

# Projeto e Desenvolvimento de Algoritmos

Introdução

Adriano Cruz e Jonas Knopman

---

cederj

# Índice

- Objetivos
- Sucessos e fracassos da Computação
- Um pouco de história
- O Software
- O Hardware

# Objetivos

- Mostrar aspectos da história da computação
- Definir termos e palavras chaves usadas pelos profissionais da área
- Apresentar conceitos básicos sobre software e hardware

## Avanços

- Aumento de velocidade desde anos 40 foi da ordem de 100000
- Custo caiu de milhões de dólares para valores em torno de milhares
- Consumo caiu de centenas de kilowatts para apenas alguns
- Tamanho caiu de centenas de metros quadrados para menos de um metro quadrado

# ENIAC

- Considerado por muito tempo o primeiro computador programável digital



## Lei de Moore

- Em 1965 Gordon Moore, um dos fundadores da Intel, enunciou o que ficou conhecido como a lei de Moore.

**“Cada novo circuito integrado terá o dobro do número de transistores do anterior e será lançado em um intervalo entre 18 e 24 meses.”**

## Lei de Moore cont.

- Transistores são os tijolos básicos usados na construção dos microprocessadores
- Redução de tamanho dos transistores significa:
  - ❖ Menor consumo;
  - ❖ Menor tamanho;
  - ❖ Maior velocidade;

# A Família x86 e sucessores

Ano	Processador	Transistores	Ano	Processador	Transistores
1971	4004	2.250	1989	80486DX	1.180.000
1972	8008	2.500	1993	Pentium	3.100.000
1974	8080	5.000	1997	Pentium II	7.500.000
1982	80286	120.000	1999	Pentium III	24.000.000
1985	80386	275.500	2000	Pentium 4	42.000.000

## Onde parar?

- Moore achava que sua lei valeria até 1975, mas ela continua valendo até hoje
- A fronteira final é o tamanho de um elétrons, que está se aproximando rapidamente
- Intel anunciou em final de 2001 um transistor com 70 átomos de largura e 3 átomos de profundidade permitindo integrados com 1,5 Bilhões de transistores e velocidade 20 Gigahertz

# O Futuro

- Computadores paralelos, que são vários processadores cooperando para acelerar a solução do trabalho
- Computadores quânticos, armazenam informação no alinhamento e rotação dos eletrons
- Computadores biológicos, viagem completa!

## Os Fracassos

- Onde está o computador HAL do filme 2001 - Uma Odisséia no Espaço de Stanley Kubrik?
- HAL falava, via e até ficou maluco.
- Não temos nenhum neste nível, nem maluco!
- O olho de HAL



cederj

# Computadores são estúpidos!

- Picasso na sua genialidade apontou que o rei está nu e disse:

“Computadores são estúpidos, eles somente respondem perguntas.”

# E a Internet?

- A Internet, a rede das redes, será mesmo o maior e mais completo sucesso?
- Hoje faz-se tudo na Internet: namorar, comprar, estudar, comunicar-se, jogar, etc.
- Quem sabe no futuro teremos uma enorme praça virtual onde, como na Grécia Antiga, iremos discutir nossas leis?



cederj

# Evolução da Internet

MÓDULO 1  
AULA 1

<b>Meio de Comunicação</b>	<b>Anos p/ atingir 50 milhões usuários</b>
Telefone	70
Rádio	38
Televisão	13
Internet	5

**cederj**

# Distribuição da Internet

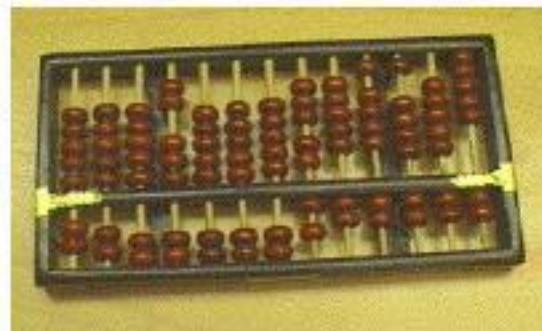
<i>Meio</i>	<i>Lançado</i>	<i>Atingiu 50 M habitantes</i>	<i>População Mundial</i>	<i>Um sistema para cada</i>
<b>Telefone</b>	1900	1970	3.8 B	76
<b>Rádio</b>	1930	1968	3.7 B	74
<b>Televisão</b>	1950	1964	3.2 B	64
<b>Internet</b>	1990	1990	5.8 B	116

## Comentários s/ Internet

- A Internet foi o meio que mais rapidamente atingiu 50 milhões de usuários
- No entanto também o meio que está pior distribuído
- Atualmente, para uso requer mais tanto em treinamento como em investimento monetário

## Pré-história

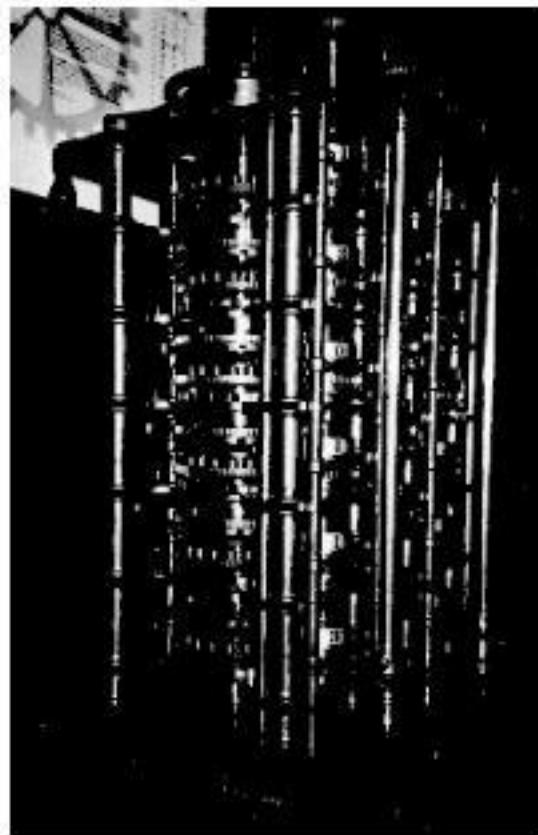
- Ábaco (2500 A.C.)
- Máquina de Calcular Mecânica (1642 - Pascal)
- Primeiro computador de uso específico (mecânico) projetado por Charles Babbage em 1812



# Charles Babbage

- Características do projeto de 1840
- 50 dígitos decimais de precisão;
- Memória para 1000 números (165000 bits);
- Controle das operações em cartões perfurados;
- Soma e subtração em 1 segundo;
- Multiplicação e divisão em 1 minuto;
- Subrotinas, arredondamento automático e detecção de transbordo (overflow);

# Charles Babbage



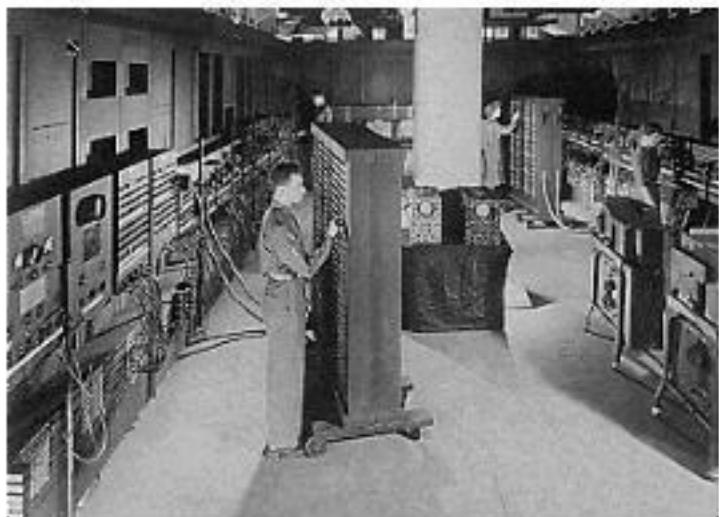
**cederj**

## Durante a 2<sup>a</sup> Guerra Mundial

- John Atanasoff: depois de um caso judicial, passou a ser considerado o construtor do primeiro computador digital (1939, Iowa State University)
- Howard Aiken: (1937-1944, Harvard University)
- George R. Stibitz: (1938-1940, Bell Telephone Labs) Primeiro a usar um computador remotamente.

## Durante a 2<sup>a</sup> Guerra Mundial

- Konrad Zuse: Computadores destruídos durante a guerra (1936-1940, Berlin Technische Hochsule)
- J. P. Eckert e J. Mauchly: (1946, Universidade da Pensilvânia) Primeiro computador digital operacional chamado de ENIAC (Electronic Numerical Integrator and Calculator). Perderam o título para John Atanassof



- **19.000 válvulas, 15.000 relés e milhares de componentes diversos**
- **42 painéis com 2,70 m de altura, 60 cm de largura e 30 cm de comprimento**
- **200 Kw de consumo, espaço especial com ar**
- **Programado por especialistas com fios**

- Electronic Delay Storage Automatic Calculator (1949), Universidade de Cambridge, Inglaterra
- Primeiro computador a usar programa armazenado na memória junto com dados
- Adeus aos programas com fios!

# Hardware

- “Hardware é o que vemos nos computadores”
- Um computador simples é composto de:
  - processador - a parte do computador onde os dados sofrem modificações;
  - memória principal - local onde o processador busca dados e instruções para operar;
  - periféricos - dispositivos usados para armazenar dados ou interagir com humanos.

# Hardware



- Um processador composto por um ou uns integrados é o microprocessador
- Um microprocessador mais memória e periféricos é o microcomputador



- Estação de trabalho é apenas um microcomputador de maior desempenho

# Bits e Bytes

- Bit é a menor unidade de informação processada pelo computador
- Bit somente pode assumir valores 0 e 1
- Um conjunto de 8 bits é o byte
- Uma palavra de memória é um conjunto de bytes, mais comum 4 bytes.

# Muitos bytes

- Em matemática kilo (k) significa  $10^3 = 1000$
- Em computação tudo está relacionado a base 2, então k é igual a  $2^{10} = 1024$
- Mega (M) igual a  $1k \times 1k = 2^{20} = 1024 \times 1024 = 1.048.576$
- Giga igual a  $1M \times 1k = 2^{20} \times 2^{10}$

# Megabytes?

- Uma memória de computador de 128 Mega bytes significa 128 vezes 1.048.576 bytes
- Em cada byte pode ser armazenado um carácter de texto
- Uma página de texto ocupa aproximadamente 3k bytes, portanto em 128 M podemos armazenar, também aproximadamente, 40.000 páginas de texto.

# Palavras e Bytes

- A memória do processador é dividida em conjuntos de bytes, as palavras
- Os tamanhos de palavras mais comuns são 2, 4 e 8 bytes
- Os computadores podem processar palavras inteiras

# Memórias e Endereços

- Como recuperar informação com tantos bytes?
- Os dados são referenciados por meio de endereços como nossas casas
- Cada palavra de memória possui um endereço único
- Dado um endereço posso escrever ou ler o seu conteúdo

## RAMs

- Random Access Memory (RAM) ou memória de acesso randômico
- RAM é uma memória que pode ser lida e escrita pelo processador com igual facilidade
- As memórias, ditas principais, dos computadores são compostas por chips de memória RAM
- A partir destas memórias, que são muito rápidas, são rodados os programas

# ROMs

- Read Only Memory, ou memória somente de leitura, armazena dados que não se modificam durante o funcionamento do computador ou quando ele é desligado
- A BIOS, que é o primeiro programa que o computador executa ao ser ligado é armazenado em ROM

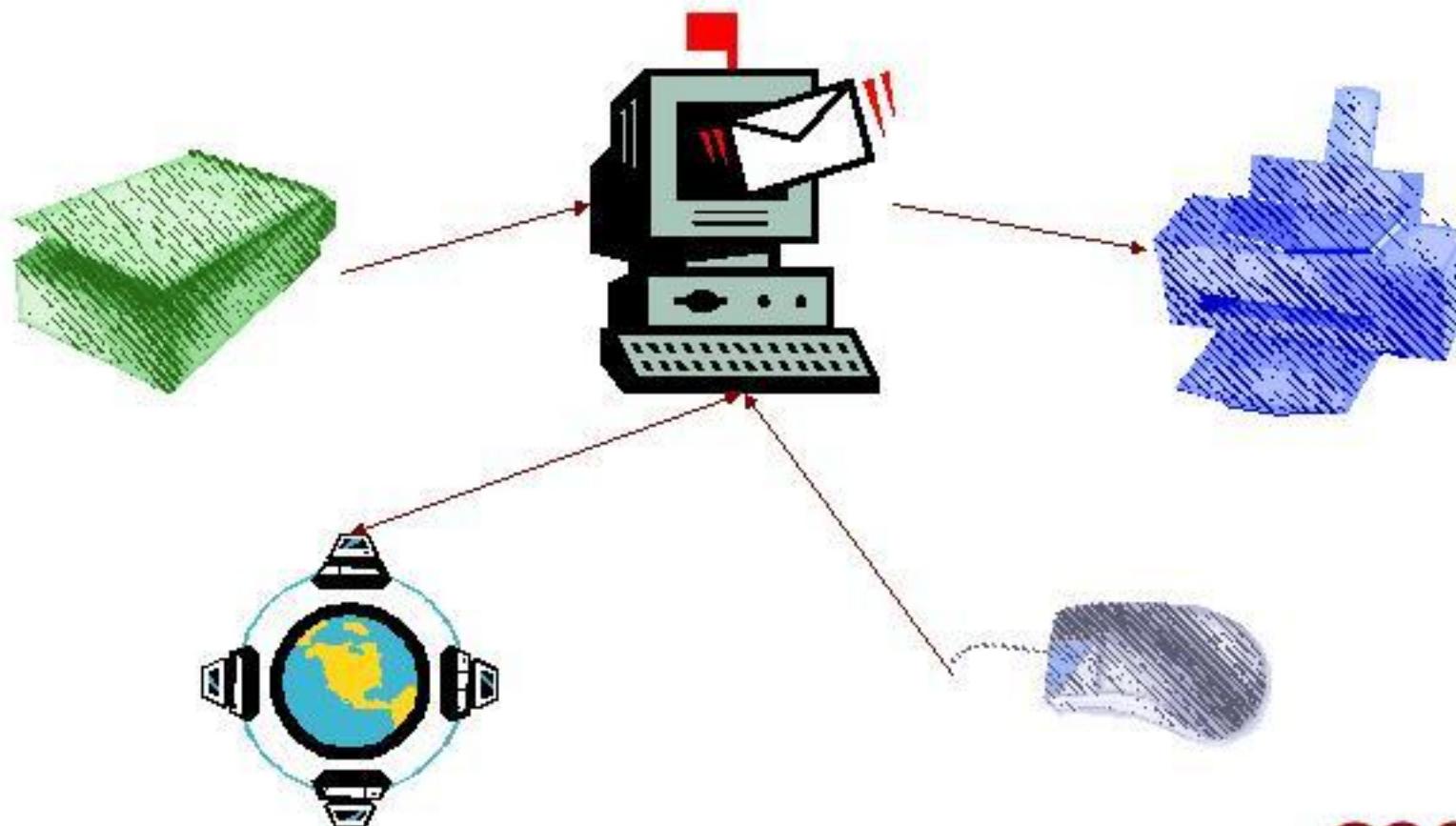
## Tipos de ROMs

- ROM gravada pelo fabricante e nunca modificada
- PROM possível de ser gravada em equipamentos especiais pelos usuários
- EPROM possível de ser gravada e desgravada
- EEPROM possível de ser gravada e desgravada eletricamente

# Periféricos

- Existem periféricos para entrada, saída e entrada e saída ao mesmo tempo.
- Periféricos de entrada de dados: teclado, mouse, joystick, CD-ROM
- Periféricos de saída de dados: vídeo, impressora, plotter
- Periféricos de entrada e saída de dados: disquetes, fitas magnéticas, discos rígidos

# Computador e Periféricos



cederj

# Software

- “Se hardware é o que vemos podemos dizer que software é o que não vemos.”
- Software engloba todos os programas que rodam no computador
- Exemplos de programas que usamos:  
editores de texto, planilhas eletrônicas, jogos,  
sistemas operacionais, correios eletrônicos e  
navegadores de internet.

# Linguagens de Programação

- Neste curso iremos aprender a desenvolver algoritmos, que são receitas indicando como resolver um determinado problema
- Este algoritmo deve ser escrito em uma linguagem que possa ser traduzida para a linguagem que o computador usa
- As linguagens que usamos para escrever os algoritmos são chamadas de linguagens de programação

# Linguagens de Programação cont

- Existem diversas linguagens de programação à nossa disposição para escrever nossos algoritmos
- Embora sejam linguagens de uso geral, há que se escolher a linguagem que melhor se adapte ao problema
- A escolha depende, entre outros fatores, do problema, do conhecimento do programador e do custo

# Linguagens de Programação exemplos

MÓDULO 1  
AULA 1

- Pascal e C usadas para desenvolver programas de uso geral e para ensino
- Delphi e C++ linguagens orientadas à objetos derivadas de Pascal e C respectivamente
- Basic o nome diz tudo, básica e simples
- Lisp e Prolog usadas em programas de IA
- Fortran, do tempo dos dinossauros, usada em engenharia e ciência
- COBOL, da mesma época, usada em programas comerciais

cederj

# Sistemas Operacionais

- Programas que gerenciam o funcionamento do computador
- Controlam quem vai usar o que por quanto tempo
- O que pode ser o processador, impressora, espaço em disco, uso de memória, etc
- Quem, são os diversos programas que usamos

# Sistemas Operacionais Exemplos

- Família Windows, produzido pela Microsoft e muito popular
- Família Unix, marca registrada do Bell Labs, usado em estações de trabalho.
- Unix é produto de diversos fabricantes por exemplo: AIX (IBM), HPUX (HP), Linux (software livre), etc

# Projeto e Desenvolvimento de Algoritmos

O que são algoritmos?

Adriano Cruz e Jonas Knopan

# Índice

- Objetivos
- Introdução
- Representação de Algoritmos
  - ❖ Linguagem Natural
  - ❖ Fluxogramas
  - ❖ Pseudo-Linguagem
- Técnicas de Projeto de Algoritmos

# Objetivos

- Apresentar o que são algoritmos.
- Formas de representação.
- Técnicas de criação e desenvolvimento de algoritmos.

# Receita

- Um algoritmo é um conjunto finito de regras que fornece uma seqüência precisa de operações para resolver um problema específico.
- Por exemplo, uma receita é um algoritmo para resolver o problema de fazer um bolo.
- Claro que a receita deve ser precisa, por exemplo, colocar sal à gosto não vale!

# Exemplos

- Todos nós sabemos criar e seguir algoritmos
- Como ir de casa até o trabalho?
- Como fazer um bife à milanesa?
- Como trocar o pneu do carro?



# Origem

- Segundo Donald Knuth, um conceituado pesquisador na criação e estudo de algoritmos, a palavra tem origem no nome de um matemático persa, que viveu no século IX, cujo nome era

Abu Ja'far Maomé ibn Mûsâ al-Khowârizm

ou

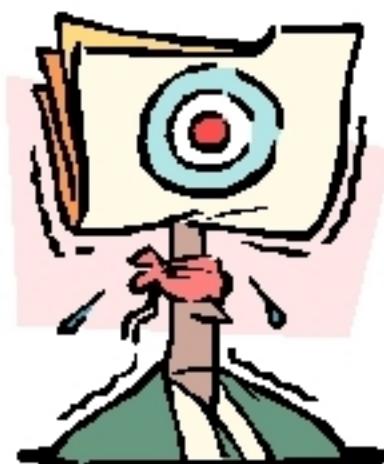
Pai de Já'far, Maomé, filho de Moisés da cidade  
de Khowârizm

# Características

- Finitude – algoritmos devem terminar após um número finito de passos;
- Definição - cada passo deve ser precisamente definido;
- Entradas - devem ter zero ou mais entradas;
- Saídas - devem ter uma ou mais saídas;
- Efetividade - todas as operações devem ser simples de modo que possam ser executadas em um tempo limitado por um ser humano.

## Dificuldades

- Pode haver mais de uma solução para um problema.
- Criação de algoritmos é um processo não automático e tem muito de arte.
- Difícil para iniciantes saber o que o computador pode ou não fazer



# Exemplo

- Como fazer para que as três rãs que estão em quatro casas nas seguintes posições,



Rã 1



Rã 2



Rã 3

- terminem assim?



Rã 3



Rã 2



Rã 1

**cederj**

# Regras para as rãs

- Somente pular para a casa da frente ou de trás se ela estiver vazia.
- Pular a rã vizinha se for parar em uma casa vazia.
- Este algoritmo serve para ordenar dados, e é muito utilizado em computação.

# Rãs se movendo 1



Rã 1



Rã 2



Rã 3

# Rãs se movendo 2



Rã 1



Rã 2



Rã 3



Rã 2



Rã 1



Rã 3

# Rãs se movendo 3



Rã 1



Rã 2



Rã 3



Rã 2



Rã 1



Rã 3



Rã 2



Rã 1



Rã 3

**cederj**

# Rãs se movendo 4



Rã 2



Rã 3



Rã 1



# Rãs se movendo 5



Rã 2



Rã 3



Rã 1



Rã 2



Rã 3



Rã 1

# Rãs se movendo 6



Rã 2



Rã 3



Rã 1



Rã 2



Rã 3



Rã 1



Rã 3



Rã 2



Rã 1

**cederj**

# Representação de algoritmos

- Linguagem natural: algoritmos expressos diretamente em linguagem natural, como nas receitas
- Fluxogramas: representação gráfica
- Pseudo-linguagem: emprega linguagem intermediária entre linguagem natural e linguagem de programação

# Linguagem Natural

- Repetir 10 vezes cada um dos quatro exercícios abaixo:
  - ❖ Levantar e abaixar o braço direito;
  - ❖ Levantar e abaixar o braço esquerdo;
  - ❖ Levantar e abaixar a perna direita;
  - ❖ Levantar e abaixar a perna esquerda.



