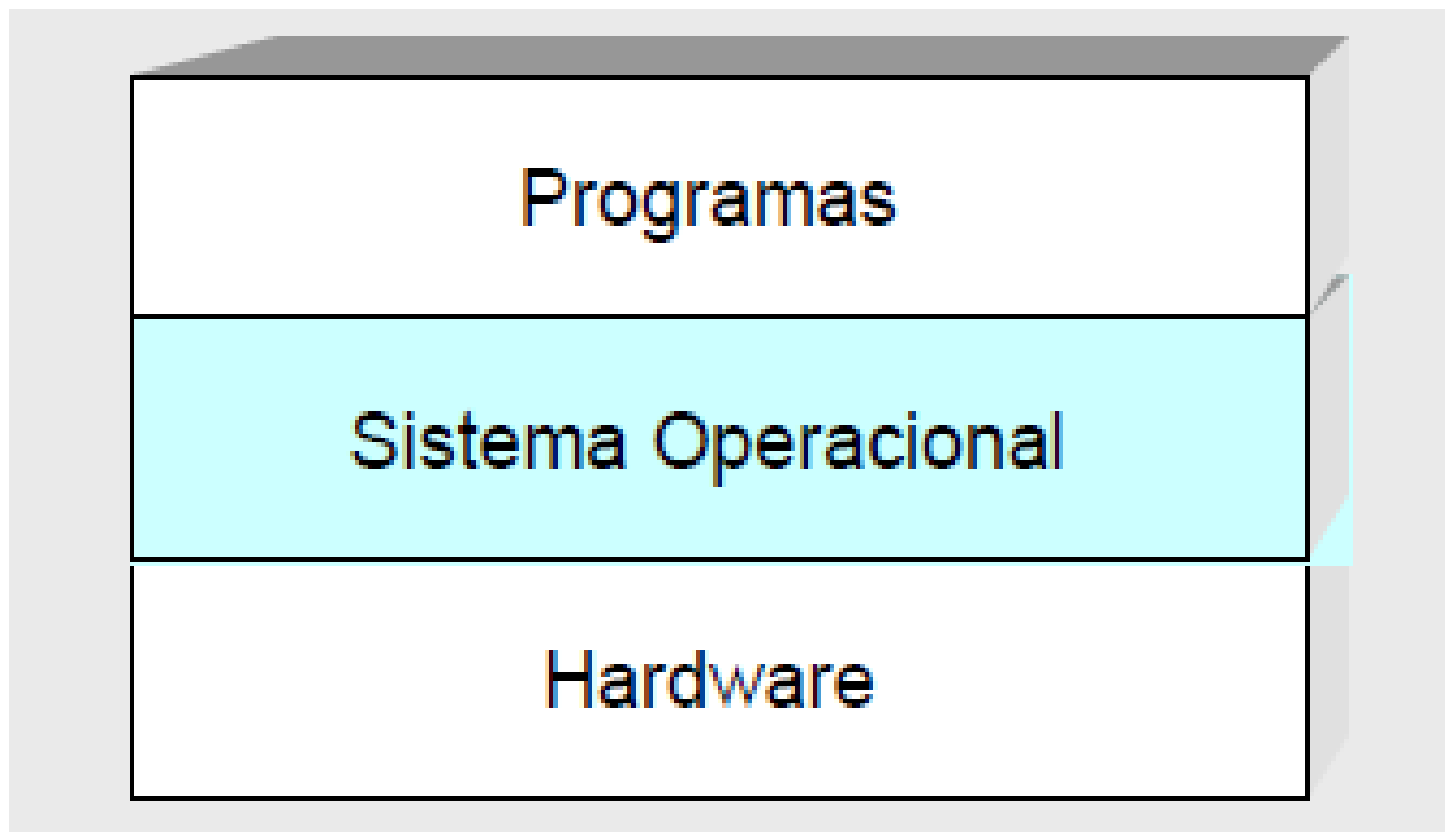
# Interpretação

---

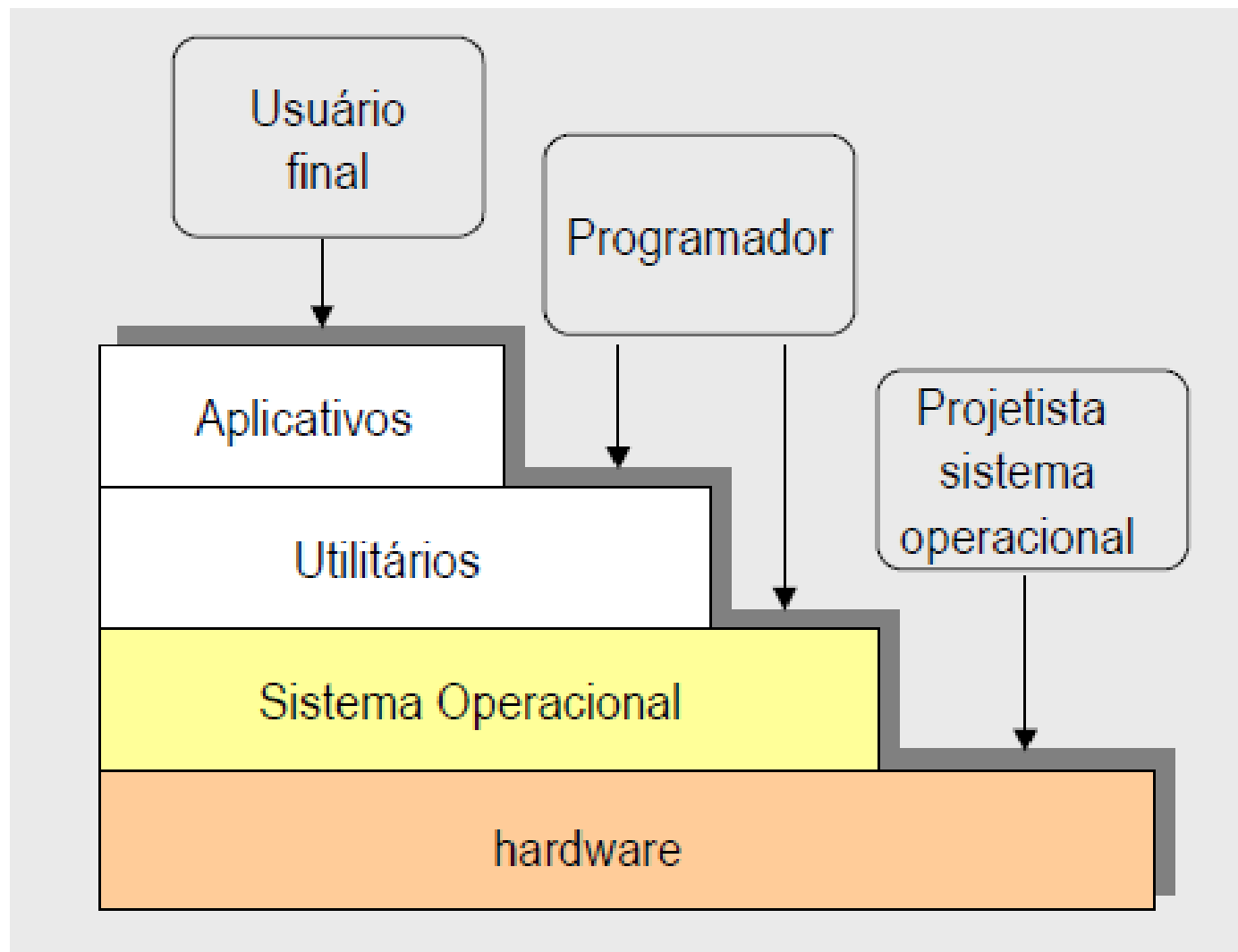
# Sistema Operacional

---

- O Sistema Operacional é um programa que permite ao usuário uma utilização mais **conveniente** (fácil) e **eficiente** (justa) dos recursos (hardware) de um sistema computacional.