# Fluxogramas

- Representação de algoritmos por meio de símbolos geométricos.
- Cada tipo de operação é representado por um símbolo diferente.
- Tem a vantagem de permitir o acompanhamento visual do fluxo do algoritmo

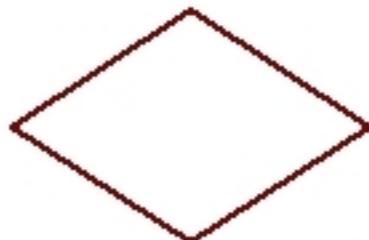
# Fluxogramas – alguns símbolos



**Início e fim de algoritmo**



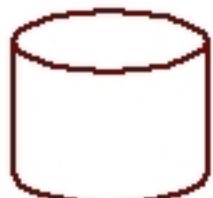
**Atribuições e cálculos de valores**



**Decisões**



**Entrada de dados**



**Discos**

# Fluxogramas – alguns símbolos



**Impressão de resultados**



**Conector na página**



**Conector fora da página**



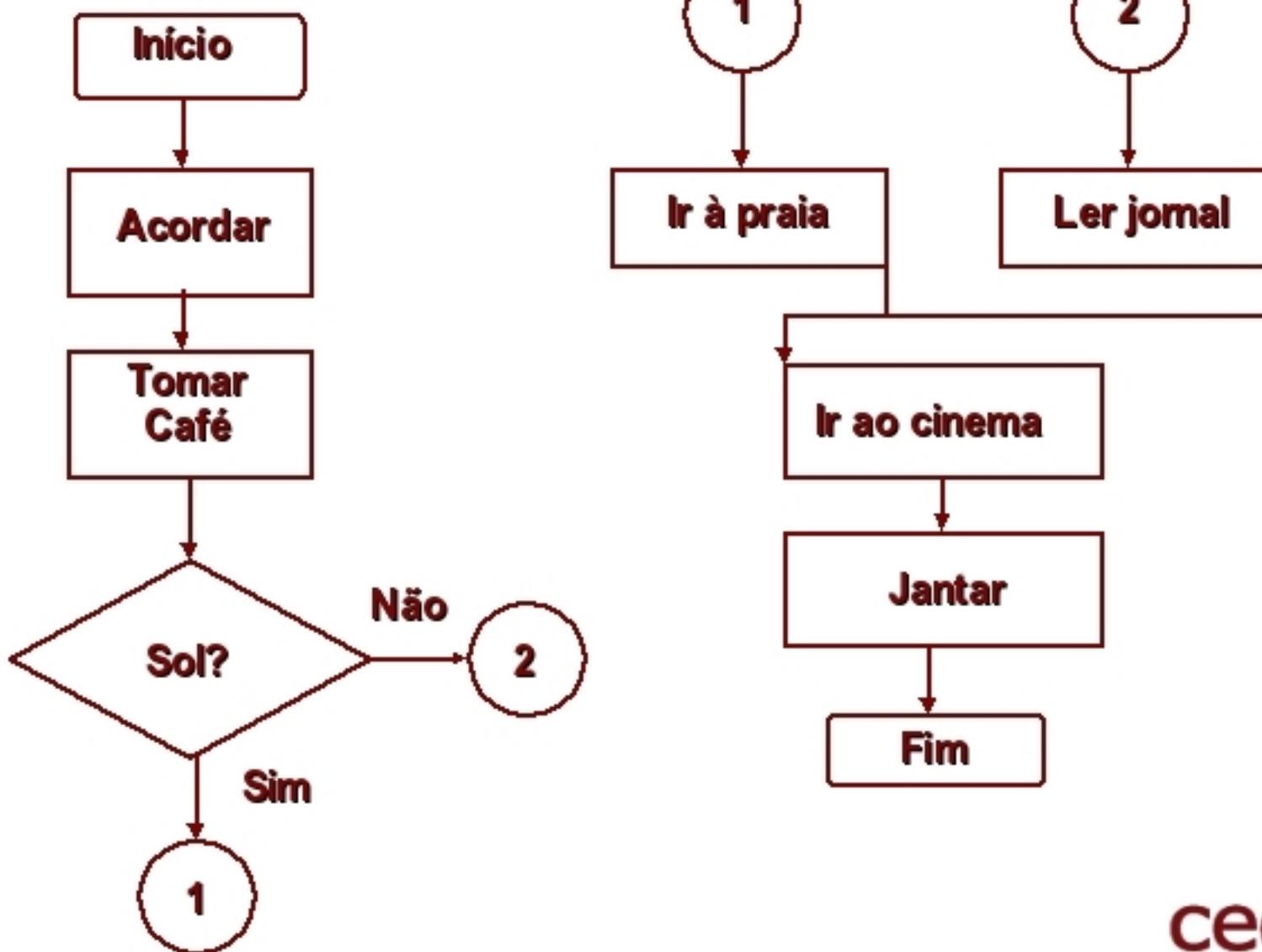
**Entrada manual**



**Ligaçāo entre símbolos**

# Um dia de sol!

MÓDULO 1  
AULA 2



# Pseudo-linguagem

- Este método será o mais empregado ao longo do nosso curso, portanto agora somente iremos mostrar alguns exemplos.
- Este método procura misturar as facilidades da linguagem natural com a precisão das linguagens de programação
- Não existe um padrão para esta forma de descrição.
- A nossa pseudo-linguagem será definida ao longo do curso.

# Exemplo de pseudo-linguagem

- Calcular a área de uma mesa retangular

## Início

**ler** comprimento

**ler** largura

área <- comprimento \* largura

**imprimir** 'Área igual a', área

## Fim

- Observe que as palavras em negrito são as palavras chaves da linguagem.

# Exemplo de pseudo-linguagem

- Calcular o preço de uma passagem de ônibus

**Inicio**

**ler** idade

**ler** preço

**se** idade < 65 **então**

**imprimir** 'Preço é ', preço

**senão**

**imprimir** 'Grátis'

**fim se**

**Fim**

# Técnicas de Construção

- Considere a receita a seguir.
- Filé de peixe com molho branco
  - ❖ {preparo dos peixes}
    - Lave os filés e tempere com o suco de limões, sal, ...
  - ❖ {preparo do molho branco}
    - Coloque em uma panela a manteiga, a farinha e o leite ...
  - ❖ {juntando os dois}
    - Adicione queijo parmesão ralado e queijo gruyère. Misture e ponha sobre os filés.
- Fim da receita de filé de peixe com ...
- Observe na receita acima **técnica de divisão do problema em partes menores** (preparo dos peixes, molho e o final)
- Permite que o prato seja preparado por mais de uma pessoa e simplifica a execução



# Reaproveitamento

- Considere a receita de alface com molho branco
- Alface com molho branco
  - ❖ {preparo do alface}
    - Derreta a manteiga. Junte alface cortada. ...
  - ❖ {preparo do molho branco}
    - Coloque em uma panela a manteiga, a farinha e o leite ...
  - ❖ {juntando os dois}
    - Junte suco de limão ao alface e ao molho branco ...
- Fim da receita de alface com ...
- Nesta receita também temos uma parte descrevendo como preparar o molho branco.
- Se o livro de receitas tiver vários pratos com molho branco há desperdício de papel.



## Reaproveitamento cont.

- Considere agora que a página 25 do livro ensine como preparar o molho branco.
- As duas receitas poderiam indicar que para preparar o molho branco o mestre cuca deve ler a página 25.
- O livro fica menor.
- Se amanhã descobrir que colocar um dente de alho melhora o sabor, somente preciso alterar a página 25 e todas as receitas ficam mais saborosas automaticamente

## Reaproveitamento cont.

- Uma solução mais radical para reaproveitar algoritmos é usar o que outros criaram.
- Assuma que você descobriu que no supermercado há um enlatado de molho branco ótimo, melhor que o seu (heresia, vale somente como exemplo!).
- Neste caso a sua receita indicaria apenas
- Compre molho branco da marca tal no supermercado (heresia novamente!)

# **Projeto e Desenvolvimento de Algoritmos**

Tipos de Dados

Adriano Cruz e Jonas Knopman

# Índice

- Objetivos
- Introdução
- Dados Numéricos
- Dados Literais
- Dados Lógicos

# Objetivos

- Apresentar os tipos de dados manipulados pelos computadores
- Mostrar as limitações do armazenamento de dados nos computadores

# **Os tipos de dados**

- Algoritmos manipulam dados fornecidos pelos usuários.
- Algoritmos devolvem dados aos usuários.
- Que tipos de dados podemos manipular?
- Algoritmos manipularão os seguintes tipos de dados:
  - ❖ Dados numéricos;
  - ❖ Dados literais;
  - ❖ Dados lógicos.

# Como representar os dados?

- A maneira de representar os dados em nossos algoritmos deverão seguir padrões rígidos.
- Linguagens de programação estabelecem regras de como os dados são escritos.
- Existem regras para indicar quais os símbolos que podem ser usados.
- Existem regras para indicar como estes símbolos devem ser combinados.

# Representação

- Para descrever a maneira como representar os dados iremos usar as seguintes convenções
- [ ] o que estiver dentro dos colchetes é opcional.
- { } o que estiver dentro das chaves deve ser repetido zero ou mais vezes.
- | usado para mostrar as opções de uma definição.

# O que usar para representar?



- Letras maiúsculas = 'A' | 'B' | 'C' | 'D' | 'E' | 'F' | 'G' | 'H' | 'I' | 'J' | 'K' | 'L' | 'M' | 'N' | 'O' | 'P' | 'Q' | 'R' | 'S' | 'T' | 'U' | 'V' | 'W' | 'X' | 'Y' | 'Z'
- Letras minúsculas = 'a' | 'b' | 'c' | 'd' | 'e' | 'f' | 'g' | 'h' | 'i' | 'j' | 'k' | 'l' | 'm' | 'n' | 'o' | 'p' | 'q' | 'r' | 's' | 't' | 'u' | 'v' | 'w' | 'x' | 'y' | 'z'
- Algarismos = '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'
- Outros = '@' | '\*' | '%' | '#' | '\$' | ',' | ';' | '.' | '?' | '>' | '<' | '-' | '+' | '=' | '^' | '|' | '/' | '\\' | '/' | '\\' | '<' | '>' | ']' | '[' | '}' | '{' | '}' | '{' | '}' | '{'

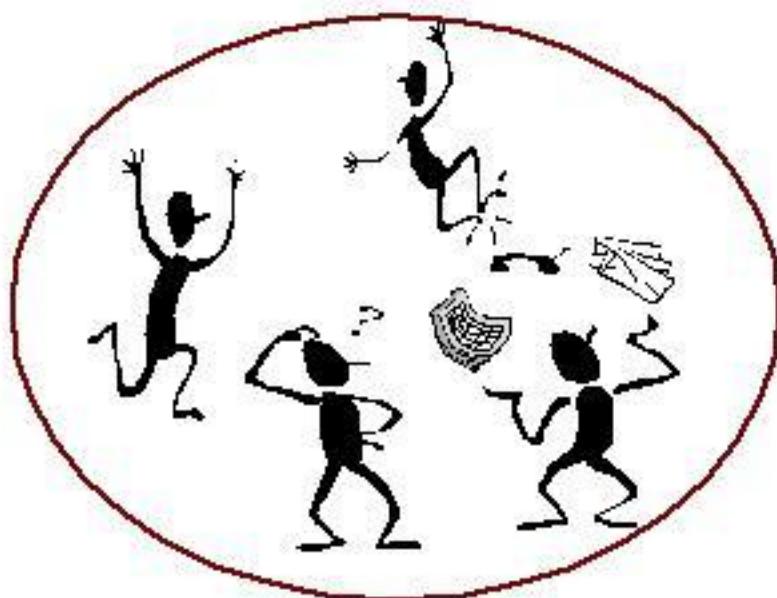
# Dados numéricos



- Antes de apresentar os dados numéricos que os computadores podem manipular vamos considerar os vários tipos que são estudados na Matemática.
- Os tipos de dados que os computadores manipulam são subconjuntos dos estudados em Matemática.

# Dados numéricos naturais

- Usados para representar
  - ❖ número de amigos
  - ❖ quantidade de CDs
  - ❖ número de ovelhas no pasto



## Dados numéricos naturais

- O número zero é recente e foi descoberto pelos hindus
- Era fácil pastores contarem com pedras (calculus) as ovelhas existentes. Mas para que calcular quando não há ovelhas.
- Em Matemática este conjunto é representado por  $N = \{0, 1, 2, 3, \dots\}$

# Dados numéricos inteiros

- O conjunto dos números inteiros é definido como

$$\mathbb{Z} = \{\dots, -3, -2, -1, 0, +1, +2, +3, \dots\}$$

- Usado para representar quantidades que podem assumir valores positivos e negativos.
- Se eu tenho 7 ovelhas e vendi 5, tenho 2 ovelhas
- Se eu tenho 5 ovelhas e vendi 7, tenho -2 ovelhas.

## Dados numéricos racionais

- Composto por todos os números que podem ser representados como uma fração da forma  $p/q$ , onde  $p$  e  $q$  pertencem ao conjunto dos números inteiros.
- Conjunto usado para representar temperaturas ( $-10,0^{\circ}\text{C}$ ), preços ( $\text{R\$ } 312,50$ ), alturas ( $1,75 \text{ m}$ ), etc.
- Pode ser representado como

$$Q = \{p/q \mid p, q \in \mathbb{Z}\}$$

## Dados numéricos irracionais

- Composto por todos os números que não podem ser representados como uma fração da forma  $p/q$ , onde  $p$  e  $q$ ,  $q \neq 0$  pertencem ao conjunto dos números inteiros.
- Exemplos são os números  $\pi = 3,1415\dots$ ,  
 $e = 2,71828\dots$
- Podemos representar este conjunto por  $\mathbb{Q}'$

# Dados numéricos reais

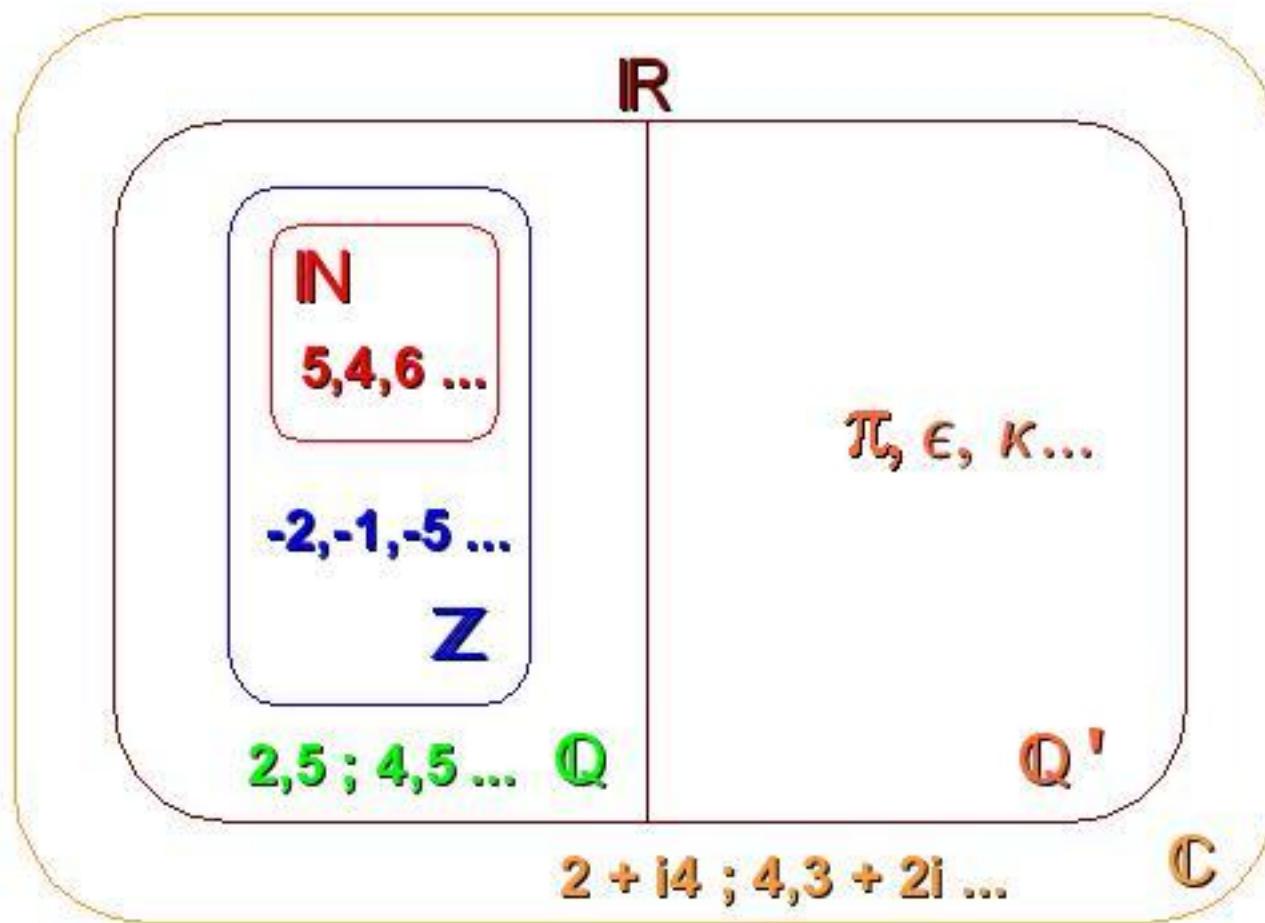
- É a união dos conjuntos dos números racionais e irracionais.
- Este conjunto é normalmente representado pela letra  $\mathbb{R}$

# Dados numéricos complexos

- Números que são representados pela forma  $a+ib$ ,
- $i$  é a raiz quadrada de  $-1$ .
- A raiz quadrada de  $-1$  é o número imaginário  $i$ .
- $a$  e  $b$  são números pertencentes ao conjunto dos números reais.
- Quando  $b=0$  o número complexo se torna um número real.



# Relações de Pertinência



# Os números e os computadores

- Computadores típicos manipulam números inteiros e reais
- Computadores trabalham internamente com números na base 2 e não na base 10 que costumamos usar.
- O número de dígitos na base 2 que um computador pode armazenar é limitado e é função da largura da palavra de memória do computador.

# Números inteiros e bits

- Um bit pode representar dois números inteiros 0 e 1.
- Dois bits podem representar quatro números inteiros diferentes: 00, 01, 10 e 11.
- Em geral com  $n$  bits podemos representar  $2^n$  números inteiros.
- Normalmente 1 bit é reservado para guardar o sinal.

## Números inteiros e bits cont.

- Considere um computador típico que use 32 bits para armazenar números inteiros.
- Reservar um bit para guardar o sinal.
- Portanto os números inteiros podem variar entre

$$-2^{31} \leq N \leq 2^{31}-1$$

- Observar que há um número negativo a mais porque não é necessário representar o número -0.

# Números inteiros

Inteiro = [ '+' | '-' ] algarismo {algarismo}

- A definição, dada pelos colchetes, diz que um número inteiro **pode ou não** ter um sinal (+ ou -).
- Em seguida deve vir obrigatoriamente um algarismo, que **pode ser ou não** seguido de outros algarismos.
- Observe que não há espaços em branco entre o sinal e os algarismos.

# Exemplos de inteiros

- +3
- 3
- -3
- -121
- +12345

## Exemplos de erros

- **+ 3** Não é permitido espaços em branco entre o sinal e o algarismo.
- **-1.0** Não é possível usar ponto.
- **-2,0** Não é possível usar vírgula.
- **3<sup>2</sup>** Expoentes não são permitidos.
- A definição deve ser seguida ao pé da letra.

# Números reais

Real = ['+' | '-']algarismo{algarismo}.algarismo{algarismo}

- Um número real pode ou não ter um sinal.
- Em seguida deve vir um algarismo seguido de zero ou mais algarismos.
- O próximo item é o ponto decimal que é obrigatório.
- Finalmente um algarismo seguido de zero ou mais algarismos.

## Exemplos de reais

- +3.0
- 3.0
- -3.0
- -121.15
- 3.1415

## Exemplos de erros

- **+ 3.0** Não é permitido espaço em branco entre o sinal e o algarismo.
- **-1.** É necessário um algarismo após o ponto.
- **-2,0** Não é possível usar vírgula.
- **.325** É necessário um algarismo antes do ponto.
- **0.7...** Dízimas periódicas não podem ser representadas desta maneira.

# Dados literais

- São usados por exemplo em:
  - ❖ Tratamento de textos;
  - ❖ Impressão de avisos aos usuários;
  - ❖ Tratamento de dados do tipo nome, endereços, etc.

# Caracteres

- Dados literais são compostos por caracteres.
- Caracteres são basicamente as letras minúsculas e maiúsculas, algarismos, sinais de pontuação, etc.
- Caracteres são representados por códigos binários.

## Caracteres - cont.

- O código mais disseminado de todos é o ASCII (American Standard Code for Information Interchange).
- ASCII usa 8 bits para representar os caracteres
- ASCII pode representar portanto até  $2^8=256$  caracteres.
- Entre os códigos há vários que são usados somente para comunicação entre computadores (ACK, NACK, EOF, EOL, etc)

## Exemplos de caracteres ASCII

- 'a' = 97
- 'z' = 122
- 'A' = 65
- 'Z' = 90
- '0' = 48
- '9' = 57
- '\$' = 36
- '+' = 43

# Dados caracteres

- Um caractere isolado em nossos algoritmos será representado pelo caractere entre 's.
- Por exemplo:
- 'a' caractere a minúsculo
- '1' caractere 1
- 'A' caractere A maiúsculo
- ':' caractere dois pontos
- """ caractere ', representado por dois ", para ser reconhecido pelo computador

# Cadeias de caracteres

- Conjuntos de caracteres, como por exemplo um nome, devem ser representados entre 's
- Por exemplo:
- '**12345**' Conjunto de algarismos
- '**Jorge da Silva**' Nome
- '**!@#\$%^&\***' Conjunto de caracteres variados



# Dados lógicos

- Aplicados no processo de tomada decisões que o computador faz.
- Este tipo de dados também é chamado de dado booleano, devido a George Boole, matemático inglês, que deu ao nome à álgebra (álgebra booleana) que manipula este tipo de dados.
- A álgebra booleana é aplicada no projeto de computadores digitais.



# Dados lógicos

- Os dados deste tipo somente podem assumir dois valores: **verdadeiro** e **falso**.
- Computadores tomam decisões, durante o processamento de um algoritmo, baseados nestes dois valores.
- Considere a questão:
  - ❖ Se saldo maior que valor do cheque então aceitar cheque senão devolver cheque.

# Dados lógicos - representação

- Os dados lógicos serão representados como:
- Verdadeiro = verdadeiro
- Falso = falso

# **Projeto e Desenvolvimento de Algoritmos**

Variáveis

**Adriano Cruz e Jonas Knopman**

# Índice

- Objetivos
- Introdução
- Modelo de Memória
- Armazenamento de Dados Numéricos
  - Dados Inteiros
  - Dados Reais
- Armazenamento de Dados Literais
- Armazenamento de Dados Lógicos
- Variáveis

# Objetivos

- Apresentar o modelo de memória do computador
- Mostrar como os diversos tipos de dados são armazenados na memória do computador.
- Apresentar o conceito de variável e sua utilidade no desenvolvimento de algoritmos.

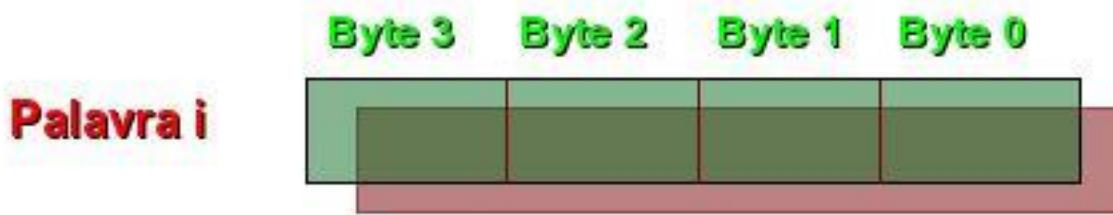
# Memória

- A memória é um conjunto ordenado de células, cada um identificada por um endereço.
- A unidade mínima de informação na memória é o bit que consegue armazenar ou o valor 0 ou 1.
- Um conjunto de 8 bits forma um byte.
- Um conjunto de bytes, usualmente 4 (32 bits) forma uma palavra de memória.



# Palavras e bytes

- Usualmente uma palavra de memória é composta de 4 bytes.



- A maioria das memórias são endereçadas por byte, portanto o endereço permite manipular o conteúdo de um determinado byte

# Memória e endereçamento

- Memória com  $2^n - 1$  palavras de 4 bytes.
- Então  $n+2$  é o número de bits do endereço.



# Memória e endereços

- Podemos pensar cada byte como uma casa.
- Uma palavra seria uma vila de casas.
- A memória uma rua de vilas.
- Suponha uma rua (memória) com 5 vilas (palavras), cada vila com 4 casas (bytes).



# Memória e endereços



# Memória e endereços

- Temos endereços de casas indo de casa 0 até casa 19.
- Temos endereços de vilas indo de vila 0 até vila 4.
- Posso dar meu endereço como casa 6 ou casa 2 da vila 1.
- Observar que  $6$  dividido por  $4$  (casas por vila) é igual a  $1$  com resto  $2$ .
- *Em computação os endereços de memória começam sempre em 0.*

## Exemplo

- Memória de 128 Mega ( $2^{20}$ ) bytes
- Memória com  $128 \times 2^{20} = 2^7 \times 2^{20} =$   
 **$128 \times 1.048.576$  bytes**
- O endereço deve ter 27 bits
  - 7 bits devido aos  $128 = 2^7$
  - 20 bits devido aos  $2^{20}$
- Se cada palavra de memória tem 4 bytes  
temos então 32 Mega palavras

# Bits e dados

- Cada tipo de dado requer uma quantidade de bits para armazenar o valor.
- Esta quantidade é variável e depende da linguagem e do computador.
- Atualmente computadores típicos possuem memórias com palavras de 32 bits, ou 4 bytes.

# Armazenamento de Inteiros

- O conjunto dos números inteiros ( $Z = \{\dots, -2, -1, 0, +1, +2, \dots\}$ ) contém uma quantidade infinita de elementos.
- Como o número de bits disponíveis na memória para armazenar os números é finito não é possível representar todos os números deste conjunto.

# Armazenamento de Inteiros

- Considere uma palavra de 32 bits.
- Um bit é reservado para o sinal.
- O bit mais significativo está à esquerda e o menos significativo à direita;
- Um número inteiro  $i$  pode variar entre

$$-2^{31} \leq i \leq 2^{31}-1$$

$$-2.147.483.648 \leq i \leq 2.147.483.647$$

- Como não temos  $+0$  e  $-0$ , há um número negativo a mais que os positivos.



# Armazenamento de Inteiros

- Observar que não podemos armazenar números maiores que 2.147.483.647 e menores que -2.147.483.648.
- Isto pode ser contornado, em alguns casos, empregando-se os números reais que veremos em seguida.

# Armazenamento de Reais

- Os números reais também são armazenados em 32 bits
- Estes números são, às vezes, chamados de números em ponto flutuante devido à forma como eles são processados.
- O método é parecido com a notação científica que algumas calculadoras empregam.



# Notação Científica

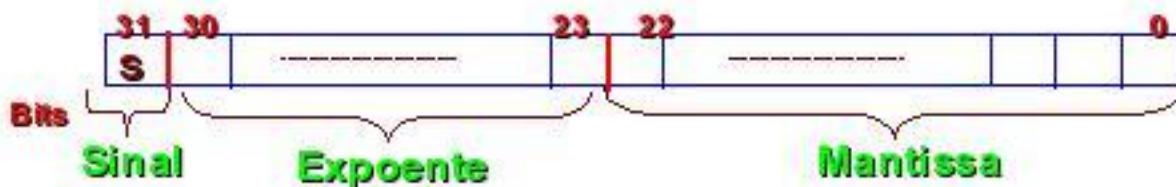
- Os números em notação científica nas calculadoras são expressos por um número real normalizado multiplicado por um número elevado a uma potência.
- Ex.  $1,5 \text{ E+8} = 1,5 \times 10^8$
- Um número normalizado implica que antes da vírgula somente deve aparecer um algarismo.
- Ex.  $(5,0 \text{ E+8}) + (6,0 \text{ E+8}) = 11,0 \text{ E+8}$
- Resultado final =  $1,10 \text{ E+9}$
- Errata do Vídeo: O correto é dizer que em  $1,5 \text{ E+8}$  há 7 zeros antes da vírgula

## Armazenamento de Reais

- O conjunto dos números reais contém uma quantidade infinita de elementos.
- Como o número de bits para armazenar os números é finito não é possível representar todos os números deste conjunto.

# Reais em bits

- Considere uma palavra de 32 bits.
- Um bit é reservado para o sinal.
- Oito bits são reservados para o expoente.
- A base do expoente depende do computador, normalmente 2.
- Vinte e três bits são usados para o número real, chamado de mantissa.



# Quais números reais?

- Computadores têm dificuldade de armazenar os números reais muito grandes e os muito pequenos.
- O subconjunto dos números reais disponíveis pode ser representado como



- O zero está incluído no conjunto.
- Para uma palavra de 32 bits temos  $R_{MAX}=3.4E+38$  e  $R_{MIN}=3.4E-38$

# Armazenamento de caracteres

- O código de caracteres normalmente empregado é o ASCII que precisa de 8 bits ou um byte.
- Como os computadores endereçam a memória por bytes então é possível armazenar um caractere por byte.
- Para armazenar um conjunto de caracteres normalmente se emprega um conjunto de bytes.

# Valores Lógicos

- Este tipo de dado somente possui dois valores verdadeiro e falso.
- Portanto um bit é suficiente para armazenar estes dados.
- Normalmente se usa um byte inteiro para armazenar valores lógicos devido a dificuldade de endereçar bits.

# Variáveis

- Diversos tipos de dados são armazenados na memória.
- A memória é endereçada por meio de números.
- Para procurar um determinado dado na memória seria preciso saber o número da palavra (ou byte) onde este dado está armazenado.
- Este método seria complicado e ilegível.
- Mais fácil empregar nomes como fazemos com as ruas de nossa cidade.

# Nomes de Variáveis

- Variáveis receberão nomes.
- Cada variável deve receber um nome diferente para poder ser identificada sem problemas.
- Estes nomes deverão ser utilizados sempre que quisermos modificar ou saber o conteúdo de uma posição na memória do computador.

# Nomes de Variáveis

- As regras para criação dos nomes das variáveis são as seguintes:
  - Um nome de variável pode conter letras, algarismos e o carácter \_ (sublinha);
  - Um nome de variável deve necessariamente começar por uma letra;
  - Um nome de variável não deve conter nenhum símbolo diferente de letra ou algarismo, exceto o símbolo \_ (sublinha)
  - Não existe limitação para o número de caracteres do nome;
  - Não será feita diferenciação entre letras maiúsculas e minúsculas.

# Dicas

- **Escolher nomes significativos para as variáveis**
  - P. Ex. salario, total, nota, pagamento
- **Nomes significativos ajudam a tornar os algoritmos e os programas auto-explicativos**
- **Nomes de variáveis com mais de uma palavra podem ajudar também**
- **Separar as palavras por sublinhados**
  - P. Ex. total\_pagamentos, prova\_final

# Dicas

- Não é necessário alongar desnecessariamente os nomes.
  - P. Ex. `total_de_recebimentos_do_ano`,  
`variavel_nota`
- Evitar nomes que não ajudem o entendimento do algoritmo.

## Exemplos de nomes corretos

- soma
- salario\_total
- nota\_final
- prova1
- raio
- velocidade\_inicial

## Exemplos de nomes incorretos

- soma\$                   **\$ não é permitido**
- salario total           **Espaço em branco não é permitido**
- 2prova                  **Não começou por uma letra**
- Salario/hora           **/ não é permitido**

# **Projeto e Desenvolvimento de Algoritmos**

**Operadores e Expressões 1**  
**Adriano Cruz e Jonas Knopman**

# Índice

- **Objetivos**
- **O que são Expressões?**
- **Constantes**
- **Tipos de Operadores**
- **Operadores Aritméticos**
- **Expressões Aritméticas**
- Operadores Relacionais
- Expressões Relacionais
- Operadores Lógicos
- Expressões Lógicas
- Expressões Mistas
- Atribuição

# Objetivos

- Apresentar os diversos tipos de operadores e expressões.
- Mostrar como as expressões devem ser escritas em pseudo-código.
- Mostrar as regras de avaliação de expressões.
- Apresentar o conceito de atribuição de resultados.

# Expressões

- Expressões combinam variáveis, operadores e constantes para produzir um resultado.
- Variáveis são nomes usados para representar posições na memória onde estão dados que serão processados.
- Constantes são símbolos usados para representar dados.
- Operadores são usados para combinar as variáveis e constantes fornecendo um valor como resposta.

# Expressões - Observações

- Nesta aula e na seguinte iremos mostrar as regras que teremos de seguir ao escrever expressões no pseudo-código usado neste curso.
- Notar que as regras variam de linguagem para linguagem de programação.
- O conjunto de regras definido para este pseudo-código permite que o aluno mude facilmente para outra linguagem de programação.

# Exemplos de Expressões

- $0.5 * \text{base} * \text{altura}$
- $(\text{nota1} + \text{nota2}) / 2.0$
- $(\text{temperatura} > 0) \text{ e } (\text{quantidade} < \text{limite})$
- $4 \bmod 3 + 5$
- $A > B$

## Expressões - observações

- Observar que as expressões são escritas sempre em uma mesma linha.
- Observar os símbolos usados para multiplicação (\*) e divisão (/).
- Avaliar primeiro as operações de maior prioridade, por exemplo (multiplicação e divisão).

## Expressões – observações

- Se temos de escolher entre operadores de mesma prioridade então escolher o que está mais à esquerda.  
Ex.  $4/2*3$  -- primeiro divide-se 4 por 2 e em seguida multiplica-se o resultado por 3, dando como resultado 6
- Caso queira trocar a prioridade use parênteses.
- Não são permitidos outros símbolos para esta função tais como { } e [ ].

# Constantes

- Constantes aparecem em expressões do tipo  
 $(lado1 + lado2) / 2$
- Nesta expressão temos a variável **lado1** somada à variável **lado2** e o resultado dividido pela constante **2**.
- Cada variável representa uma posição de memória.
- As constantes são armazenadas junto com o código do programa, não ocupando espaço da área onde estão os dados.

# Constantes

- Constantes podem ser do mesmo tipo que os dados que já estudamos e devem ser representados do mesmo modo.
- Constantes podem ser dos seguintes tipos:
  - ❖ Inteiras
  - ❖ Reais
  - ❖ Caracteres
  - ❖ Cadeias de caracteres

# Constantes Inteiras

- Constantes inteiras como já visto para os dados inteiros têm o seguinte formato:

Inteiro = ['+' | '-']algarismo{algarismo}

- Exemplos de constantes inteiras
- +256
- 128
- 32768
- 555
- 12345

# Constantes Reais

- Constantes reais, também como já visto para os dados reais têm o seguinte formato:

Real =

[‘+’ | ‘-’]algarismo{algarismo}.algarismo{algarismo}

- Exemplos de constantes reais
- 3.141516
- -22.354
- +0.567
- 128.0

# Constantes Caractere

- Constantes do tipo caractere serão representadas em nossos algoritmos pelo caractere entre 's.
- Exemplos de constantes caractere são:
  - 'a'
  - '0'
  - '+'
  - ''
  - 'A'
  - '@'

## **Constantes Cadeia de Caracteres**

- Constantes cadeias de caracteres são conjuntos de caracteres e também devem ser representados entre 's.
- Exemplos de cadeia de caracteres são:
- 'Projeto de sistemas'
- 'Pedro Silva'
- '125'
- 'Onde ela foi?'

# Constantes Lógicas

- Constantes do tipo lógica serão representadas em nossos algoritmos por verdadeiro e falso

# Operadores

- Operadores são símbolos que indicam a operação que deve ser realizada entre os operandos (constantes e/ou variáveis).
- Exemplos de operadores são: + e -

# Operadores: classificação

- De acordo com o número de operandos envolvidos na expressão, os operadores podem ser classificados em:
  - ❖ Binários, quando atuam sobre dois operandos. Exemplo: soma  
 $nota1 + nota2$
  - ❖ Unários, quando modificam um único operando: Exemplo: sinal de –  
-352

# Operadores: classificação

- Operadores também podem ser classificados de acordo com o tipo dos operandos envolvidos.
- De acordo com esta classificação os operadores podem ser divididos em:
  - ❖ Aritméticos, quando os operandos são dados aritméticos.
  - ❖ Exemplos:
    - $a + b$
    - $4.0 * \text{raio}$

# Operadores: classificação cont.

- ❖ Lógicos, quando os operandos são dados lógicos.
- ❖ Exemplos:
  - optou ou saiu
  - maior e aprovado
  - não terminou
- ❖ Relacionais, quando comparamos dados de tipos compatíveis e o resultado é um valor lógico.
- ❖ Exemplos:
  - $a > 10$
  - $x < -1$

## **Operadores: classificação cont.**

- ❖ **Caracteres**, quando os operandos são dados do tipo caractere.
- ❖ Este tipo de operador não é padronizado e varia de linguagem para linguagem.
- ❖ Em nosso estudo não utilizaremos nenhum operador de caractere.

# Operadores: classificação cont.

- ❖ Um exemplo de operação comum em várias linguagens é a concatenação de duas cadeias de caracteres.
- ❖ Símbolo + é usado em algumas linguagens para representar esta operação.
- ❖ Considere as cadeias 'dia', ' ', 'de' e 'semana'
- ❖ A operação  
**'dia' + ' ' + 'de' + ' ' + 'semana'**
- ❖ cria a cadeia  
**'dia de semana'**

# Operadores Aritméticos

<i>Operador</i>	<i>Tipo</i>	<i>Operação</i>	<i>Prioridade</i>
-	Unário	Inversão de Sinal	1
+	Unário	Manutenção de Sinal	1
/	Binário	Divisão	2
*	Binário	Multiplicação	2
<b>mod</b>	Binário	Resto da divisão inteira	2
+	Binário	Soma	3
-	Binário	Subtração	3

# Expressões Aritméticas

- Resultado é um valor numérico.
- Os operadores aritméticos mostrados na tabela anterior estão classificados por prioridade.
- Números baixos indicam maior prioridade, operações que devem ser executadas primeiro.
- Os símbolos para multiplicação e divisão mostrados na tabela são os únicos permitidos na maioria das linguagens.

## Expressões Aritméticas - representação

- $A + B * C = A + B \times C$

Em vermelho está a representação em pseudo-código.

- $4 / (x + y) = \frac{4}{x + y}$

- $b^2 - 4*a*c = b^2 - 4ac$

- $1/3*3 = \frac{1}{3} \times 3$

- $1/(3*3) = \frac{1}{3 \times 3}$

## Expressões Aritméticas obs.

- Não existem operações implícitas como em  $4ac$ , que significa 4 vezes  $a$  vezes  $c$ .
- A solução deve ser  $4^*a^*c$
- Cuidado com expressões do tipo

$$\frac{a+b}{c-d}$$

- A maneira correta é  $(a+b)/(c-d)$
- $a + b / c - d$  equivale a

$$a + \frac{b}{c} - d$$

# Exercícios

- Escreva as expressões a seguir em pseudo-código:
- $x^2 + 2by$
- $2(lado1+lado2)$
- $\frac{1}{1 + \frac{1}{x}}$

## Exercícios - Solução

- $x^2 + 2by$        $x^*x + 2^*b^*y$
- $2(\text{lado 1} + \text{lado 2})$
- $\frac{1}{1 + \frac{1}{x}}$

## Exercícios - Solução

- $x^2 + 2by$        $x*x + 2*b*y$

- $2(\text{lado 1} + \text{lado 2})$        $2*(\text{lado1}+\text{lado2})$

- $\frac{1}{1 + \frac{1}{x}}$

## Exercícios - Solução

- $x^2 + 2by$        $x^*x + 2^*b^*y$

- $2(\text{lado 1} + \text{lado 2})$        $2^*(\text{lado1}+\text{lado2})$

- $\frac{1}{1 + \frac{1}{x}}$        $1/(1+(1/x))$

## Expressões Aritméticas obs.

- Expressões aritméticas que envolvem operandos inteiros fornecem resultados inteiros.
- Expressões aritméticas que envolvem operandos reais fornecem resultados reais.
- Em operações com dados de tipos diferentes (inteiro e real) os operandos são convertidos para o tipo real.

## Expressões Aritméticas exs.

- $1 / 4$  – resultado 0
- $1.0 / 4$  – resultado 0.25
- $1 / 4 + 7.1$  – resultado 7.1
  - ❖ 1a. Operação:  $1 / 4 = 0$
  - ❖ 2a. Operacão:  $0 + 7.1 = 7.1$

# Expressões Aritméticas exs.

- $(2 + 4) / (3 - 1)$  – resultado 3
  - ❖ 1a. Operação:  $2 + 4 = 6$
  - ❖ 2a. Operação:  $3 - 1 = 2$
  - ❖ 3a. Operação:  $6 / 2 = 3$
- $10 \bmod 3$  – resultado 1
  - ❖ O resto da divisão de 10 por 3 é igual a 1.

# Exercícios

- Qual é o resultado das expressões abaixo?
- $1 / 3 * 3$
- $1.0 / 3 * 3$
- $3 + 6 / 3 - 1$
- $13 / 2 \bmod 4$

## Exercícios - Solução

- $1 / 3 * 3$  – resultado 0
  - ❖ 1a. Operação:  $1 / 3 = 0$
  - ❖ 2a. Operação:  $0 * 3 = 0$

## Exercícios - Solução

- $1 / 3 * 3$  – resultado 0
  - ❖ 1a. Operação:  $1 / 3 = 0$
  - ❖ 2a. Operação:  $0 * 3 = 0$
  
- $1.0 / 3 * 3$  – resultado 0.999...
  - ❖ 1a. Operação:  $1.0 / 3 = 0.333\dots$
  - ❖ 2a. Operação:  $0.333\dots * 3 = 0.999\dots$
  - ❖ O número de casas depende do número de bits usados para armazenar os dados do tipo real.

## **Exercícios - Solução cont.**

●  $3 + 6/3 - 1$  – resultado 4

- ❖ 1a. Operação:  $6/3 = 2$
- ❖ 2a. Operação:  $3 + 2 = 5$
- ❖ 3a. Operação:  $5 - 1 = 4$

## **Exercícios - Solução cont.**

●  $3 + 6/3 - 1$  – resultado 4

- ❖ 1a. Operação:  $6/3 = 2$
- ❖ 2a. Operação:  $3 + 2 = 5$
- ❖ 3a. Operação:  $5 - 1 = 4$

●  $13 / 2 \text{ mod } 4$  – resultado 2

- ❖ 1a. Operação:  $13 / 2 = 6$
- ❖ 2a. Operação:  $6 \text{ mod } 4 = 2$

# **Projeto e Desenvolvimento de Algoritmos**

**Operadores e Expressões 2**  
**Adriano Cruz e Jonas Knopman**

# Índice

- Objetivos
- O que são expressões?
- Constantes
- Tipos de Operadores
- Operadores Aritméticos
- Expressões Aritméticas
- **Operadores Relacionais**
- **Expressões Relacionais**
- **Operadores Lógicos**
- **Expressões Lógicas**
- **Expressões Mistas**
- **Atribuição**

# Objetivos

- Apresentar os diversos tipos de operadores e expressões.
- Mostrar como as expressões devem ser escritas em pseudo-código.
- Mostrar as regras de avaliação de expressões.
- Apresentar o conceito de atribuição de resultados.

# Operadores Relacionais

- Usados para fazer comparação entre dados compatíveis.

Operador	Comparação
=	Igual a
<>	Diferente de
<	Menor que
<=	Menor ou igual a
>	Maior que
>=	Maior ou igual a

# Operadores Relacionais

- O resultado de uma expressão que envolve operadores relacionais é do tipo lógico (verdadeiro ou falso).
- Neste curso iremos considerar que os operadores deste grupo têm a mesma prioridade entre si.

## Expressões Relacionais

- Considere  $a=3$ ,  $b=7$ ,  $total=200.0$ ,  
 $média=8.1$  e  $ano=2000$
- $a > b$  -- resultado falso
- $total = 100.0$  -- resultado falso
- $média \geq 7.0$  -- resultado verdadeiro
- $ano <> 2001$  -- resultado verdadeiro

# Exercícios

- Considerando  $a=3$ ,  $b=7$ ,  $t=20.0$  e  $m=8.1$   
qual o resultado das expressões?
- $t <> 100.0$
- $(m + b) \leq 7.0$
- $(t/(b+a)) = 2.0$

## Solução

- Temos  $a=3$ ,  $b=7$ ,  $t=20.0$  e  $m=8.1$ .
- $t <> 100.0$  -- resultado verdadeiro
- $(m + b) \leq 7.0$  -- resultado falso
  - ❖ 1a. Operação  $m+b = 15.1$
  - ❖ 2a. Operação  $15.1 \leq 7.0$  resultado falso

## Soluções cont.

- Temos  $a=3$ ,  $b=7$ ,  $t=20.0$  e  $m=8.1$ .
- $(t/(b+a)) = 2.0$ 
  - ❖ 1a. Operação:  $b+a = 10$
  - ❖ 2a. Operação:  $t / 10 = 20.0 / 10 = 2.0$
  - ❖ 3a. Operação:  $2.0 = 2.0$  portanto resultado verdadeiro

# Operadores Lógicos

- Usados em expressões cujo resultado deve ser ou o valor verdadeiro ou falso

Operando	Tipo	Operação	Prioridade
não	Unário	Negação	1
e	Binário	E lógico	2
ou	Binário	Ou lógico	3

# Operadores Lógicos

- Existem outros operadores lógicos, como por exemplo, os operadores ou-exclusivo, ne (não e), nou (não ou).
- Estes outros operadores podem ser obtidos a partir dos três já definidos (e, ou e não).
- Como os operadores aritméticos estes operadores também têm uma “tabuada”.

# Operadores Lógicos

A	B	A ou B	A e B	não A
falso	falso	falso	falso	verdadeiro
falso	verdadeiro	verdadeiro	falso	verdadeiro
verdadeiro	falso	verdadeiro	falso	falso
Verdadeiro 0	verdadeiro	verdadeiro	verdadeiro	falso

- A ou B tem como resultado verdadeiro se A ou B for igual a verdadeiro.
- A e B tem como resultado verdadeiro somente se A e B forem iguais a verdadeiro.

# Operadores Lógicos

A	B	A ou B	A e B	não A
0	0	0	0	1
0	1	1	0	1
1	0	1	0	0
1	1	1	1	0

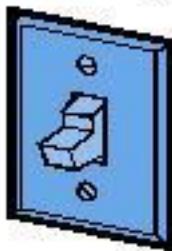
- Algumas vezes estas tabelas são mostradas com o algarismo 0 representando falso e 1 verdadeiro.

# Operadores e interruptores

- Para ilustrar os resultados dos operadores lógicos vamos usar como exemplo interruptores que devem ser usados para acender lâmpadas.