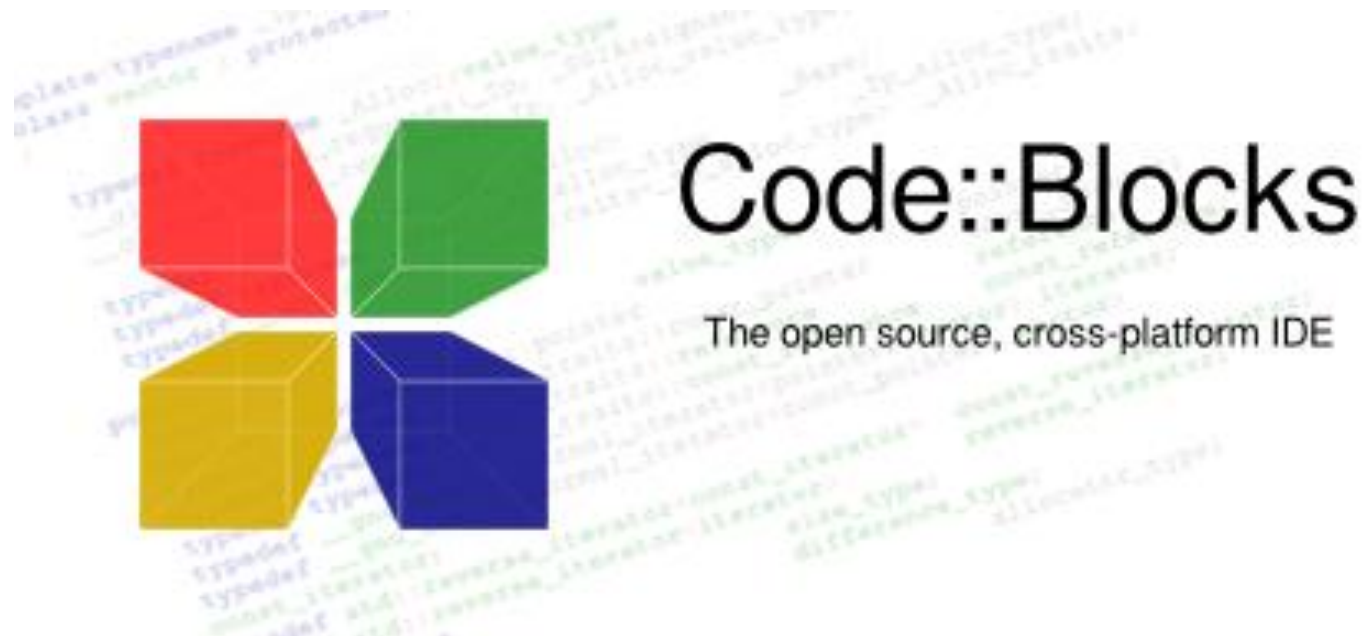
Sistema  
Operacional  
Interface entre  
usuário e sistema

---

# IDEs

---

- Ambientes de desenvolvimento (Integrated Development Environment – IDE)
  - CodeBlocks: <http://www.codeblocks.org>
  - Eclipse CDT (com compilador gcc): <http://www.eclipse.org/cdt/>
  - NetBeans IDE: <https://netbeans.org/>
  - CodeLite IDE: <http://www.codelite.org/>
  - Microsoft Visual C++ (Express Edition):  
<https://www.visualstudio.com/pt-br/>.



# Code::Blocks

---

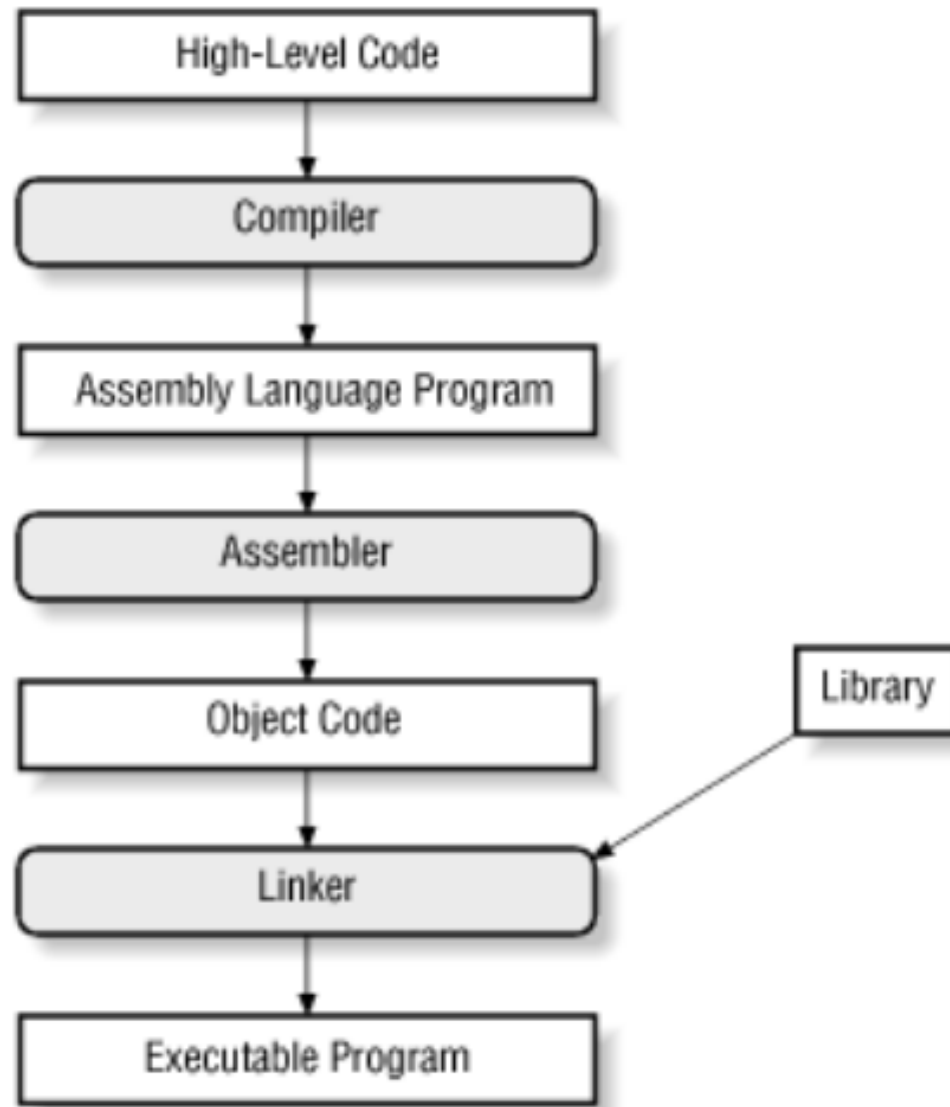
- O Code::Blocks é uma plataforma livre, de código aberto, distribuída sob a licença GPL v3.0, orientada para o desenvolvimento em C / C++ / Fortran.
- Elabora, ainda, Diagramas de Nassi-Shneiderman