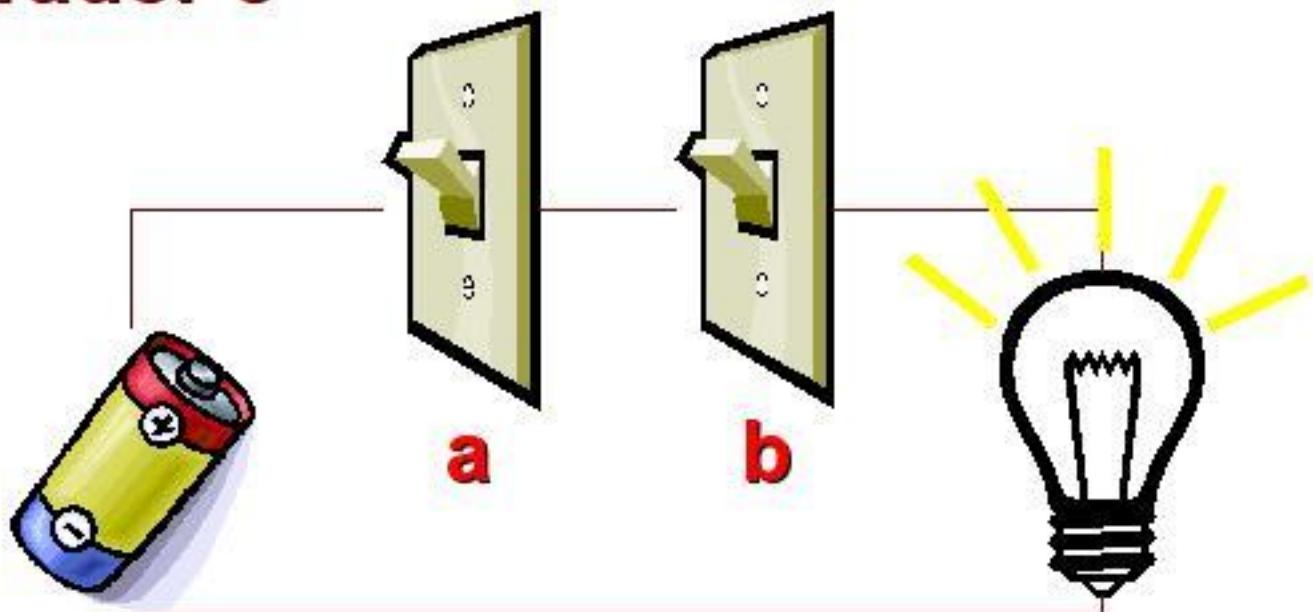


- **Interruptor ligado**



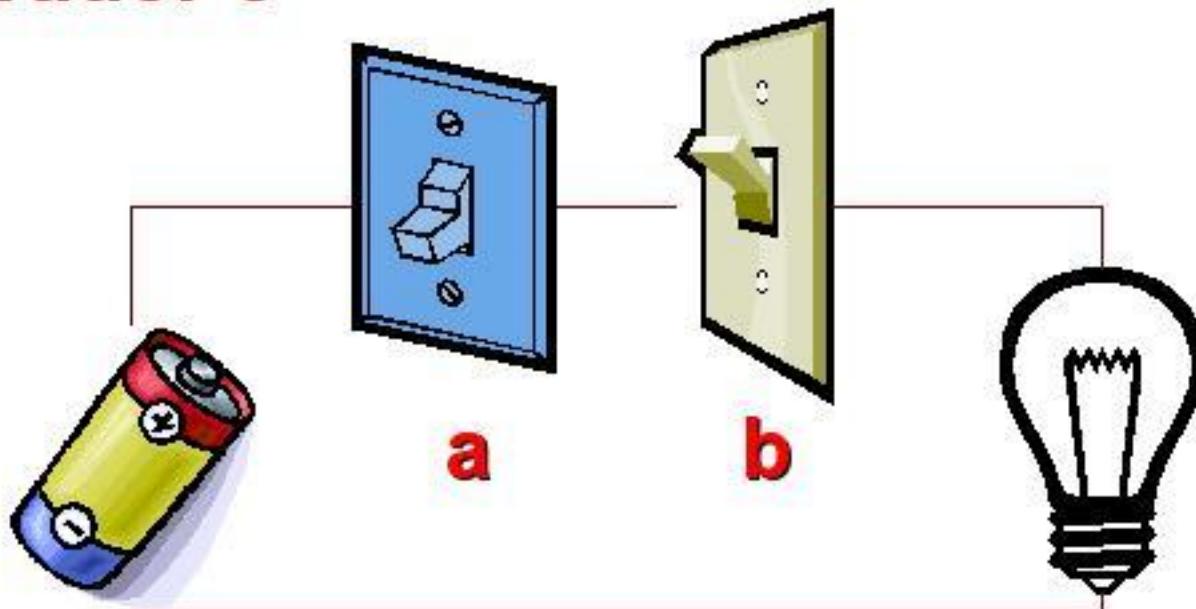
- **Interruptor desligado**

# Operador e



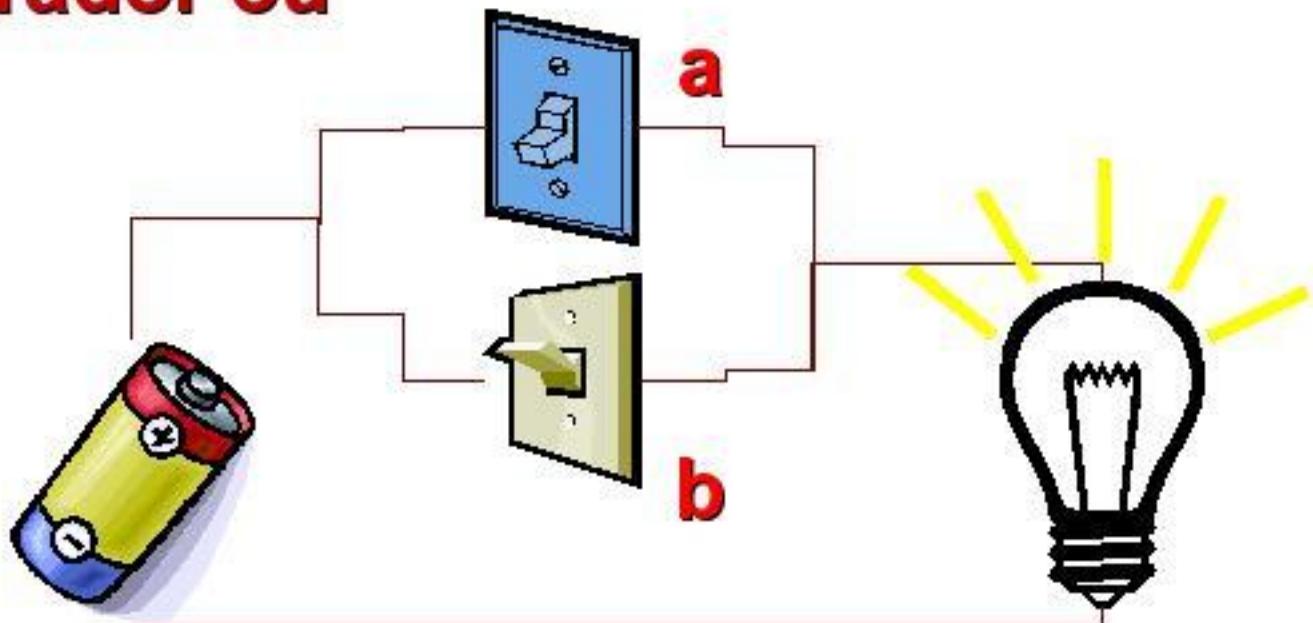
- Lâmpada acende (verdade) somente se interruptor a e interruptor b estiverem ligados (verdade).

# Operador e



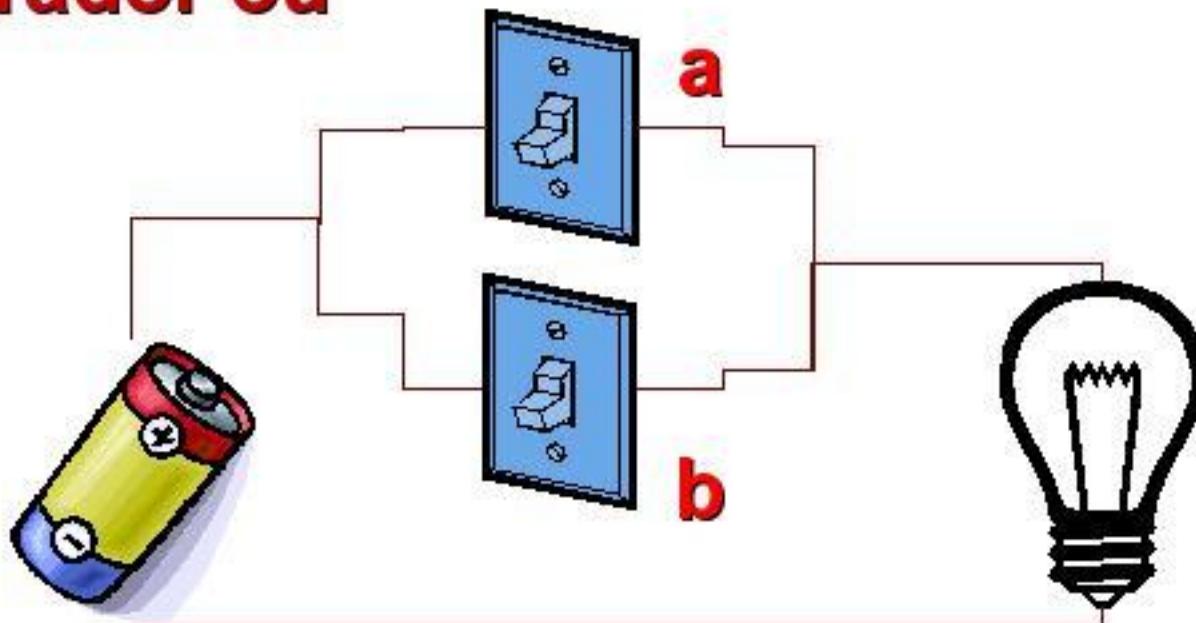
- Lâmpada apaga (**falso**) porque interruptor a está desligado (**falso**).
- Basta um interruptor desligado para que a lâmpada apague.

# Operador ou



- Para a lâmpada acender (**verdade**) basta pelo menos um interruptor ligado (**verdade**).

# Operador ou



- Para a lâmpada apagar (**falso**) é necessário que os dois interruptores sejam desligados (**falso**).

# Expressões lógicas exs.

- Considere  $a=\text{falso}$ ,  $b=\text{verdadeiro}$  e  $c=\text{falso}$ .
- $a \text{ e } b \text{ ou } c$  -- resultado falso
  - ❖ 1a. Operação:  $a \text{ e } b = \text{falso} \text{ e verdadeiro} = \text{falso}$
  - ❖ 2a. Operação:  $\text{falso} \text{ ou } c = \text{falso} \text{ ou } \text{falso} = \text{falso}$

## Expressões lógicas exs.

- Considere  $a=\text{falso}$ ,  $b=\text{verdadeiro}$  e  $c=\text{falso}$ .
- $\neg a \text{ e } b$  -- resultado verdadeiro
  - ❖ 1a. Operação:  $\neg a = \neg \text{falso} = \text{verdadeiro}$
  - ❖ 2a. Operação:  $\text{verdadeiro} \text{ e } b = \text{verdadeiro} \text{ e }$   
 $\text{verdadeiro} = \text{verdadeiro}$

# Exercícios

- Considerando  $a=\text{falso}$ ,  $b=\text{verdadeiro}$  e  $c=\text{falso}$ , qual é o resultado das expressões?
- $a \text{ e } (b \text{ ou } c)$
- $\text{não } (a \text{ e } b)$
- $a \text{ ou } b \text{ ou } c$

# Solução

- Considerando  $a=\text{falso}$ ,  $b=\text{verdadeiro}$  e  $c=\text{falso}$  temos:
- $a \text{ e } (b \text{ ou } c)$  -- resultado **falso**
  - ❖ 1a. Operação:  $b \text{ ou } c = \text{verdadeiro ou falso} = \text{verdadeiro}$
  - ❖ 2a. Operação:  $a \text{ e verdadeiro} = \text{falso e verdadeiro} = \text{falso}$

# Solução

- Considerando  $a=\text{falso}$ ,  $b=\text{verdadeiro}$  e  $c=\text{falso}$  temos:
- $\neg(a \wedge b)$  -- resultado **verdadeiro**
  - ❖ 1a. Operação:  $a \wedge b = \text{falso} \wedge \text{verdadeiro} = \text{falso}$
  - ❖ 2a. Operação:  $\neg \text{falso} = \text{verdadeiro}$

# Solução

- Considerando  $a=\text{falso}$ ,  $b=\text{verdadeiro}$  e  $c=\text{falso}$  temos:
- $a \text{ ou } b \text{ ou } c$  -- resultado **verdadeiro**
  - ❖ 1a. Operação:  $a \text{ ou } b = \text{falso} \text{ ou } \text{verdadeiro} = \text{verdadeiro}$
  - ❖ 2a. Operação:  $\text{verdadeiro} \text{ ou } c = \text{verdadeiro} \text{ ou } \text{verdadeiro} = \text{verdadeiro}$

## Expressões mistas

- É muito comum em algoritmos juntar operadores relacionais e lógicos em expressões.
- Estas expressões são geralmente do tipo  
 $(\text{nota1} > 7.0)$  ou  $(\text{nota2} > 7.0)$   
 $(\text{salario} > \text{valor})$  e  $(\text{ano} > 2001)$
- O resultado destas expressões é do tipo lógico (verdadeiro ou falso).

## Expressões lógicas+relacionais ex

- Considerando  $I1=5.0$ ,  $I2=3.0$ ,  $I3=4.0$  e  $I4=7.1$ :
- $(I1 > I3) \text{ e } (I2 > I4)$  -- resultado falso
  - ❖ 1a. Operação:  $5.0 > 4.0$  = verdadeiro
  - ❖ 2a. Operação:  $3.0 > 7.1$  = falso
  - ❖ 3a. Operação: verdadeiro e falso = falso

## Expressões Mistas ex

- É possível juntar também operadores aritméticos.
- Considere  $I1=5.0$ ,  $I2=3.0$ ,  $I3=4.0$  e  $I4=7.1$ , qual o resultado da expressão?
- $((I1+2) = I3)$  ou  $(I2 <= I4)$  -- resultado verdadeiro
  - ❖ 1a. Operação:  $5.0 + 2 = 7.0$
  - ❖ 2a. Operação:  $7.0 = 4.0$  = falso
  - ❖ 3a. Operação:  $3.0 <= 7.1$  = verdadeiro
  - ❖ 4a. Operação: falso ou verdadeiro = verdadeiro

# Exercício

- Considerando `presente=verdadeiro` ,  
`n1=7.5` e `n2=6.5`, qual é o resultado da expressão?
- `((n1+n2)/2.0) >= 7.0` e `presente`

# Solução

- Considerando **presente=verdadeiro** , **n1=7.5** e **n2=6.5** temos:
- Observe o uso de parênteses para indicar a prioridade das operações
- **((n1+n2)/2.0) >= 7.0** e **presente**
  - ❖ 1a. Operação: **n1+n2 = 7.5+6.5 = 14.0**
  - ❖ 2a. Operação: **14.0/2.0 = 7.0**
  - ❖ 3a. Operação: **7.0 >= 7.0 = verdadeiro**
  - ❖ 4a. Operação: **verdadeiro e presente = verdadeiro e verdadeiro = verdadeiro.**

# Prioridades dos Operadores

- Em expressões podemos misturar vários tipos de operadores.
- A tabela mostra a prioridade relativa dos operadores estudados.

Operador	Tipo	Prioridade
não - +	Unário	1
* / mod e	Binário	2
+ - ou	Binário	3
= <> >= <= > <	Binário	4

# Atribuição

- O resultado de expressões normalmente deve ser armazenado em uma variável para uso futuro.
- Costuma-se chamar de atribuição esta operação.
- Em nosso pseudo-código o comando de atribuição é representado pelo símbolo  $\leftarrow$
- Por exemplo:
  - ❖  $a \leftarrow 35 * 6 + 2$
- O comando acima faz com que o resultado da expressão ( $=212$ ) seja armazenado na variável  $a$ .

# Atribuição - semântica

- Todo comando de atribuição pode ser dividido em duas etapas:
  2. Avaliação da expressão;
  3. Armazenamento do resultado da avaliação na posição de memória representada pela variável.

# Atribuição e memória

- Considere  $x=10$ ,  $y=5$  e  $z=8$ .
- As figuras abaixo mostram a memória antes e depois do comando de atribuição

$$x \leftarrow y + z$$

End	antes	
0	10	x
1	5	y
2	8	z
3		
4		
5		

End	depois	modificação
0	13	
1	5	
2	8	
3		
4		
5		

---

# **Projeto e Desenvolvimento de Algoritmos**

Entrada e Saída de Dados  
Adriano Cruz e Jonas Knopman

cederj

# **Índice**

---

- **Objetivos**
- **Comandos de Entrada de Dados**
- **Comandos de Saída de Dados**

# **Objetivos**

---

- Apresentar os comandos de entrada e saída de dados
- Mostrar como formatar a saída de dados para facilitar o entendimento dos resultados

# **Entrada e Saída de Dados**

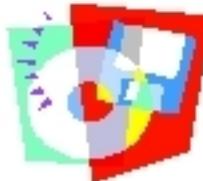
---

- Algoritmos úteis normalmente devem receber dados para processar (entrada de dados)
- Algoritmos devem devolver os resultados das transformações executadas nestes mesmos dados (saída de dados)

# **Entrada de Dados**

---

- Em um programa os dados podem ser lidos de um teclado, de um arquivo no disco rígido, disquete, etc.
- Abaixo mostramos exemplos de periféricos de entrada que em alguns casos podem também ser de saída.
- Você é capaz de dizer qual (ou quais) destes periféricos é (são) de entrada e saída?

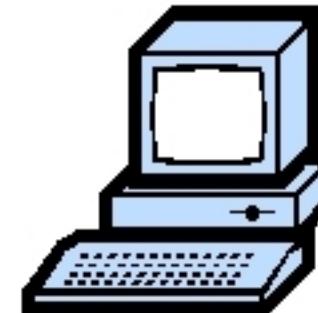


**cederj**

# **Saída de Dados**

---

- Os dados de saída podem ser escritos em uma impressora, terminal de vídeo, disquete, disco rígido etc.



# **Entrada e Saída de Dados**

---

- No pseudo-código que estamos usando a origem e o destino dos dados não é importante.

# **Fluxos de dados**

---

- Várias linguagens consideram que os algoritmos (programas) recebem dados de um fluxo de dados de entrada e enviam os seus dados para um fluxo de dados de saída
- Estes fluxos são organizados como seqüências de linhas de caracteres
- Que periféricos geram e recebem estes fluxos é dependente do programador e da implementação e configuração do compilador

**cederj**

# **Fluxos de dados cont.**

---

- Na maior parte dos casos estes fluxos são lidos de maneira semelhante a uma fita cassette áudio sem *rewind*.
- O computador avança lendo os dados na ordem em que são fornecidos pelo usuário.
- Devemos então considerar que uma vez lido um dado o computador avança para o próximo e não existe a possibilidade de retorno.



# Fluxos de dados cont.

- A separação entre os dados é feita por um ou mais espaços em branco.

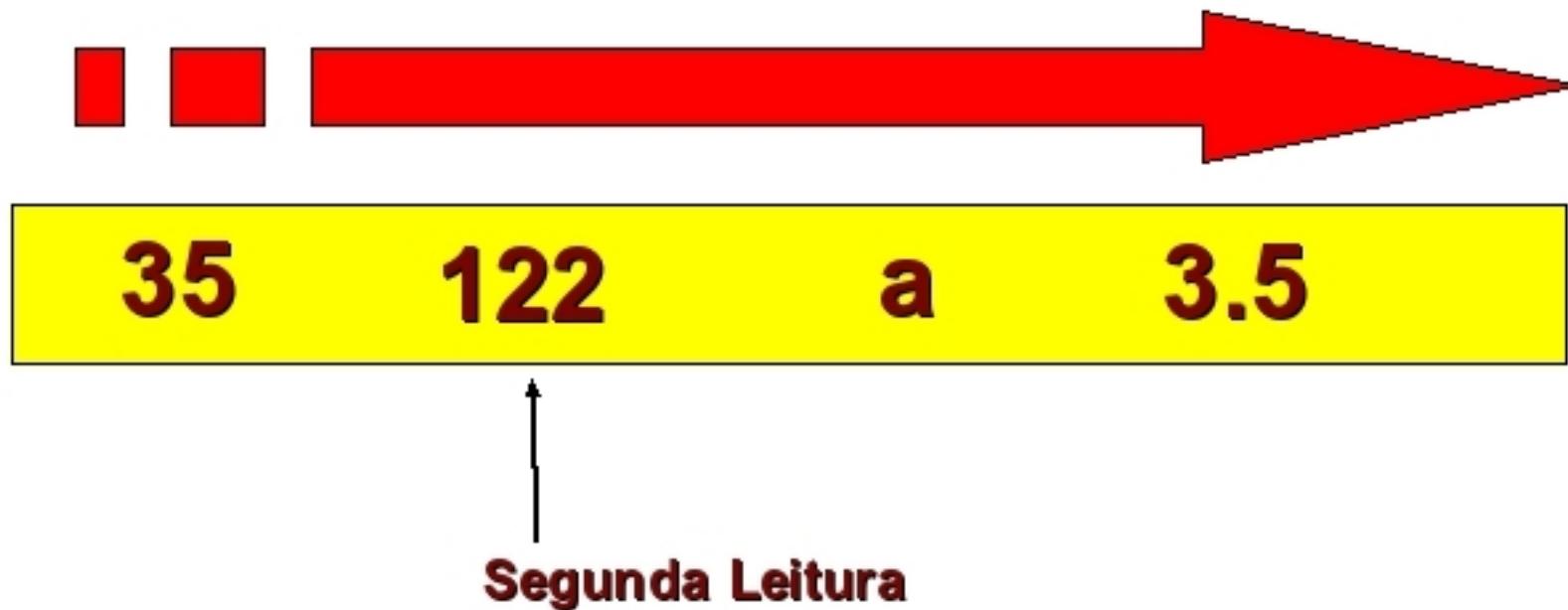


<b>35</b>	<b>122</b>	<b>a</b>	<b>3.5</b>
-----------	------------	----------	------------

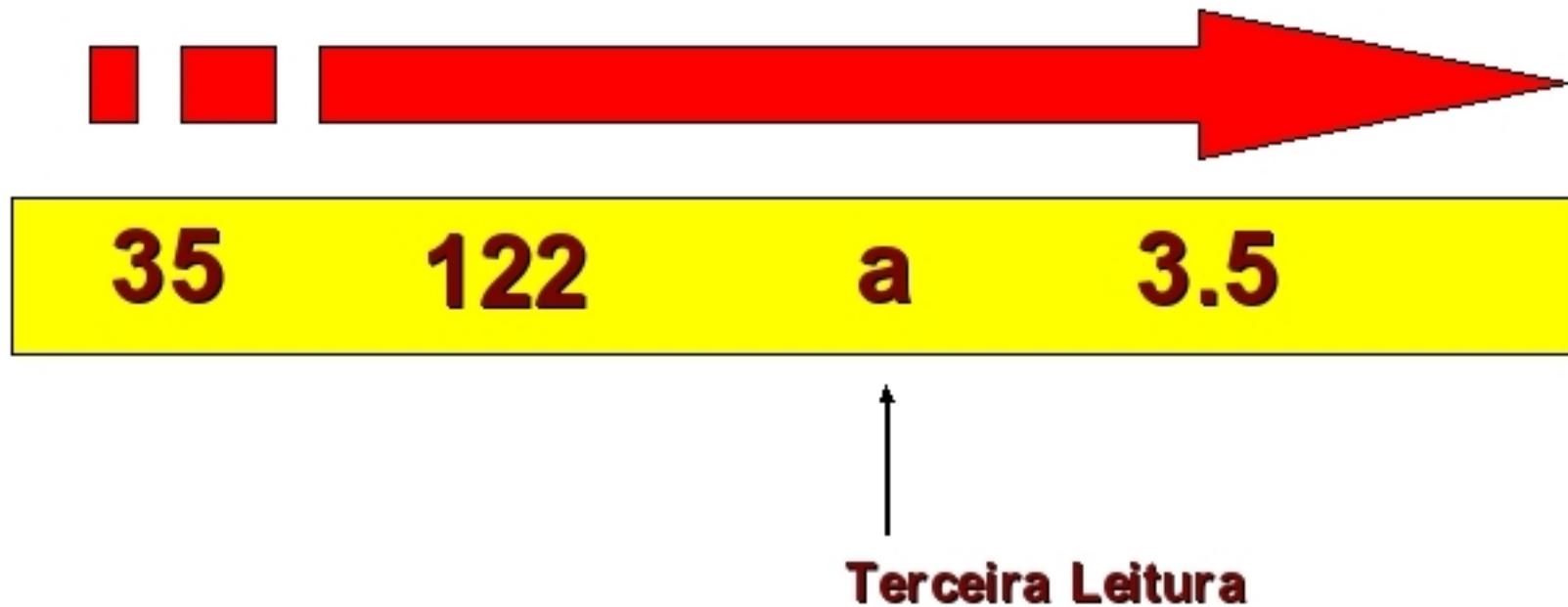


**Primeira Leitura**

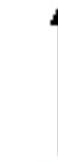
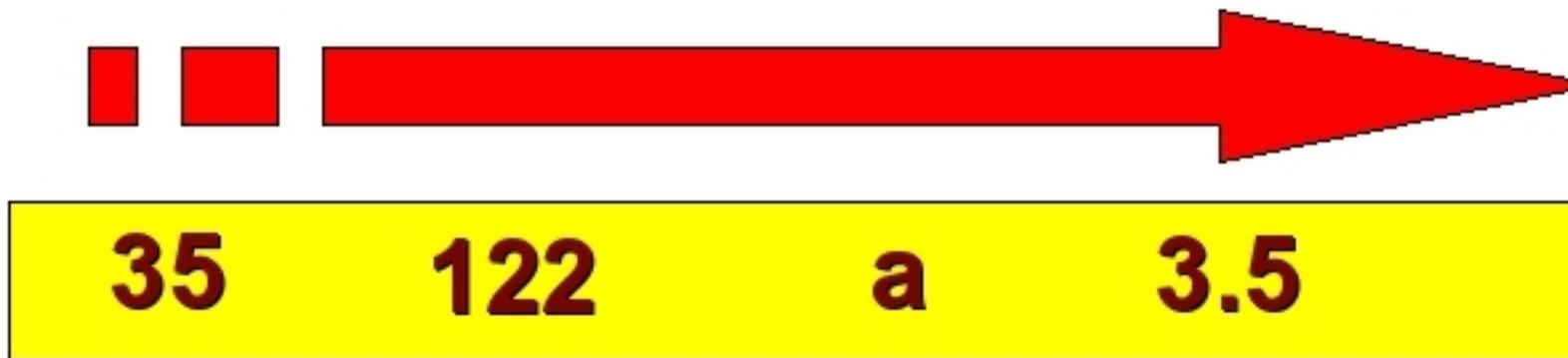
# Fluxos de dados cont.



# Fluxos de dados cont.



# Fluxos de dados cont.



Quarta Leitura

cederj

# **Comandos de Entrada e Saída**

---

- Em pseudo-código iremos empregar dois comandos básicos:
  - ❖ Comando de leitura de dados
  - ❖ Comando de saída de dados

# **Forma do comando de leitura**

---

- A forma geral do comando de leitura é a seguinte:

**leia <lista de variáveis>**

- **<lista de variáveis>** é um conjunto de nomes de variáveis separados por vírgulas.
- A lista de variáveis pode conter variáveis de todos os tipos válidos: inteiros, reais, caracteres

# **Comando de leitura**

---

**leia comprimento, largura, altura**

- A ação do comando de leitura acima é obter do fluxo de dados de entrada (teclado?) três valores numéricos e atribuí-los, na ordem em que foram obtidos, às três variáveis **comprimento, largura e altura**
- Como os dados são lidos de um fluxo, as próximas leituras obterão seus dados do ponto a seguir às últimas leituras.

# **Exemplo de comando de leitura**

---

## **Inicio**

**leia altura, largura**

**leia quantidade**

- Caso estivéssemos lendo os dados de um teclado, o usuário teria de fornecer três valores
- Por exemplo: 5.5, 3.1 e 10
- Estes comandos de leitura são equivalentes à:

**altura ← 5.5**

**largura ← 3.1**

**quantidade ← 10**

# **Forma do comando de escrita**

---

- A forma geral do comando de escrita é a seguinte:

**imprima <lista de saída>**

- **<lista de saída>** é uma lista de valores separados por vírgulas.
- Os valores podem ser variáveis e/ou constantes de todos os tipos válidos

# Comando de escrita

---

**imprima 'A área vale', area**

- O comando de escrita acima imprime primeiro a constante cadeia de caracteres

**'A área vale'**

- Em seguida imprime o valor da variável **area** que está na armazenado na memória
- Assumindo que **area** valha 4.5 teríamos na saída o seguinte resultado

**A área vale 4.5**

**cederj**

# Uso de espaços em branco

---

- É importante facilitar a leitura dos resultados para os usuários.
- Procurar então colocar informações adicionais que expliquem o significado dos resultados.
- Usar espaços em branco para separar resultados.
- Por exemplo:  
`imprima 'Lado =' , lado, ' comprimento =' , comp`
- Na tela sairia o seguinte resultado:  
`Lado = 3.5 comprimento = 10.5`

# **Usando espaços em branco**

---

**Inicio**

```
Index ← 17
imprima Index, Index
imprima Index, ' ', Index
Conta ← 27.5678
imprima Conta, Conta
imprima Conta, ' ', Conta
letra ← 'Z'
imprima letra, letra
imprima letra, ' ', letra
fim
```

# Usando espaços em branco

- No próximo slide os resultados dos comandos em vermelho estão em vermelho.
- Observe como os espaços em branco facilitou a leitura dos resultados.

# Espaços em branco (a saída)

**1717**

**17 17**

**27 . 567827 . 5678**

**27 . 5678 27 . 5678**

**zz**

**z z**

**cederj**

# Projeto e Desenvolvimento de Algoritmos

Desvios Condicionais

Adriano Cruz e Jonas Knopman

cederj

- **Objetivos**
- **Comando de desvio**
- **Comandos de desvio aninhados**

# Objetivos

---

- Apresentar o comando de desvio
- Apresentar o funcionamento dos comandos de desvio aninhados

# Comando de desvio

- Permite que o algoritmo decida autonomamente entre dois caminhos possíveis, qual irá executar.

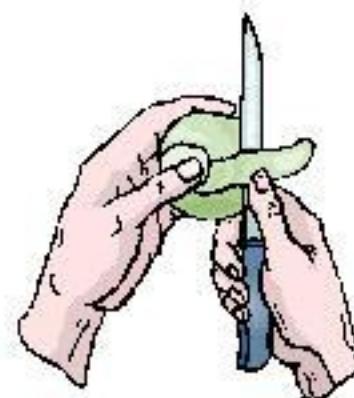
**Traga a cesta com as batatas**

**Se a roupa é clara então**

**coloque avental**

**Fim se**

**Descasque as batatas**



# Bloco de comandos

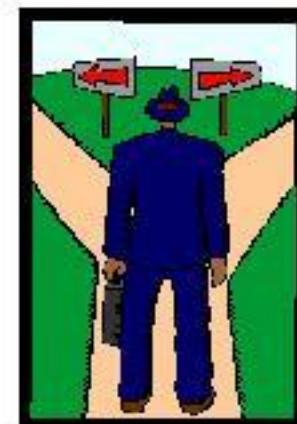
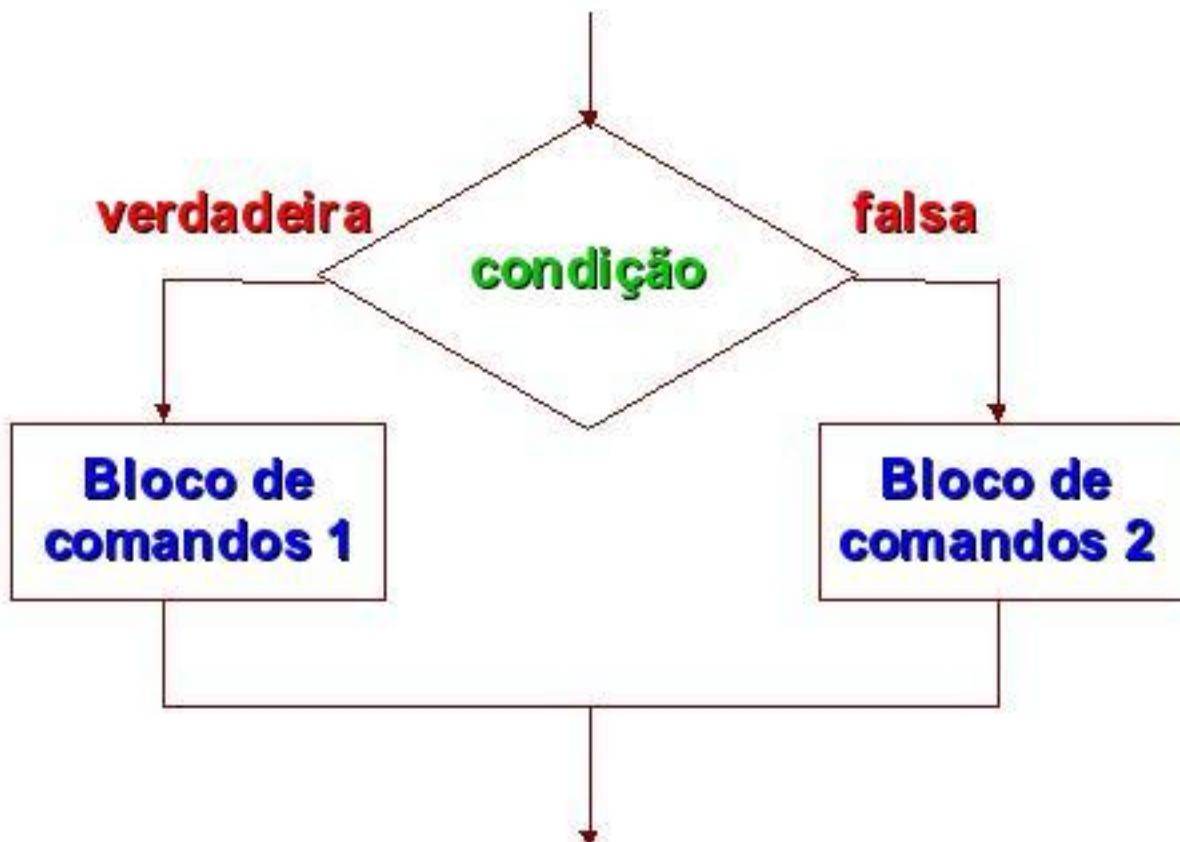
- Os comandos na nossa linguagem são escritos um por linha
- Não há nenhum sinal de pontuação ao final da linha
- Um bloco de comandos é uma série de comandos
- Em um bloco ou todos os comandos são executados ou nenhum é.

# Bloco de comandos exemplo

```
leia n1, n2
media ← (n1+n2)/2
imprima 'A média vale ', media
```

# Fluxograma do desvio

- Somente um dos blocos de comando é executado.



# Forma geral comando se

**se <expressão booleana> então**

**bloco de comandos 1**

**senão**

**bloco de comandos 2**

**fim se**



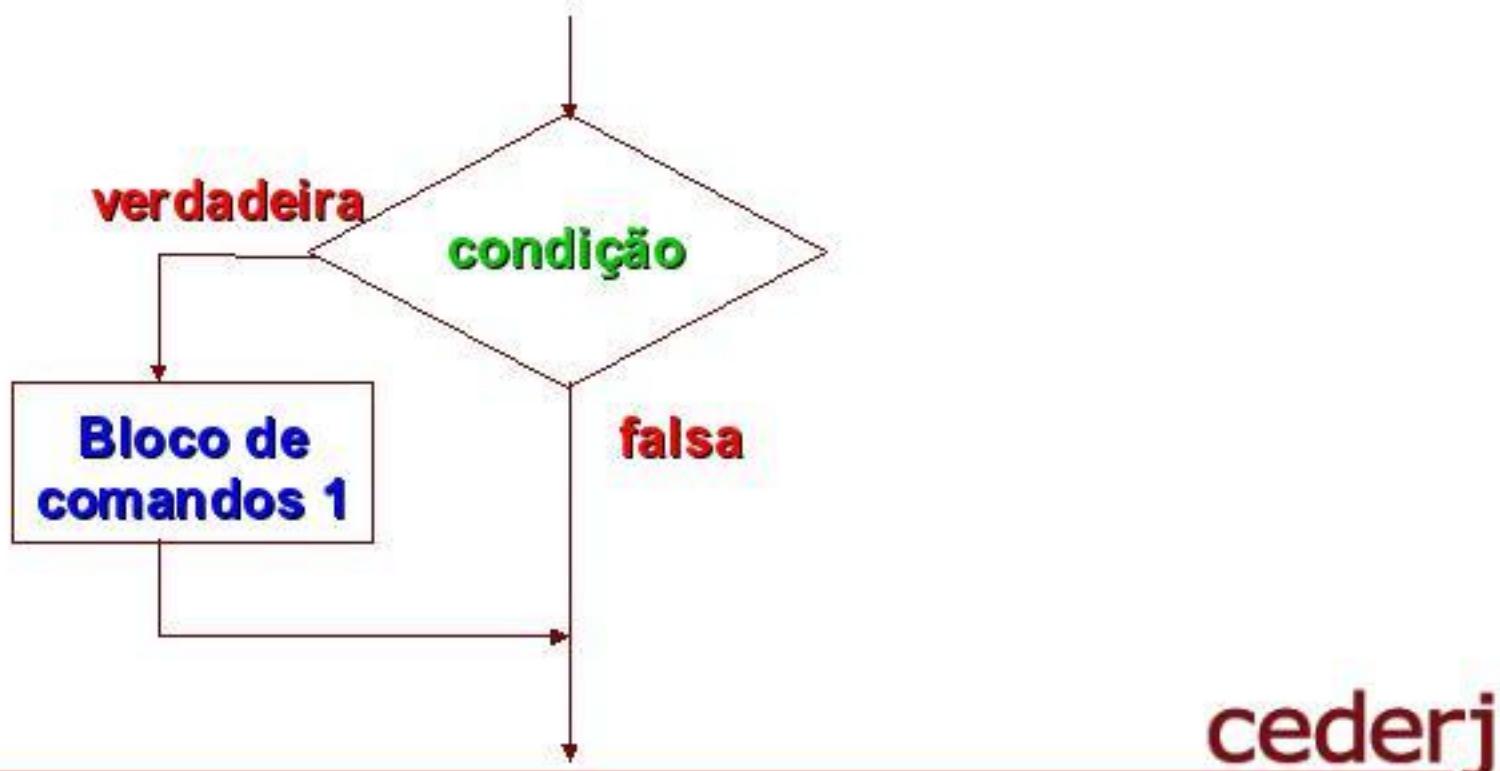
# Comando se simplificado

- Caso o bloco de comandos depois do senão seja vazio, esta parte pode ser omitida
- A forma geral simplificada é:

```
se <expressão booleana> então
    bloco de comandos
fim se
```

# Fluxograma se simplificado

- Neste caso o bloco de comandos é executado somente se a condição for verdadeira
- Caso contrário o algoritmo prossegue normalmente



# Exemplo comando se

início

acorda

se estiver fazendo sol então

vai à praia

senão

lê jornal

dorme

acorda

fim se

almoça

fim

# Seqüência de ações

- Se estiver fazendo sol  
**acorda**  
**vai à praia**  
**almoça**



- Senão  
**acorda**  
**lê jornal**  
**dorme**  
**acorda**  
**almoça**



**cederj**

# Exemplo algoritmo

- Ler dois números e imprimir o maior

**início**

```
imprima 'Primeiro número?'
leia num1
imprima 'Segundo número?'
leia num2
se num1 > num2 então
    imprima 'O maior é ', num1
senão
    imprima 'O maior é ', num2
fim se
fim
```

## **Exemplo e casos não previstos**

---

- No exemplo anterior um caso não foi previsto, qual foi?

## Exemplo e casos não previstos

- No exemplo anterior um caso não foi previsto, qual foi?
- Não foi verificada a possibilidade dos dois números serem iguais.
- Como resolver este problema?
- Aguarde...

# Comandos se aninhados

- As duas formas do comando **se** podem aparecer dentro de outros comando **se**.
- Diz-se que o comando **se** interno está aninhado no comando **se** externo.

```
se estiver sol então
  se eu tiver dinheiro  então
    Vou à praia
  fim se
fim se
```

## Comandos se aninhados

- Como o computador liga um fim se a um se?
- A indentação não é significativa para os compiladores!
- O fim se estará relacionado com o se anterior mais próximo.

# Melhorando o Exemplo

- Ler dois números e imprimir o maior

**início**

```
imprima 'Primeiro número?'
leia num1
imprima 'Segundo número?'
leia num2
se num1 > num2 então
    imprima 'O maior é ', num1
senão
    se num1 < num2 então
        imprima 'O maior é ', num2
    senão
        imprima 'Números iguais '
    fim se
fim se
fim
```

# Usando comandos se aninhados

- Observe as seguintes construções:

```
se estiver sol então
    se eu tiver dinheiro então
        vou à praia
    fim se
fim se
```

- ou

```
se estiver sol e
    eu tiver dinheiro
então
    vou à praia
fim se
```

- As duas construções são equivalentes. Então, quando usar comandos aninhados?

# Quando usar comandos aninhados?

- Quando tivermos de executar blocos de comandos diferentes para a cláusula **senão** das duas condições.

```
se estiver sol então
  se eu tiver dinheiro então
    Vou à Fazenda Felicidade
  senão
    Vou à praia
  fim se
senão
  Vou dormir
fim se
```

# Como construir sem aninhamento?

- Construir o exemplo anterior sem aninhamento.

# Como construir sem aninhamento?

- Construir o exemplo anterior sem aninhamento.

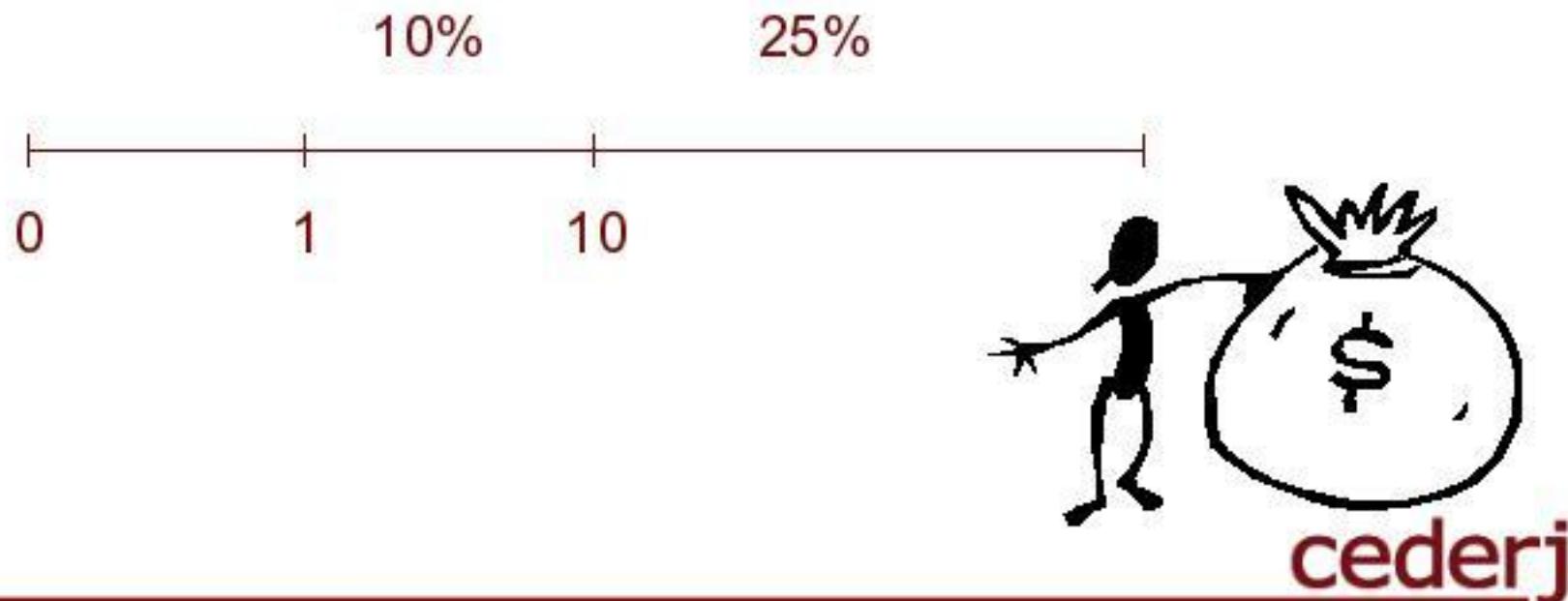
```
se estiver sol e tiver dinheiro então
    Vou à Fazenda Felicidade
fim se
```

```
se estiver sol e não tiver dinheiro então
    Vou à praia
fim se
```

```
se não estiver sol então
    Vou dormir
fim se
```

## Um exemplo

- Exemplo: Uma empresa vai dar um abono aos empregados que tenham mais de 1 ano de casa: 10% para os que tenham menos de 10 anos e 25% para os demais. Calcule o abono de um dado funcionário, dado o seu salário e o tempo de casa



# Exemplo: solução

```
inicio
    imprima 'Entre com o salario: '
    leia salario
    imprima 'Entre com o tempo de servico: '
    leia tempo
    se tempo > 1 então
        se tempo < 10 então
            salario ← 1.1*salario
        senão
            salario ← 1.25*salario
        fim se
        imprima 'Salário com abono: ', salario
    senão
        imprima 'Salário inalterado '
    fim se
```

# Outro exemplo

- Sistema de controle de temperatura

Tamb	Resfria	Aquece
$T_{amb} < T_{min}$	Falso	Verdadeiro
$T_{min} \leq T_{amb} < T_{max}$	Falso	Falso
$T_{amb} \geq T_{max}$	Verdadeiro	Falso



cederj

# Solução para temperatura

**constantes**

**Tmin = 15**

**Tmax = 25**

**inicio**

**imprima 'Temperatura ambiente: '**

**leia Tamb**

**se Tamb < Tmin então**

**aquece ← verdadeiro**

**resfria ← falso**

**senão**

**{ continua }**

# Solução para temperatura cont.

```
senão
    se Tamb < Tmax então
        aquece ← falso
        resfria ← falso
    senão
        aquece ← falso
        resfria ← verdadeiro
    fim se
fim se
imprima 'aquece : ', aquece
imprima 'resfria: ', resfria
fim
```

# Laços com número determinado de repetições

---

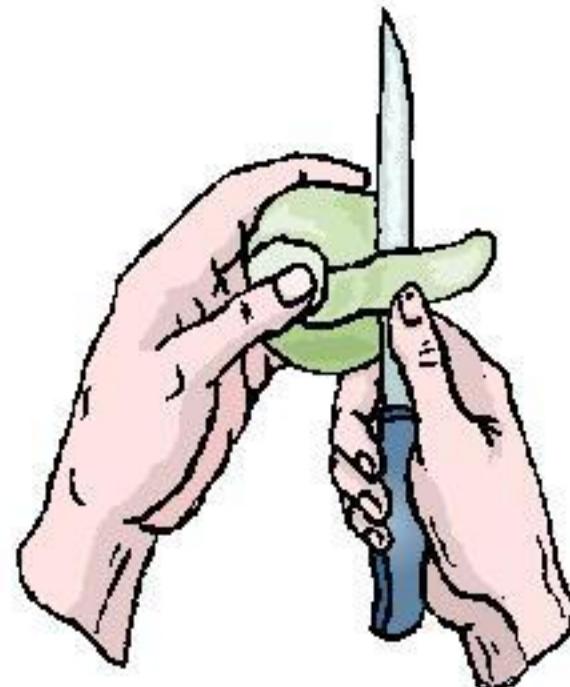
Traga a cesta com as batatas

Se a roupa é clara então

coloque avental

Fim se

Descasque uma batata



# Laços com número determinado de repetições

---

**Traga a cesta com as batatas**

**Se a roupa é clara então**

**coloque avental**

**Fim se**

**Repita 5 vezes**

**Descasque uma batata**

**Fim repita**

# Laços com número determinado de repetições

---

Em muitas situações práticas, existe a necessidade de saber o número da repetição. Por este motivo, PETEQS introduz um contador de voltas:

```
Traga a cesta com as batatas
Se a roupa é clara então
    coloque avental
Fim se
Para i ← 1 até 5 faça
    Descasque uma batata
    proximo i
```

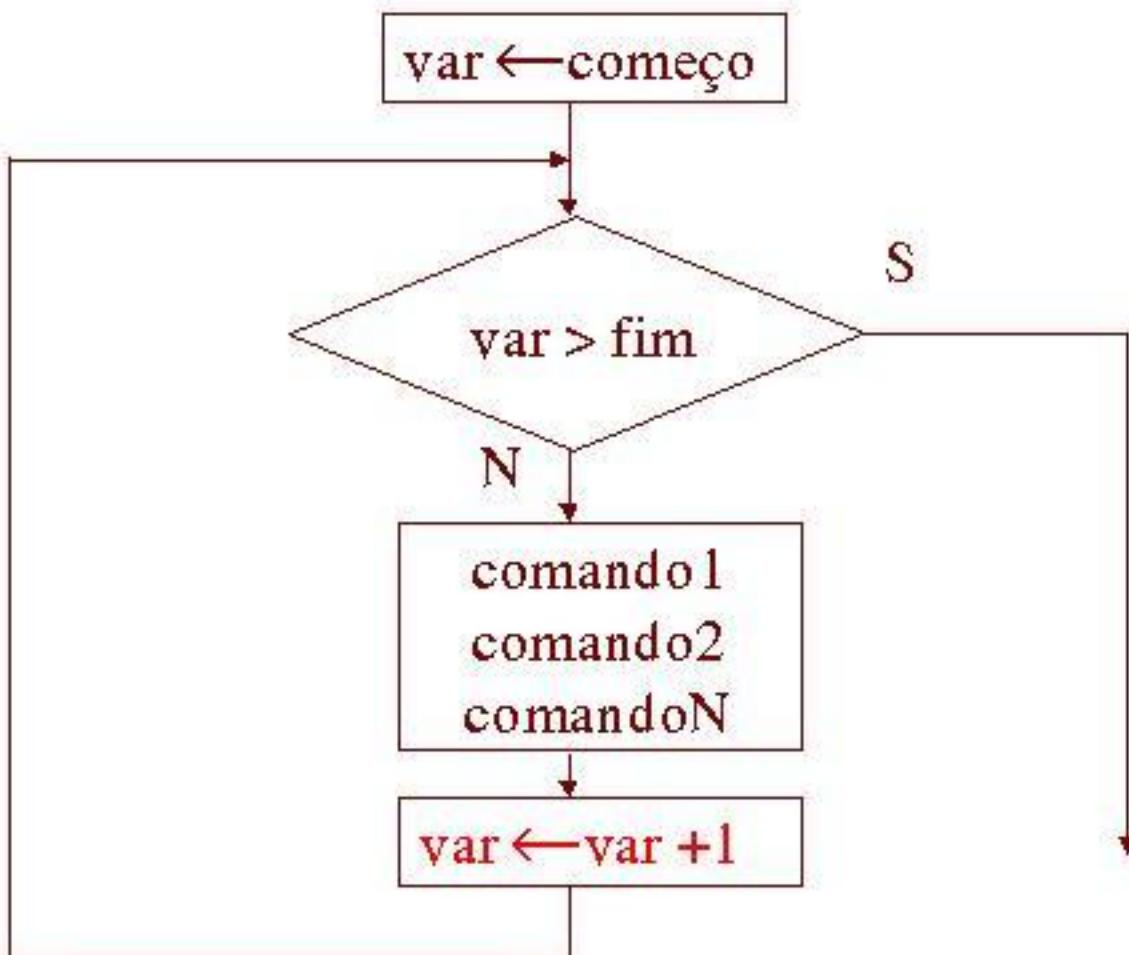
# Laços com número determinado de repetições

---

## Sintaxe:

```
para var ← começo até fim faça
    comando 1
    comando 2
    ...
    comando N
proximo var
```

# Visualização na forma de um fluxograma



# Laços com número determinado de repetições

---

`var` é uma variável qualquer, do tipo inteiro

`começo` é, em geral, igual a 1, mas pode assumir qualquer valor inteiro.

se `começo` for maior do que `fim`, o laço não é executado nem uma vez.