# Programação

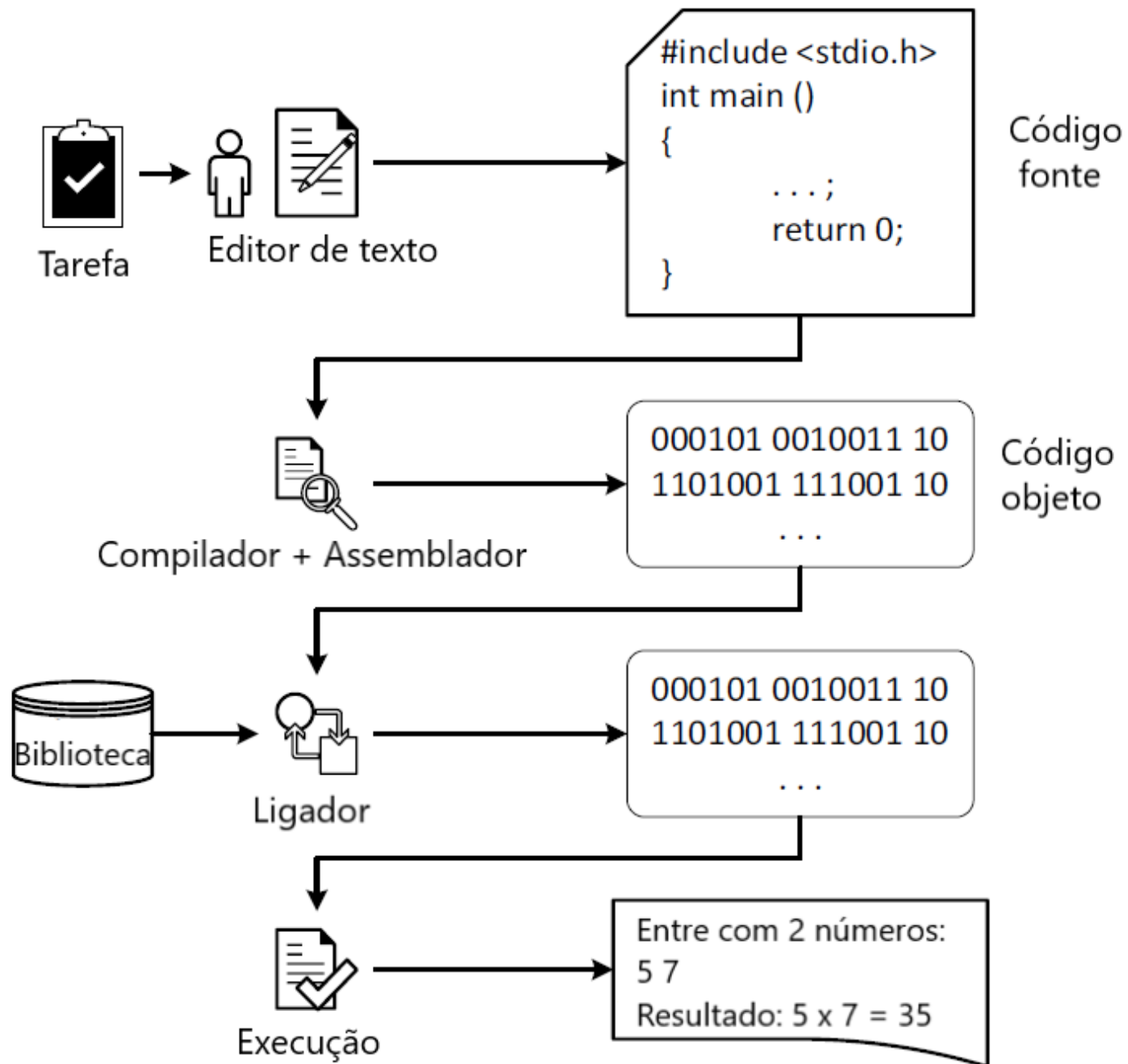
## Fluxo do processo

---

- Estudar o problema e definir uma solução
- Escrever código-fonte em um arquivo fonte
- Compilar e gerar um arquivo objeto
- Ligar (“linkar”) o arquivo objeto com rotinas de bibliotecas e gerar arquivo executável
- Se necessário, voltar ao primeiro passo



# Processo de montagem



# Processo de montagem: Diagrama Pictórico

# Estrutura de um programa C

---

- Comentários: muito importante!
- Dados: “o que vai ser usado”
- Instruções: “como usar”

```
/* ****  
* Cabeçalho / Comentários *  
**** */  
int main()  
{  
    //Declaração de dados (globais)  
    //Comandos  
    return(0);  
}
```

# Estrutura de um programa C

---

- Comentários
- Trechos (mais de uma linha): entre `/*` e `*/`
- Uma única linha: `//`
- Em C, linha de código termina com `;`
- Blocos de código são colocados entre chaves  
`{ }`



# Palavras especiais

## Palavra chave vs palavra reservada

---

- Como o nome diz, possuem significado especial em uma linguagem
- Uma palavra-chave é especial apenas em alguns contextos. Por exemplo, no Fortran:

`INTEGER Fruta` → Integer é palavra reservada

`INTEGER = 4` → Integer é o nome de uma variável

- Uma palavra reservada é uma palavra especial de uma linguagem que não pode ser usada como um nome de variável.