# Exemplo

---

**Escreva um programa para imprimir os números inteiros entre 2 e 5**

```
inicio
    i ← 2
    f ← 5
    para conta ← i até f faça
        imprima 'passo',conta
        proximo conta
    fim
```

# Exemplo

---

**Saída:**

**passo 2**

**passo 3**

**passo 4**

**passo 5**

## Outro exemplo

---

**Escreva um programa para imprimir os números inteiros entre 10 e 40, múltiplos de 10**

```
inicio
    total ← 0
    para i ← 1 até 4 faça
        total ← total + 10
        imprima 'total=' , total
    proximo i
fim
```

## Outro exemplo

---

**Saída:**

**total=10**

**total=20**

**total=30**

**total=40**

# Laços com número determinado de repetições

---

É possível ainda utilizar variáveis caracter para controlar o número de iterações do laço.

```
inicio
    para ch ← 'A' até 'D' faça
        imprima 'ch =', ch
        proximo ch
    fim
```

# Laços com número determinado de repetições

---

**Saída:**

**ch = A**

**ch = B**

**ch = C**

**ch = D**

## Laços dentro de laços

---

É bastante comum a situação em que um comando, ou grupo de comandos, tem de ser repetido dentro de um laço externo.

Imagine as horas do dia. Elas assumem todos os dias os mesmos valores: 12:00, 13:00, 14:00, etc.

No entanto, cada hora é unicamente identificada:

12:00 (laço interno)

do dia 19/06/2001 (laço externo)

12:00 (laço interno)

do dia 20/06/2001 (laço externo)



# Laços dentro de laços

---

## Um exemplo

```
inicio
    para cExt ← 1 até 3 faça
        imprima 'Laco externo: ',cExt
        para cInt ← 1 até 3 faça
            imprima 'Laco interno: ', cExt, '.', cInt
            proximo cInt
        proximo cExt
fim
```

# Laços dentro de laços

---

**Saída:**

Laco externo: 1

Laco interno: 1.1

Laco interno: 1.2

Laco interno: 1.3

Laco externo: 2

Laco interno: 2.1

Laco interno: 2.2

Laco interno: 2.3

Laco externo: 3

Laco interno: 3.1

Laco interno: 3.2

Laco interno: 3.3

## Outro Exemplo Conversão de Temperaturas

---

**Outro exemplo:** Escreva um programa para converter temperaturas em graus Celsius para Fahrenheit.

As temperaturas a serem convertidas estão na faixa  $[0^{\circ}\text{C}..100^{\circ}\text{C}]$  e devem variar de  $10^{\circ}\text{C}$  em  $10^{\circ}\text{C}$ . Assinale as temperaturas de congelamento e fervura da água.



# Outro Exemplo Conversão de Temperaturas

inicio

Celsius ← 0.0

para i ← 1 até 11 faça

Fahrenheit ← 1.8\*Celsius+32.0

Imprima 'C =', Celsius, 'F =', Fahrenheit

se Celsius = 0.0 então

    Imprima ' Congelamento'

    fim se

    se Celsius = 100.0 então

        Imprima ' Fervura'

        fim se

    Celsius ← Celsius+10.0

proximo i

fim

## Outro Exemplo Conversão de Temperaturas

---

**Saída:**

C = 0.0	F = 32.0	Congelamento
C = 10.0	F = 50.0	
C = 20.0	F = 68.0	
C = 30.0	F = 86.0	
C = 40.0	F = 104.0	
C = 50.0	F = 122.0	
C = 60.0	F = 140.0	
C = 70.0	F = 158.0	
C = 80.0	F = 176.0	
C = 90.0	F = 194.0	
C = 100.0	F = 212.0	Fervura

## Outro Exemplo Seu Aniversário

---

Escreva um programa que escreva os números inteiros de 1 a 12 e escreva uma mensagem ao lado do número correspondente ao mês do seu aniversário



# Outro Exemplo Seu Aniversário

---

```
inicio
    para mes ← 1 até 12 faça
        imprima 'mes : ', mes
        se mes=3 entao
            imprima ' Feliz Aniversario, Jonas!'
        fim se
    proximo mes
fim
```

# Outro Exemplo Seu Aniversário

---

## Saída

```
mes : 1
mes : 2
mes : 3  Feliz Aniversario, Jonas!
mes : 4
mes : 5
mes : 6
mes : 7
mes : 8
mes : 9
mes : 10
mes : 11
mes : 12
```

## Outro Exemplo

### Imprime números inteiros

Escreva um programa que liste os números inteiros de 1 a 12, exceto os números 2 e 9.

```
inicio
    para i ← 1 até 12 faça
        se (i<>2) E (i<>9) então
            imprima i
        fim se
    proximo i
fim
```

# Outro Exemplo

## Imprime números inteiros

---

### Saída

1  
3  
4  
5  
6  
7  
8  
10  
11  
12

# O laço enquanto

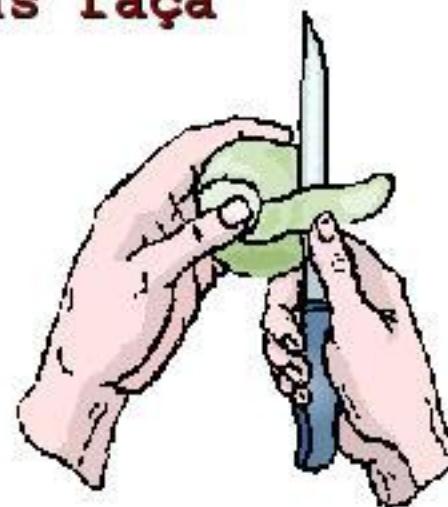
---

Traga a cesta com as batatas  
se roupa clara então  
coloque avental

fim se

enquanto não tiver 1Kg de batatas faça  
descasque uma batata

fim enquanto



# Laços com Número Indeterminado de Repetições

---

**cederj**

# Objetivos

- O conceito
- Sintaxe
- O laço enquanto é tudo o que você precisa!
- Exemplos

# O laço enquanto

Traga a cesta com as batatas

**se roupa clara então**

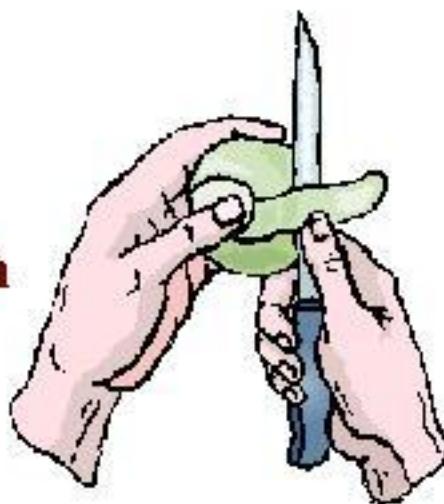
  coloque avental

**fim se**

**enquanto não tiver 1Kg de batatas faça**

  descasque uma batata

**fim enquanto**



## O laço enquanto

- O número de repetições não é, em geral, conhecido a priori
- O laço deve ser finito

**enquanto** não tiver 1Kg de batatas **faça**  
descasque uma laranja

**fim enquanto**

# O laço enquanto

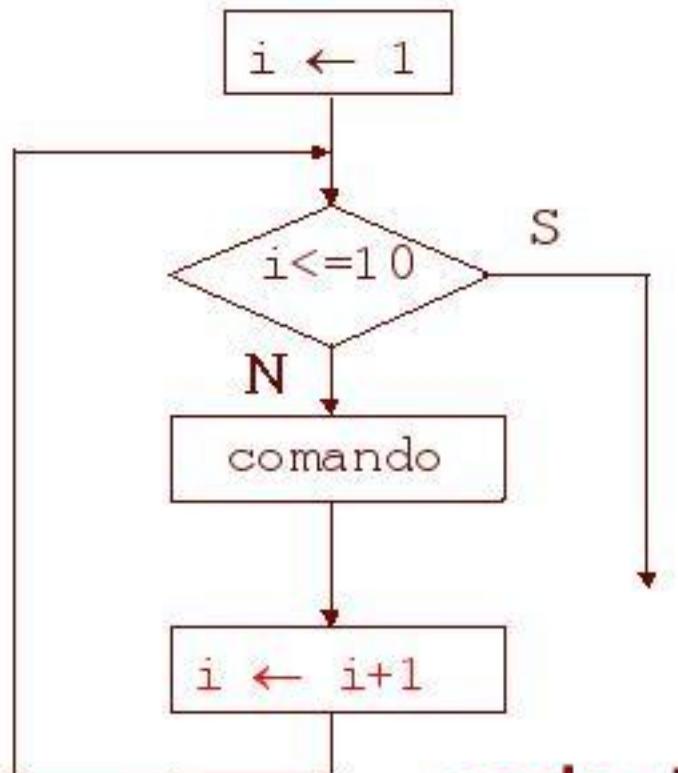
MÓDULO 8  
AULA 1

- O laço enquanto é tudo o que você precisa:

**para**  $i \leftarrow 1$  até 10 **faça**  
    comando  
**proximo**  $i$

- ou

$i \leftarrow 1$   
**enquanto**  $i \leq 10$  **faça**  
    comando  
 $i \leftarrow i + 1$   
**fim enquanto**



**cederj**

# O laço enquanto. Um exemplo

inicio

imprima 'Quer continuar?'

leia ch

enquanto ch<>'n' E ch<>'N' faça

imprima 'Vamos continuar...'

imprima 'Quer continuar?'

leia ch

fim enquanto

fim

cederj

# O laço enquanto. Um exemplo

inicio

**acabou** ← falso

**enquanto** nao acabou **faça**

**imprima** 'Quer continuar?'

**leia** ch

**se** (ch='n') **OU** (ch='N') **então**

**acabou** ← verdadeiro

**senão**

**imprima** 'Vamos continuar...'

**fim se**

**fim enquanto**

**fim**

**cederj**

# O laço enquanto. Um exemplo

MÓDULO 8  
AULA 1

**inicio**

**acabou**  $\leftarrow$  **falso**

**enquanto** **nao** **acabou** **faça**

**imprima** 'Quer continuar?'

**leia** **ch**

**se** (**ch** = 'n') **então**

**acabou**  $\leftarrow$  **verdadeiro**

**senão**

**se** (**ch** = 'N') **então**

**acabou**  $\leftarrow$  **verdadeiro**

**senão**

**imprima** 'Vamos continuar...'

**fim se**

**fim se**

**fim enquanto**

**fim**

**cederj**

# O laço enquanto. Um exemplo

➤ Saída

Quer continuar? s

Vamos continuar...

Quer continuar? S

Vamos continuar...

Quer continuar? s

Vamos continuar...

Quer continuar? n

cederj

# O laço enquanto. Outro exemplo

MÓDULO 8  
AULA 1

## ➤ Um laço finito

**inicio**

**i**  $\leftarrow$  4

**enquanto** **i** < 10 **faça**

**imprima** 'Dentro do laco enquanto.'

**imprima** 'i = ',**i**

**i**  $\leftarrow$  **i** + 2

**fim enquanto**

**fim**

**cederj**

## O laço enquanto. Outro exemplo

### ➤ Saída

Dentro do laco enquanto. i = 4

Dentro do laco enquanto. i = 6

Dentro do laco enquanto. i = 8

## Exemplos de usos de laços

- Achar o maior número de uma série de números positivos fornecidos pelo usuário.



cederj

# Exemplos de usos de laços

**inicio**

maior ← -1

...

**fim**



**cederj**

# Exemplos de usos de laços

MÓDULO 8  
AULA 1

**inicio**

maior ← -1

acabou ← **falso**

**enquanto não** acabou **faça**

...

**fim enquanto**

...

**fim**



**cederj**

# Exemplos de usos de laços

inicio

maior ← -1

acabou ← **falso**

**enquanto não** acabou **faça**

**leia** valor

  ...

**fim enquanto**

  ...

**fim**



**cederj**

# Exemplos de usos de laços

MÓDULO 8  
AULA 1

**inicio**

maior ← -1

acabou ← **falso**

**enquanto não** acabou **faça**

leia valor

**se** valor < 0 **então**

acabou ← **verdadeiro**

...

**fim enquanto**

...

**fim**



**cederj**

# Exemplos de usos de laços

MÓDULO 8  
AULA 1

inicio

maior ← -1

acabou ← falso

**enquanto não acabou faça**

**leia** valor

**se** valor < 0 **então**

**acabou** ← verdadeiro

**senão**

    ...

**fim enquanto**

  ...

**fim**



**cederj**

# Exemplos de usos de laços

MÓDULO 8  
AULA 1

inicio

    maior ← -1

    acabou ← falso

**Enquanto não acabou faça**

        leia valor

        se valor < 0 então

            acabou ← verdadeiro

        senão

            se valor > maior então

                ...

**fim enquanto**

    ...

**fim**



**cederj**

# Exemplos de usos de laços

MÓDULO 8  
AULA 1

inicio

    maior ← -1

    acabou ← falso

**Enquanto não acabou faça**

        leia valor

**se** valor < 0 **então**

            acabou ← verdadeiro

**senão**

**se** valor > maior **então**

                maior ← valor

**fim se**

**fim se**

**fim enquanto**

...

**fim**



**cederj**

# Exemplos de usos de laços

inicio

maior ← -1

acabou ← falso

**Enquanto não acabou faça**

    leia valor

    se valor < 0 então

        acabou ← verdadeiro

    senão

        se valor > maior então

            maior ← valor

    fim se

    fim se

    fim enquanto

    imprima 'maior = ', maior

fim

MÓDULO 8  
AULA 1



cederj

# Problema com laços

MÓDULO 8  
AULA 1

## ➤ Saída:

```
Entre com um numero: 2
Entre com um numero: 4
Entre com um numero: 8
Entre com um numero: 5
Entre com um numero: 3
Entre com um numero: 1
Entre com um numero: -1
maior =    8
```

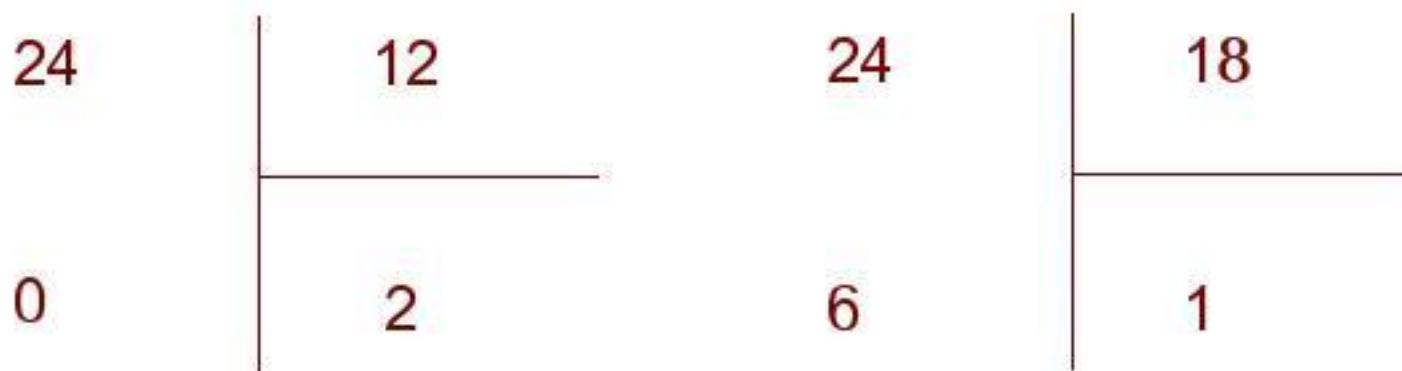
cederj

# Problema com laços

MÓDULO 8  
AULA 1

## ➤ Problema MMC

Exemplo: calcular o MMC de 12 e 18



24 não é um múltiplo comum!!!

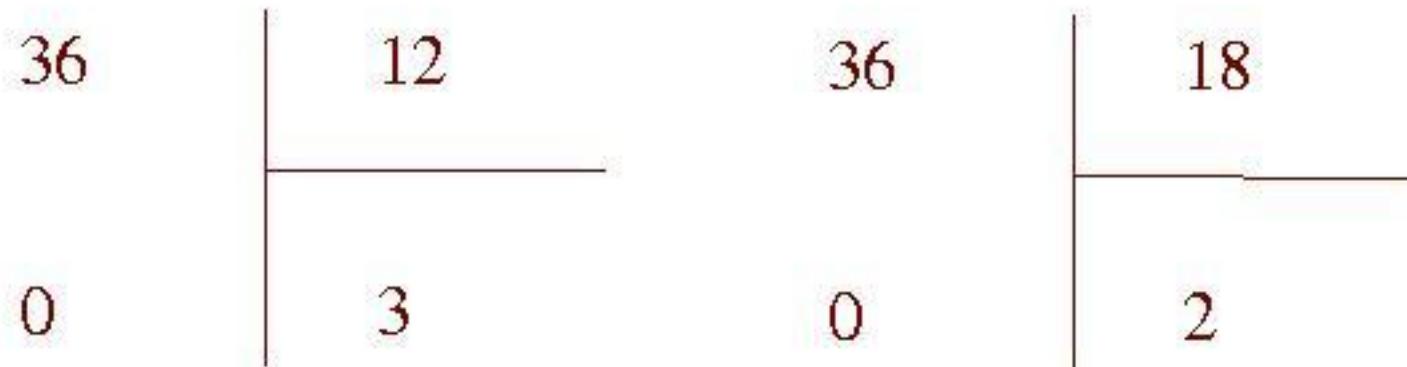
**cederj**

# Problema com laços

MÓDULO 8  
AULA 1

## ➤ Problema MMC

Exemplo: calcular o MMC de 12 e 18



36 é um múltiplo comum!!!

**cederj**

# Problemas com laços

MÓDULO 8  
AULA 1

**inicio**

**leia** num1 , num2

...

**fim**

cederj

# Problemas com laços

MÓDULO 8  
AULA 1

**inicio**

**leia** num1, num2

mmc  $\leftarrow$  1

...

**fim**

**cederj**

# Problemas com laços

**inicio**

**leia num1, num2**

**mmc**  $\leftarrow$  1

**acabou**  $\leftarrow$  **falso**

**enquanto não acabou faça**

...

**fim enquanto**

...

**fim**

# Problemas com laços

MÓDULO 8  
AULA 1

**inicio**

**leia num1, num2**

**mmc**  $\leftarrow$  1

**acabou**  $\leftarrow$  **falso**

**enquanto não acabou faça**

**se** (**mmc mod num1 = 0**) E

**(mmc mod num2 = 0)** **então**

...

**fim enquanto**

...

**fim**

**cederj**

# Problemas com laços

MÓDULO 8  
AULA 1

**inicio**

**leia num1, num2**

**mmc ← 1**

**acabou ← falso**

**enquanto não acabou faça**

**se (mmc mod num1 = 0) E**

**(mmc mod num2 = 0) então**

**acabou ← verdadeiro**

**...**

**fim enquanto**

**...**

**fim**

**cederj**

# Problemas com laços

MÓDULO 8  
AULA 1

inicio

leia num1, num2

mmc ← 1

acabou ← falso

**enquanto** não acabou **faça**

**se** (mmc mod num1 = 0) E

        (mmc mod num2 = 0) **então**

            acabou ← verdadeiro

**senão**

        mmc ← mmc + 1

**fim se**

**fim enquanto**

...

**fim**

cederj

# Problemas com laços

MÓDULO 8  
AULA 1

**inicio**

**leia num1, num2**

**mmc ← 1**

**acabou ← falso**

**enquanto não acabou faça**

**se (mmc mod num1 = 0) E**

**(mmc mod num2 = 0) então**

**acabou ← verdadeiro**

**senão**

**mmc ← mmc + 1**

**fim enquanto**

**imprima mmc**

**fim**

**cederj**

# Problemas com laços

MÓDULO 8  
AULA 1

**inicio**

**leia** num1, num2

mmc  $\leftarrow$  1

acabou  $\leftarrow$  **falso**

**enquanto** não acabou **faça**

**se** (mmc mod num1 = 0) E

        (mmc mod num2 = 0) **então**

            acabou  $\leftarrow$  **verdadeiro**

**senão**

        mmc  $\leftarrow$  mmc + 1

**fim se**

**fim enquanto**

**imprima** mmc

**fim**

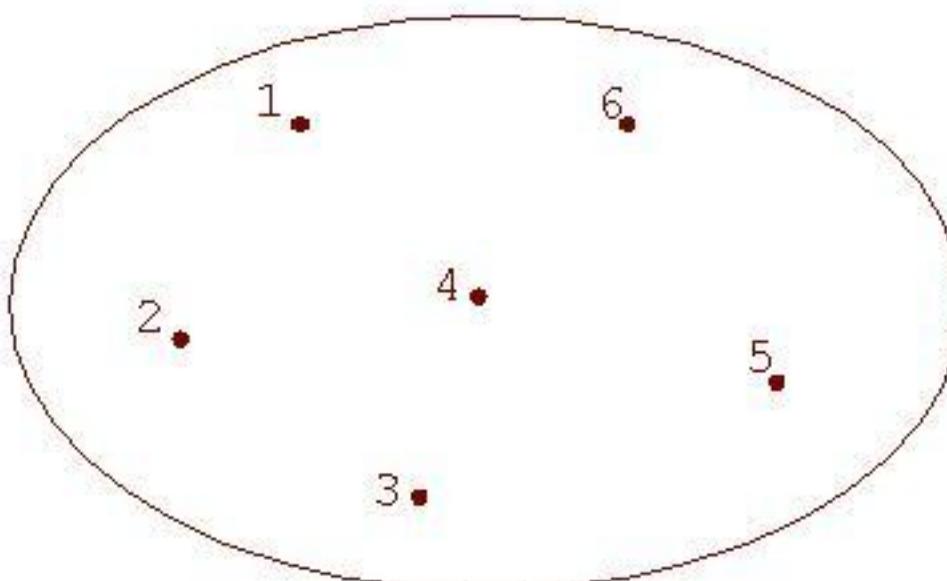
**cederj**

# Variáveis indexadas

# Objetivos

- Definição
- Necessidade de uso
- Operações básicas
- Exemplos

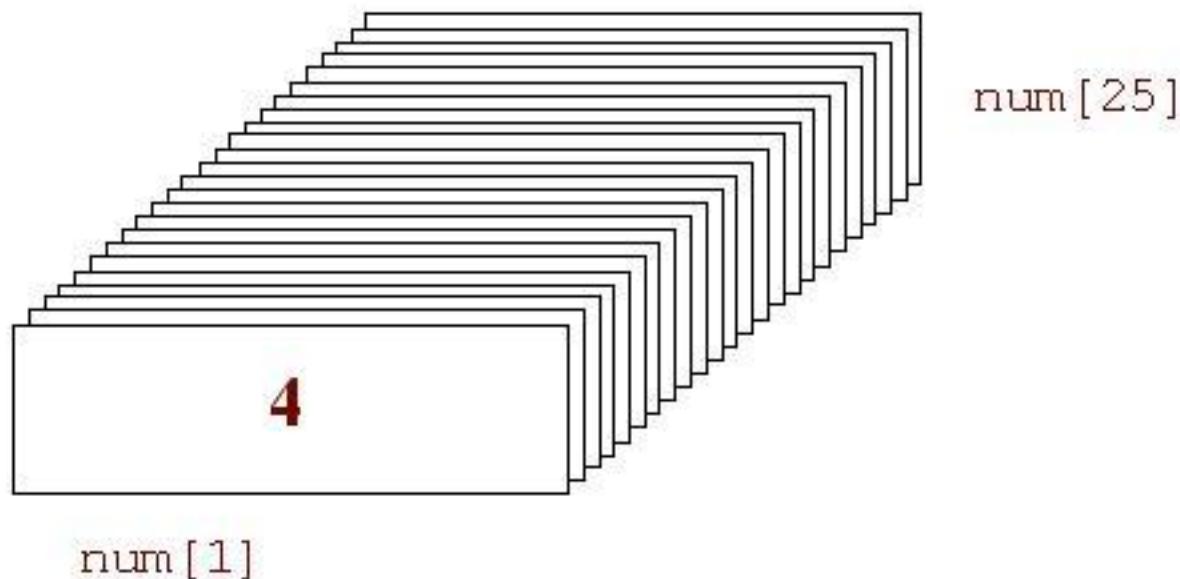
- Definição: Um vetor é uma coleção de elementos de um mesmo tipo. Cada um dos elementos é unicamente identificado por um número inteiro.



# Representação gráfica de um vetor

MÓDULO 9  
AULA 1

`num[1] ← 4`



**cederj**

- O valor do índice não deve ser confundido com o conteúdo da posição do vetor.
- O índice identifica o elemento dentro do conjunto. O índice tem de ser obrigatoriamente inteiro.
- O elemento do vetor pode ser um número inteiro, um número real, uma variável booleana, um caracter, uma string, ...

- O índice de um vetor corresponde à numeração das casas numa rua.



- O número de uma casa nada tem a ver com o seu conteúdo.

# Quando usar vetores?

MÓDULO 9  
AULA 1

- Quando desejamos processar uma grande quantidade de informações fica extremamente complicado, ou praticamente impossível, criar e manter um conjunto grande de variáveis.



cederj

# Quando usar vetores?

MÓDULO 9  
AULA 1

- Podemos imaginar a situação onde, em um programa para manutenção do cadastro de um banco, os dados de cada cliente fossem armazenados em uma variável diferente. O programa teria de lidar com milhares de variáveis. Seria difícil, por exemplo, percorrer a lista de clientes e procurar pelo cliente ‘Fernando Henrique Cardoso’



cederj

# Quando usar vetores?

MÓDULO 9  
AULA 1

- Dificuldade de manipulação de informação relacionada armazenada em variáveis simples

{Calcular a média das idades de 5 crianças}

**inicio**

**leia** idadeA

**leia** idadeB

**leia** idadeC

**leia** idadeD

**leia** idadeE

media  $\leftarrow$  (idadeA + idadeB + idadeC +  
idadeD + idadeE) / 5

**imprime** media

**fim**

**cederj**

# Quando usar vetores?

MÓDULO 9  
AULA 1

- E se fossem 1000 crianças?



cederj

# Quando usar vetores?

MÓDULO 9  
AULA 1

- Alternativa: informação armazenada em vetores

inicio

soma  $\leftarrow$  0

...

fim



cederj

# Quando usar vetores?

MÓDULO 9  
AULA 1

- Alternativa: informação armazenada em vetores

```
inicio
    soma ← 0
    para i ← 1 até 5 faça
        ...
        proximo i
    ...
fim
```



**cederj**

# Quando usar vetores?

MÓDULO 9  
AULA 1

- Alternativa: informação armazenada em vetores

```
inicio
    soma ← 0
    para i ← 1 até 5 faça
        leia idade[i]
        ...
    proximo i
    ...
fim
```



**cederj**

# Quando usar vetores?

MÓDULO 9  
AULA 1

- Alternativa: informação armazenada em vetores

```
inicio
    soma ← 0
    para i ← 1 até 5 faça
        leia idade[i]
        soma ← soma + idade[i]
    proximo i
    ...
fim
```



**cederj**

# Quando usar vetores?

MÓDULO 9  
AULA 1

- Alternativa: informação armazenada em vetores

**inicio**

**soma** ← 0

**para** i ← 1 **até** 5 **faça**

**leia** idade[i]

        soma ← soma + idade[i]

**proximo** i

    media ← soma/5

    ...

**fim**



**cederj**

# Quando usar vetores?

MÓDULO 9  
AULA 1

- Alternativa: informação armazenada em vetores

**inicio**

**soma** ← 0

**para** i ← 1 **até** 5 **faça**

**leia** idade[i]

        soma ← soma + idade[i]

**proximo** i

    media ← soma/5

**imprime** media

**fim**



**cederj**

# Quando usar vetores?

MÓDULO 9  
AULA 1

- E se fossem 1000 crianças?



cederj

# Quando usar vetores?

MÓDULO 9  
AULA 1

**inicio**

**soma**  $\leftarrow$  0

**para** i  $\leftarrow$  1 **até** 1000 **faça**

**leia** idade[i]

        soma  $\leftarrow$  soma + idade[i]

**proximo** i

    media  $\leftarrow$  soma/1000

**imprime** media

**fim**

**cederj**

- Observe a construção:

```
para i ← 1 até 1000 faça
    leia crianca[i]
    proximo i
```

- A grande força na utilização de um vetor consiste em associá-lo a um laço.
- Com isso podemos facilmente percorrer um vetor para consultas ou atualizações.

# Um exemplo: inicializando e percorrendo vetores

MÓDULO 9  
AULA 1

inicio

para i ← 1 até 5 faça

leia carros[i]

proximo i

...

fim

## Um exemplo: inicializando e percorrendo vetores

MÓDULO 9  
AULA 1

inicio

**para** *i*  $\leftarrow$  1 **até** 5 **faça**

**leia** carros[i]

**proximo i**

**para** i  $\leftarrow$  1 **até** 5 **faça**

```
imprima 'carro', i, ' quantidade:',  
    carros[i]
```

proximo i

fim

cederj

# Inicializando e percorrendo vetores

MÓDULO 9  
AULA 1

## ➤ Saída:

```
carro 1 quantidade: 10
carro 2 quantidade: 10
carro 3 quantidade: 10
carro 4 quantidade: 10
carro 5 quantidade: 10
```

- Leitura de uma tabela de 100 valores e impressão da tabela multiplicada por uma constante.

# Exemplo

MÓDULO 9  
AULA 1

**inicio**

{entrada de dados}

**para** i  $\leftarrow$  1 **até** 100 **faça**

**leia** tab[i]

**proximo** i

...

**fim**

**cederj**

# Exemplo

MÓDULO 9  
AULA 1

```
inicio
    {entrada de dados}
    para i ← 1 até 100 faça
        leia tab[i]
    proximo i
    {processamento}
    para i ← 1 até 100 faça
        tab[i] ← 3.1415*tab[i]
    proximo i
    ...
fim
```

cederj

# Exemplo

MÓDULO 9  
AULA 1

**inicio**

{entrada de dados}

**para** i  $\leftarrow$  1 **até** 100 **faça**

**leia** tab[i]

**proximo** i

{processamento}

**para** i  $\leftarrow$  1 **até** 100 **faça**

        tab[i]  $\leftarrow$  3.1415\*tab[i]

**proximo** i

{saída de dados}

**para** i  $\leftarrow$  1 **até** 100 **faça**

**imprima** tab[i]

**proximo** i

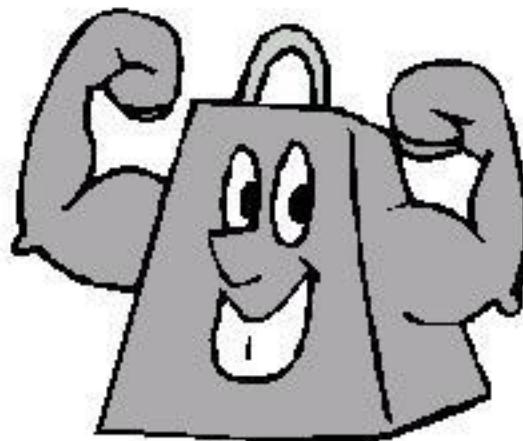
**fim**

**cederj**

## Outro exemplo

MÓDULO 9  
AULA 1

- ▶ Determinar o maior elemento de um vetor e a sua posição



# Maior elemento de um vetor

MÓDULO 9  
AULA 1

**inicio**

{entrada de dados}

**para** i  $\leftarrow$  1 **até** 20 **faça**

**leia** tabela[i]

**proximo** i

...

**fim**

**cederj**

# Maior elemento de um vetor

MÓDULO 9  
AULA 1

inicio

{entrada de dados}

para i ← 1 até 20 faça

    leia tabela[i]

proximo i

{assume que o primeiro elemento da}

{tabela é o maior}

maior ← tabela[1]

pos ← 1

...

fim



cederj

# Maior elemento de um vetor

MÓDULO 9  
AULA 1

inicio

```
...
{assume que o primeiro elemento da}
{tabela é o maior}

maior ← tabela[1]
pos ← 1
{procura o maior}
para i ← 2 até 20 faça
    se tabela[i] > maior então
        maior ← tabela[i]
        pos ← i
    fim se
proximo i
...
fim
```



cederj

# Maior elemento de um vetor

MÓDULO 9  
AULA 1

inicio

...

{assume que o primeiro elemento da}  
{tabela é o maior}

maior ← tabela[1]

pos ← 1

{procura o maior}

para i ← 2 até 20 faça

se tabela[i] > maior então

    maior ← tabela[i]

    pos ← i

fim se

proxímo i

imprima maior, pos

fim



cederj

## ➤ A declaração de constantes

constante

DIM = 100

# Exemplo

MÓDULO 9  
AULA 1

**constante**

**DIM** = 100

**inicio**

**para** i  $\leftarrow$  1 **até** DIM **faça**

**leia** tab[i]

**proximo** i

**para** i  $\leftarrow$  1 **até** DIM **faça**

tab[i]  $\leftarrow$  3.1415\*tab[i]

**proximo** i

**para** i  $\leftarrow$  1 **até** DIM **faça**

**imprima** tab[i]

**proximo** i

**fim**

**cederj**

- **Vantagem na utilização de constantes:**

Se houver necessidade de alterar a dimensão do vetor, basta alterar o valor da constante DIM.

# **Projeto e Desenvolvimento de Algoritmos**

**Procedimentos**

**Introdução**

**Adriano Cruz e Jonas Knopman**

---

**cederj**

# Objetivos

- Definição
- Necessidade e vantagens da utilização

# **Filé de peixe ao molho branco**



**{preparo dos peixes}**

Lave os filés e tempere com o suco dos limões, sal, pimenta e salsinha ...

**{preparo do molho branco}**

**Coloque numa panela a manteiga, a farinha e o leite e misture bem. Em fogo médio, cozinhе até engrossar.  
Adicione o sal, ...**

**{juntando os dois}**

Adicione queijo parmesão ralado e queijo gruyère. Misture e ponha sobre os filés.

**cederj**

# **Alface ao molho branco**



## **{preparo da alface}**

Derreta a manteiga. Junte a alface cortada. Salpique o sal e deixe cozinhar por uns 5 a 10 minutos ou até a alface ficar tenra, ou o líquido da panela secar.

## **{preparo do molho branco}**

**Coloque numa panela a manteiga, a farinha e o leite e misture bem. Em fogo médio, cozinhe até engrossar. Adicione o sal ...**

## **{juntando os dois}**

Junte ao molho branco e ao suco de limão. Coloque numa travessa e enfeite em volta com pão torrado cortado em triângulos.

**cederj**

## **Procedimentos**

- *A receita de molho branco é replicada junto a toda receita que faz uso de molho branco.*

## **Vantagem**

- *O acesso à receita de molho branco é muito rápido.*



---

**cederj**

## ***Desvantagem***

- *O livro fica maior*



## ***Outra desvantagem***

- *Se amanhã descubro que uma pitada de pimenta dá um sabor especial ao molho branco, tenho de alterar a receita em vários pontos do livro.*



## **Outra alternativa**

### ➤ Filé de peixe ao molho branco

{preparo dos peixes}

Lave os filés e tempere com o suco dos limões, sal, pimenta e salsinha ...

{preparo do molho branco}

**Prepare a receita básica de molho branco (pág.25)**

{juntando os dois}

Adicione queijo parmesão ralado e queijo gruyère. Misture e ponha sobre os filés.

**cederj**

# **Alface ao molho branco**



## **{preparo da alface}**

Derreta a manteiga. Junte a alface cortada. Salpique o sal e deixe cozinhar por uns 5 a 10 minutos ou até a alface ficar tenra, ou o líquido da panela secar.

## **{preparo do molho branco}**

**Prepare a receita básica de molho branco (pág.25)**

## **{juntando os dois}**

Junte ao molho branco e ao suco de limão. Coloque numa travessa e enfeite em volta com pão torrado cortado em triângulos.

**cederj**

## ***Na página 25...***

### **Molho branco**

**Coloque numa panela a manteiga, a farinha e o leite e misture bem. Em fogo médio, cozинhe até engrossar. Adicione o sal, a pimenta e o queijo. Continue com a panela no fogo, cozinhando até que o queijo derreta, mexendo constantemente.**

## **Vantagens da forma alternativa**

### **Economia de código**

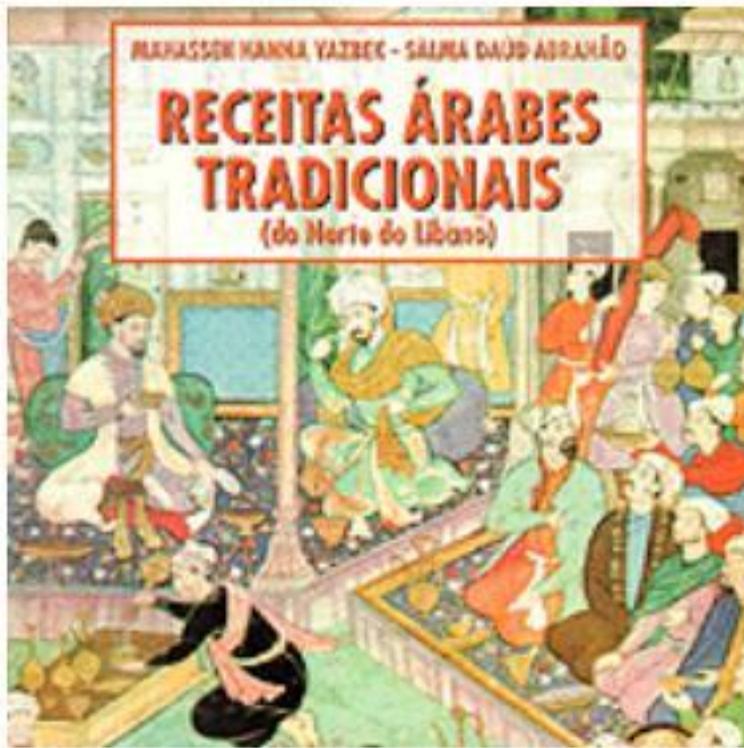
*O livro (programa) vai ficar menor*

### **Modularidade**

*Se amanhã descubro que um dente de alho dá um sabor especial ao molho branco, basta alterar o livro de receitas em um único lugar.*

## **Mais vantagens: divisão de trabalho**

*O livro de receitas pode ser escrito por 2 autores.  
Basta que um deles aprenda a preparar o peixe e  
o outro a preparar o molho branco.*



**cederj**

## **Mais vantagens: utilização de componentes prontos**

### **➤ Filé de peixe com molho branco**

**{preparo dos peixes}**

Lave os filés e tempere com o suco dos limões, sal, pimenta e salsinha ...

**{preparo do molho branco}**

**Vá até o supermercado e compre molho branco Parmalat (pronto)**

**{juntando os dois}**

Adicione queijo parmesão ralado e queijo gruyère. Misture e ponha sobre os filés.

## ***Mais vantagens: reaproveitamento de código***

### **➤ Filé de peixe com molho branco**

**{preparo dos peixes}**

Lave os filés e tempere com o suco dos limões, sal, pimenta e salsinha ...

**{preparo do molho branco}**

**Prepare o molho branco segundo receita do meu livro anterior, página 35**

**{juntando os dois}**

Adicione queijo parmesão ralado e queijo gruyère. Misture e ponha sobre os filés.

**cederj**

## ***Mais vantagens: parametrização dos procedimentos***

### **MACARRÃO COM OVOS BATIDOS**

#### **MODO DE FAZER:**

Cozinhe o macarrão em água abundante, fervente, com sal. Escorra. Coloque a manteiga numa panela e leve ao fogo. Depois de derretida, junte o macarrão já cozido e, em seguida, os ovos batidos. Misturando bem e abaixe o fogo. Levante devagar, com a ponta de uma faca, o macarrão, até que cozinhe os ovos. Despeje em uma travessa e polvilhe com o queijo ralado.

## **Parametrização dos procedimentos**

- A receita não faz referência ao tipo de macarrão...



## **Parametrização dos procedimentos**

### ➤ **Cálculo da área de uma mesa**

Mede o comprimento da mesa

Mede a largura da mesa

Multiplica o comprimento da mesa pela largura da mesa

Diz o resultado



**cederj**

## **Parametrização dos procedimentos**

### ➤ **Cálculo da área da toalha sobre a mesa**

**Mede o comprimento da toalha**

**Mede a largura da toalha**

**Multiplica o comprimento da toalha pela largura da  
toalha**

**Diz o resultado**



**cederj**

# **Parametrização dos procedimentos**

- **Procedimento para o cálculo de objeto retangular**

**Procedimento area (coisa)**

**inicio**

    Mede o comprimento da **coisa**

    Mede a largura da **coisa**

    Multiplica o comprimento da **coisa**

        pela largura da **coisa**

    Diz o resultado

**fim**

## **Parametrização dos procedimentos**

**Para calcular a área da mesa**

**area (mesa)**

**Para calcular a área da toalha**

**area (toalha)**

*coisa é um objeto virtual. coisa se materializa em mesa ou toalha na hora em que mando calcular a área → mesmo procedimento atuando sobre diferentes dados.*

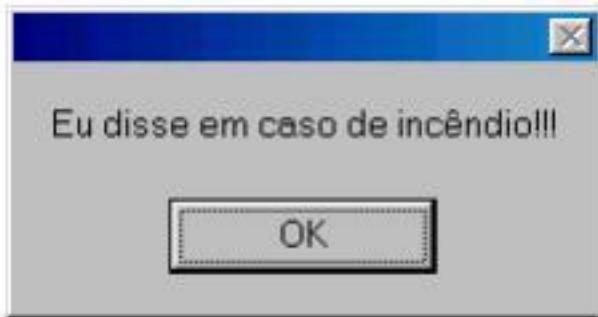
**cederj**

## **Necessidade do uso**

- **Procedimentos são a base das linguagens visuais**



## **Necessidade do uso**



### ***Tratamentos de eventos escritos na forma de procedimentos***

```
Sub Botao_Click()
    MsgBox "Eu disse em caso de incêndio!!!"
End Sub
```

## **Necessidade do uso**

**Procedimentos (métodos) são também a base da orientação a objeto**

```
TMesa = class
    largura : real;
    comprimento : real;
    function CalculaArea : real;
end;
```

## **Abordagem OO**

**Propriedades e métodos são encapsulados no objeto → o objeto sabe como lidar com suas propriedades.**

**mesa.CalculaArea**

## ***Mas qual é a vantagem na utilização de OO?***

***Muitas. Posso por "inteligência" na mesa visando a proteção e integridade dos dados***

```
mesa.comprimento := -2.2;
```

### ***Em Delphi:***

```
procedure TMesa.SetComprimento(Value: real);
begin
  if Value <=0 then
    raise Exception.Create('Valor Inválido');
  comprimento := Value;
end;
```

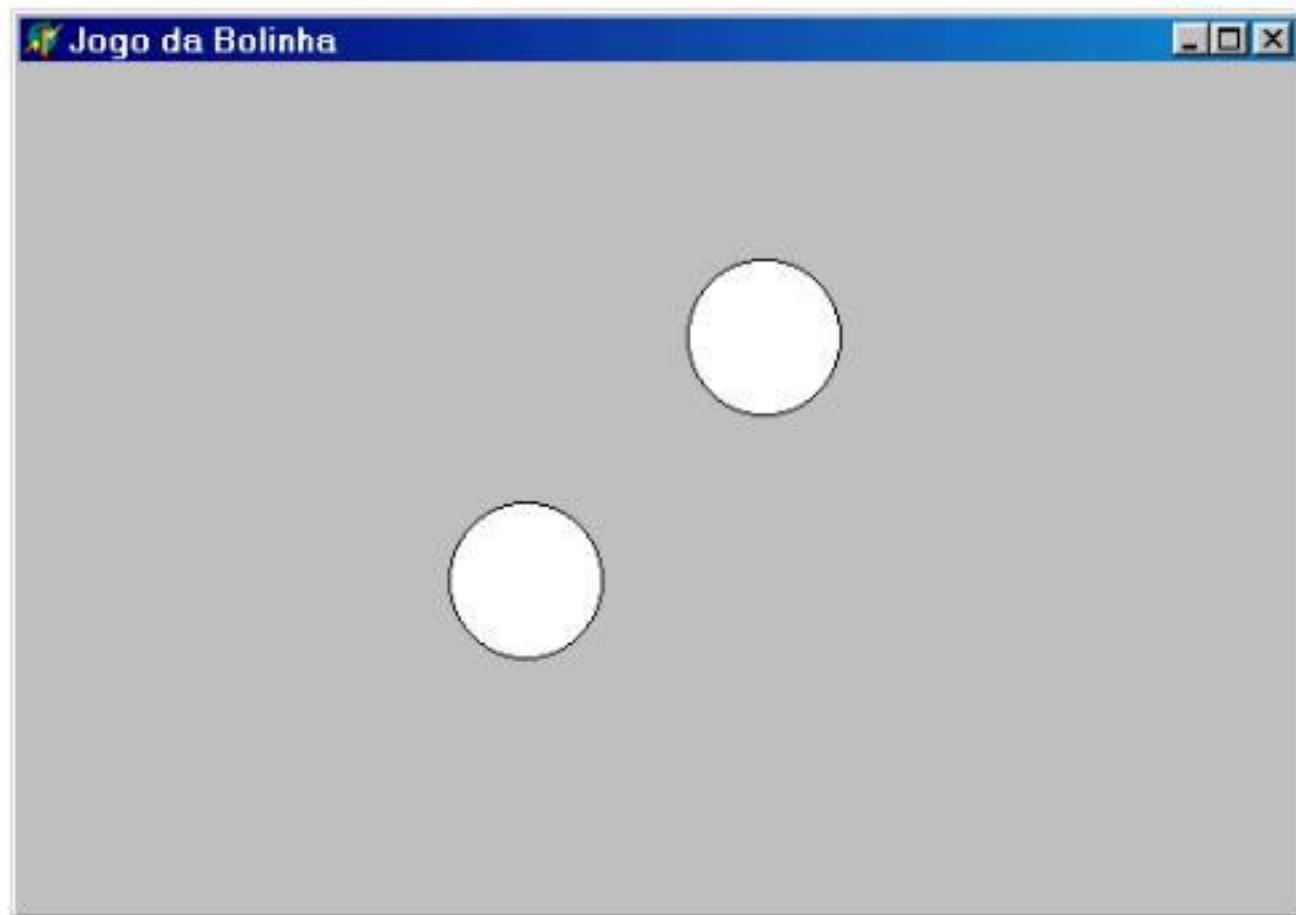
**Classe??? Orientação a Objeto???**  
**Mas para que serve isso?**



---

**cederj**

# ***Jogo da Bolinha***



**cederj**

# **Como implementar?**

## ***Abordagem tradicional (PETEQS)***

```
acabou ← falso
enquanto nao acabou faca
    para i ← 1 ate 2 faça
        anda (bolinha[i])
        tratacolisao (bolinha[i])
    proximo i
fim enquanto
```

## **Conclusão**

- *A utilização de procedimentos está na base da construção de bons padrões de programação.*
- *A utilização de procedimentos é essencial ao uso das chamadas linguagens visuais*
- *A utilização de procedimentos é essencial à programação orientada a objetos.*

# **Projeto e Desenvolvimento de Algoritmos**

**Procedimentos**

**Utilização**

**Adriano Cruz e Jonas Knopman**

**cederj**

---

# Objetivos

- Sintaxe
- Variáveis locais e variáveis públicas
- Escopo de variáveis
- Exemplos

# *Forma geral de um programa em PETEQS contendo procedimentos*

```
procedimento A
inicio
{ Comandos }
Fim
```

```
programa teste
inicio
{ Comandos }
fim
```

## ***Chamada de Procedimentos***

*A chamada a um procedimento é feita simplesmente escrevendo-se o nome do procedimento no corpo do módulo (programa principal ou outro procedimento) que efetuou a chamada.*

## **Exemplo de uso de procedimentos**

```
procedimento Escreva_uma_mensagem
    inicio
        imprima 'O contador vale:', i
    fim
```

```
inicio
    para i ← 1 até 5 faça
        Escreva_uma_mensagem
    próximo i
fim
```

Observe o uso da variável contadora!