# Palavras reservadas em C

---

- São palavras que apresentam um significado especial na linguagem
- A seguir apresentamos as palavras reservadas do ANSI C (C89)/C90. O significado da maioria delas será apresentado ao longo do curso.

|          |        |         |        |          |          |
|----------|--------|---------|--------|----------|----------|
| auto     | break  | case    | char   | const    | continue |
| default  | do     | double  | else   | enum     | extern   |
| float    | for    | goto    | if     | int      | long     |
| register | return | short   | signed | sizeof   | static   |
| struct   | switch | typedef | union  | unsigned | void     |
| volatile | while  |         |        |          |          |

# Palavras reservadas em C

---

- Na versão C99 foram acrescentadas cinco palavras reservadas:
  - `_Bool`
  - `_Imaginary`
  - `_Complex`
  - `inline`
  - `restrict`

# O programa clássico Hello World em C

---

```
#include <stdio.h>
```

```
int main()
```

```
{
```

```
    printf("Hello World!\n");
```

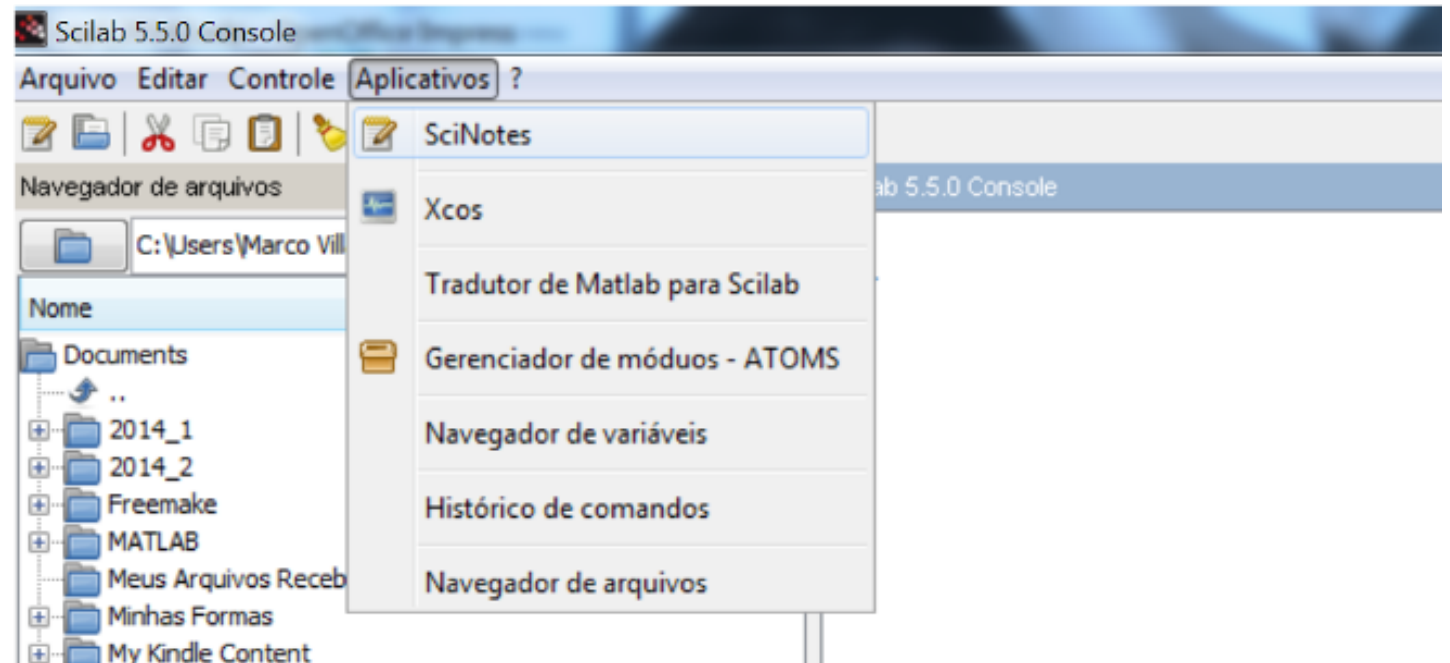
```
    return(0);
```

```
}
```

- Permite usar certos comandos; `stdio` é a abreviatura de Standard Input/Output o que significa que ela fornece funções para a entrada de dados tal como ler a partir do teclado e para a saída de dados tal com apresentar resultados na tela.
- Primeira função executada em um programa.
- As 2 chaves são usados para agrupar os comandos .
- A função `printf` “imprime” a mensagem na tela.
- A função `int main( )` deve retornar um inteiro, no caso zero.

## O programa clássico Hello World no Scilab

- Um programa fonte Scilab é um arquivo ASCII, isto é, um arquivo que só contém textos sem formatação, e que tem a extensão .sce.
- Para editá-lo usar o SciNotes.

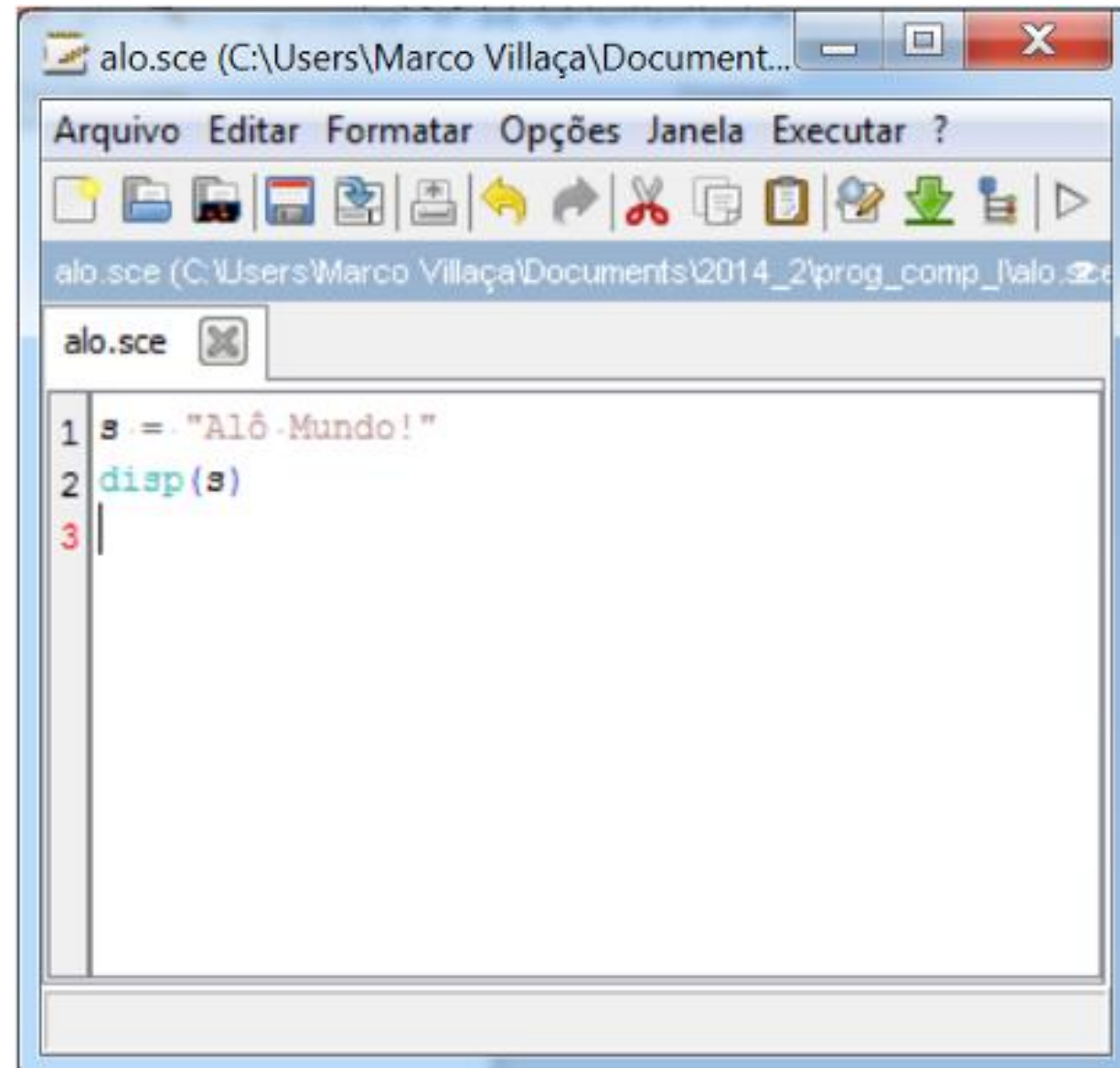


# O programa clássico Hello World no Scilab

- No editor SciNotes é possível:
  - Criar um novo programa, através do menu Arquivo/Novo;
  - Abrir para edição um programa já existente, através do menu Arquivo/Abrir;
  - Editar um programa;
  - Salvar o programa editado, através do menu Arquivo/Salvar;
  - Executar um programa, através do menu Executar/...arquivo sem eco.

## O programa clássico Hello World no Scilab

- A figura apresenta o editor durante a edição do exemplo “Alô Mundo!”:



A função `disp` exibe no console a variável `s`

# Regras de estilo

---

- Muito importante: COMENTAR o código
- Comentar é uma arte que pode ser aprendida
- Programa sem comentários pode se tornar difícil de ler
- Mas excesso de comentários também é ruim
- Pelo menos, um cabeçalho



# Regras de estilo

---

- Endentação (facilita a leitura do programador)
  - Para o compilador, é irrelevante
  - Automática nas IDE

```
for(i=0; i<=n; i++) {  
    if(!strcmp(pessoas[i].categoria, "Baixo peso"))  
        printf("%s\n", pessoas[i].nome);  
}
```

# Operadores aritméticos no C e no Scilab

---

- Multiplicação: \*
- Divisão: /
- Adição: +
- Subtração: –

# Operadores aritméticos no C e no Scilab

---

- Em C, existe o operador %
  - Resto após divisão inteira:  $5 \% 2 \rightarrow 1$
- No Scilab, a operação de potenciação é realizado por
  - $\wedge$  ou  $**$ .
- A potenciação em C é implementada pela função pow da biblioteca math
  - $3^3 \rightarrow \text{pow}(3,3)$

# Bibliografia e crédito das figuras



OUALLINE, S. Practical C Programming. 3a ed. O'Reilly, 1997.



SEBESTA, R. Conceitos de Linguagens de Programação. 5a ed. Porto Alegre: Bookman, 2003.



FORBELLONE, A. L. V.; Eberspacher, H. F. Lógica de Programação – A construção de Algoritmos e Estrutura de Dados. 2ª. Ed. São Paulo: Pearson/Makron Books, 2000.



ASHLEY, Stephen. The Fundamentals of C. 1a ed. Kindle Edition.



[http://help.scilab.org/docs/6.1.0/pt\\_BR/index.html](http://help.scilab.org/docs/6.1.0/pt_BR/index.html)



Compiler, assembler, linker and loader: a brief story. Disponível em: <http://www.tenouk.com/ModuleW.html>



KUMAR, Jema. C programming : learn to code. Boca Raton, FL : Chapman & Hall/CRC Press, 2022.