**cederj**

## ***Exemplo de uso de procedimentos***

### › Saída:

```
O contador vale: 1  
O contador vale: 2  
O contador vale: 3  
O contador vale: 4  
O contador vale: 5
```

## *Escopo de variáveis*

- *Se dois módulos de programa cooperam de forma a resolver um problema, eles têm de trocar dados de alguma forma*
- *O mecanismo mais imediato (e mais perigoso) para a troca de dados é a utilização de variáveis públicas*

# **Conversão de temperaturas**

**Procedimento** Converte

// entrada: temperatura Fahrenheit (Far)

// saída: temperatura Celsius (Cel)

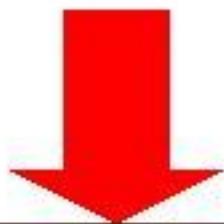
**inicio**

Cel ← (Far - 32) / 1.8

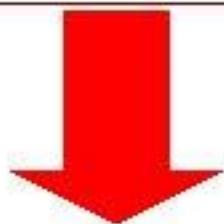
**fim**

# *Variáveis Públicas*

Far



Converte



Cel

cederj

# **Conversão de temperaturas**

**variáveis públicas**

**Far, Cel**

**programa teste**

**inicio**

**imprima** 'Temperatura em Fahrenheit: '

**leia** Far

Converte

**imprima** 'Celsius: ', Cel

**fim**

## *Conversão de temperaturas*

› Saída:

Temperatura em Fahrenheit: 212

Celsius: 100.00

## Variáveis Públicas

- As variáveis públicas são vistas por todos os módulos do programa (procedimentos e funções), incluindo o programa principal.



cederj

# **Variáveis Públicas**

## ➤ **Vantagens**

*Facilidade de Uso – Por serem públicas, tais variáveis servem para a troca de informações entre 2 módulos quaisquer do programa.*

# **Variáveis Públicas**

## ➤ **Desvantagens**

*Justamente por serem acessíveis a qualquer módulo, as variáveis públicas apresentam um risco de que outros módulos não envolvidos na troca de dados alterem (inadvertidamente ou não) os dados compartilhados.*

*Falta de proteção dos dados compartilhados. Não há como evitar que mesmo os módulos envolvidos na comunicação alterem equivocadamente dados que ainda seriam necessários a outros módulos*

# **Variáveis Públicas**

## ➤ *Desvantagens*

*Os módulos envolvidos na comunicação têm de usar os mesmos nomes para as variáveis compartilhadas.*

## *Escopo de variáveis*

*O que acontece se eu não declarar as variáveis de entrada/saída como públicas?*

# **Conversão de Temperaturas**

**Procedimento** Converte

```
// entrada: temperatura Fahrenheit (Far)  
// saída: temperatura Celsius (Cel)
```

**inicio**

```
    Cel ← (Far - 32) / 1.8
```

**fim**

**programa** teste

**inicio**

```
    imprima 'Temperatura em Fahrenheit: '
```

```
    leia Far
```

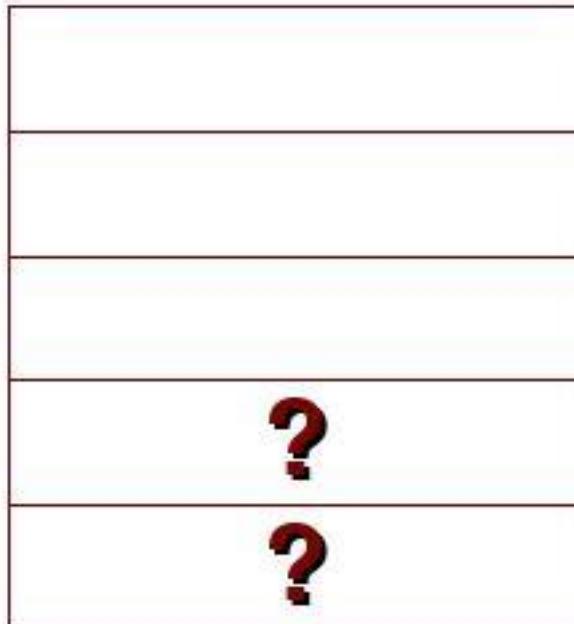
```
    Converte
```

```
    imprima 'Celsius: ', Cel
```

**fim**

# **Esquema de alocação de memória**

*Antes de iniciar-se a execução do programa: O sistema operacional aloca em memória espaço para as variáveis utilizadas pelo programa principal*



Cel:teste

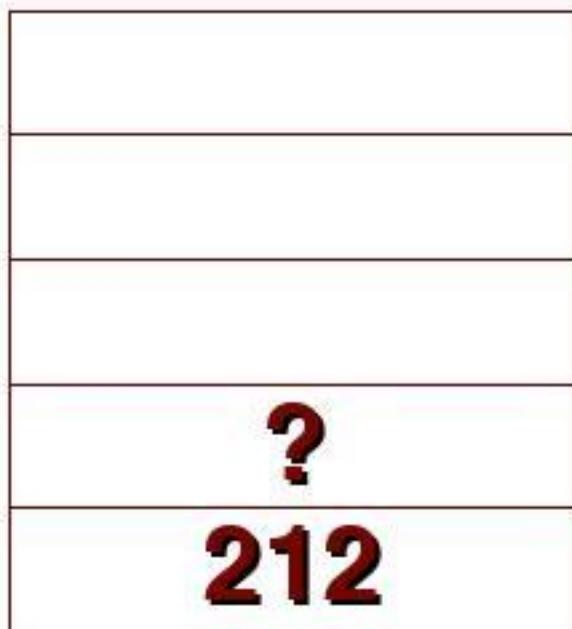
Far:teste

**cederj**

# ***Esquema de alocação de memória***

*Começa a execução do programa principal*

**leia Far**



Cel:teste

Far:teste

**cederj**

# ***Esquema de alocação de memória***

Converte

*Antes de começar a executar o procedimento*

*Converte, o SO aloca as variáveis de Converte*

?
?
?
<b>212</b>

Cel:Converte

Far:Converte

Cel:teste

Far:teste

**cederj**

# **Esquema de alocação de memória**

*Começa a execução de Converte*

**Cel ← (Far - 32) / 1.8**

?
?
?
<b>212</b>

Cel:Converte

Far:Converte

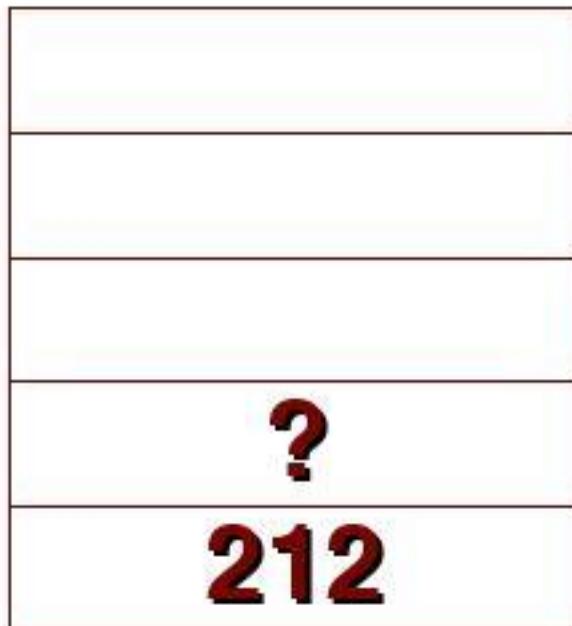
Cel:teste

Far:teste

**cederj**

# ***Esquema de alocação de memória***

***Termina a execução de Converte***



Cel:teste

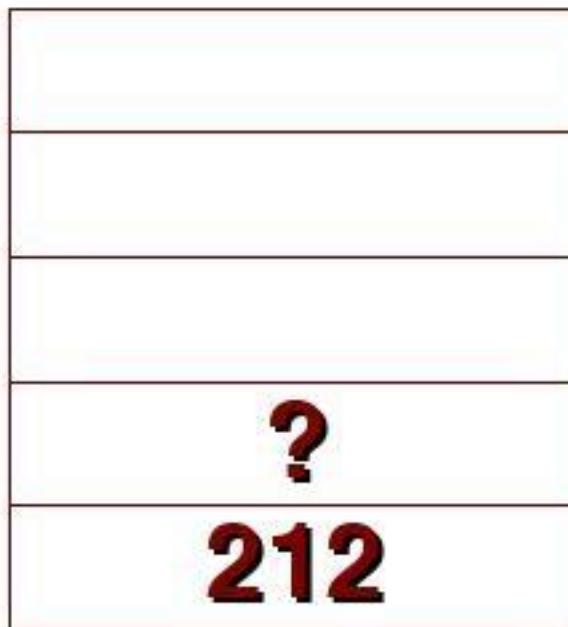
Far:teste

**cederj**

## **Esquema de alocação de memória**

*Prossegue a execução do programa principal*

imprima 'Celsius' na Cel  
O resultado dessa instrução é imprevisível!!!



Cel:teste

Far:teste

**cederj**

## **Um exemplo**

*Escreva um procedimento que calcule o peso ideal de uma pessoa:*

**homens:**

$$\text{peso} = 72.7 * \text{altura} - 58$$

**mujeres**

$$\text{peso} = 62.1 * \text{altura} - 44.7$$

## **Procedimento CalculaPeso**

- Vamos determinar o conjunto de entradas e saídas do procedimento *CalculaPeso*



# **Procedimento CalculaPeso**

- **Entradas**

- sexo*

- altura*

- **Saída**

- peso*

## **Procedimento CalculaPeso**

**Variáveis públicas**

**sexo, altura, peso**

**procedimento CalculaPeso**

**inicio**

**se sexo = 'M' entao**

**peso ← 72.7\*altura - 58**

**senão**

**peso ← 62.1\*altura - 44.7**

**fim se**

**fim**

## *Um programa que use CalculaPeso*

```
programa teste
inicio
    imprima 'Entre com a altura: '
    leia altura
    imprima 'Entre com o sexo: '
    leia sexo
    CalculaPeso
    imprima 'Peso ideal: ', peso
fim
```

# **CalculaPeso**

## › **Saida:**

Entre com a altura: 1.62

Entre com o sexo: F

Peso ideal: 55.9

# **Projeto e Desenvolvimento de Algoritmos**

**Procedimentos**

**Passagem de Parâmetros**

**Adriano Cruz e Jonas Knopman**

**cederj**

---

# *Procedimentos*

## *Passagem de Parâmetros*

---

**cederj**

# *Objetivos*

*Definição*

*Vantagens da utilização de parâmetros*

*Parâmetros de entrada (Passagem por valor)*

*Parâmetros de saída (Passagem por referência)*

*Exemplos*

# *Por que parâmetros?*

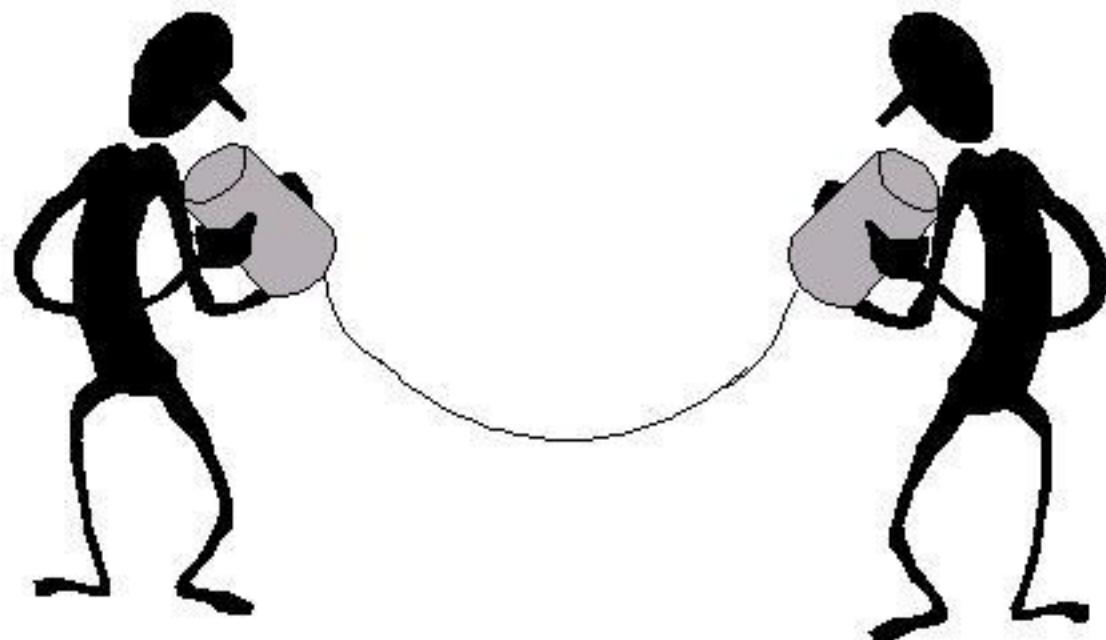
O uso de variáveis públicas expõem perigosamente os dados!



**cederj**

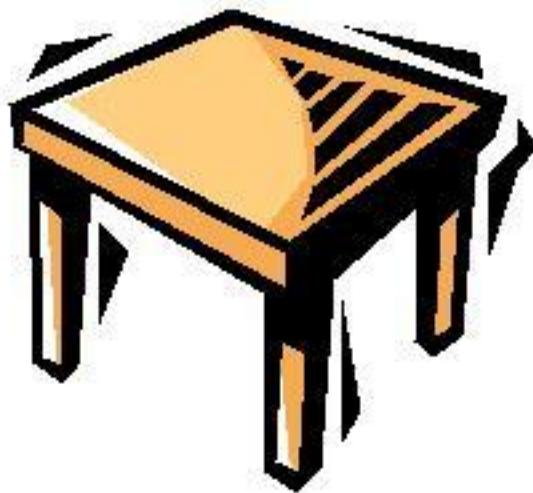
## **Parâmetros**

*O uso de parâmetros permite a comunicação privativa entre as partes*



**cederj**

## *Parametrização de Procedimentos*



X



X



**cederj**

## **Entradas e Saídas**

*Um procedimento é um bloco de código que opera sobre um conjunto de dados (suas entradas) de modo a produzir algum resultado útil para alguém (suas saídas)*



## **Entradas e Saídas**

- *O conjunto de entradas e saídas define a necessidade de comunicação do procedimento.*
- *Na seção anterior, usamos este conjunto para definir as variáveis públicas do programa (solução perigosa!!!).*
- *Este mesmo conjunto define os parâmetros do procedimento que têm de ser declarados (solução melhor!!!)*

## **Entrada e Saída**

- Os parâmetros de entrada são chamados de parâmetros passados por valor
- Os parâmetros de saída são chamados de parâmetros passados por referência

## **Sintaxe**

**procedimento**    NONONO (

**entradas:** NONO, NONONO, NONO, ...

**saidas:** NONO, NONONO, NONO, ....)

**inicio**

...

...

**fim**

## **Um exemplo**

*Um procedimento para converter uma quantia em dólares para o equivalente em reais*



## **Procedimento Converte**

**procedimento** Converte (

    entradas: USD

    saídas: BRL)

**inicio**

    BRL ← 2.4 \* USD

**fim**

## **Procedimento Converte**

- A declaração do procedimento e a relação de suas entradas e saídas define o **contrato de utilização do procedimento**

**procedimento** Converte (

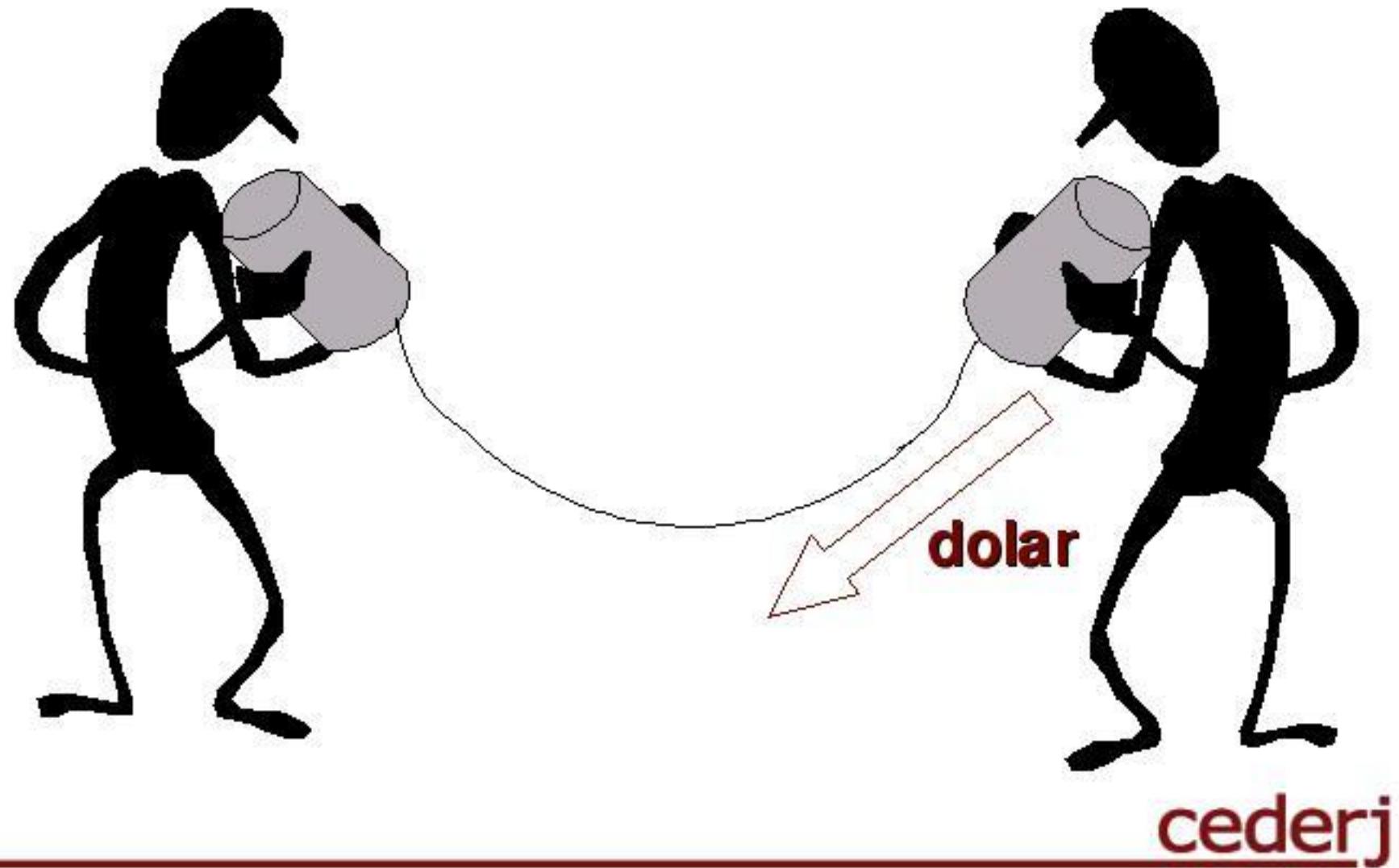
- entradas: USD
- saídas: BRL)

- Qualquer módulo que queira usar o procedimento Converte, tem de respeitar o **contrato de utilização**.

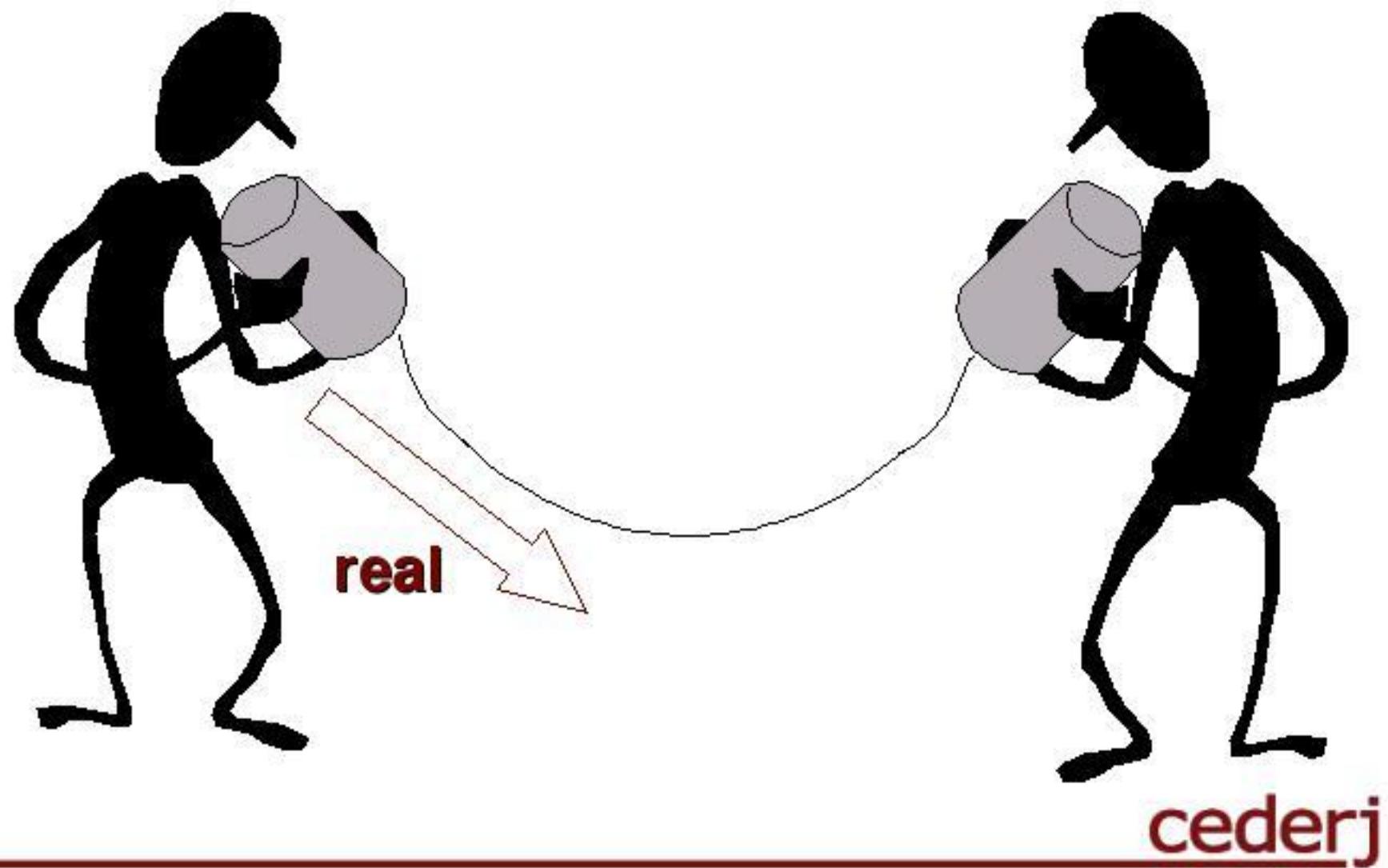
## **Exemplo de uso do procedimento Converte**

```
programa teste
  inicio
    acabou ← falso
    enquanto não acabou faça
      leia dolar
      se dolar > 0 então
        Converte (dolar, real)
        imprime real
      senão
        acabou ← verdadeiro
      fim se
    fim enquanto
  fim
```

## *O procedimento Converte*



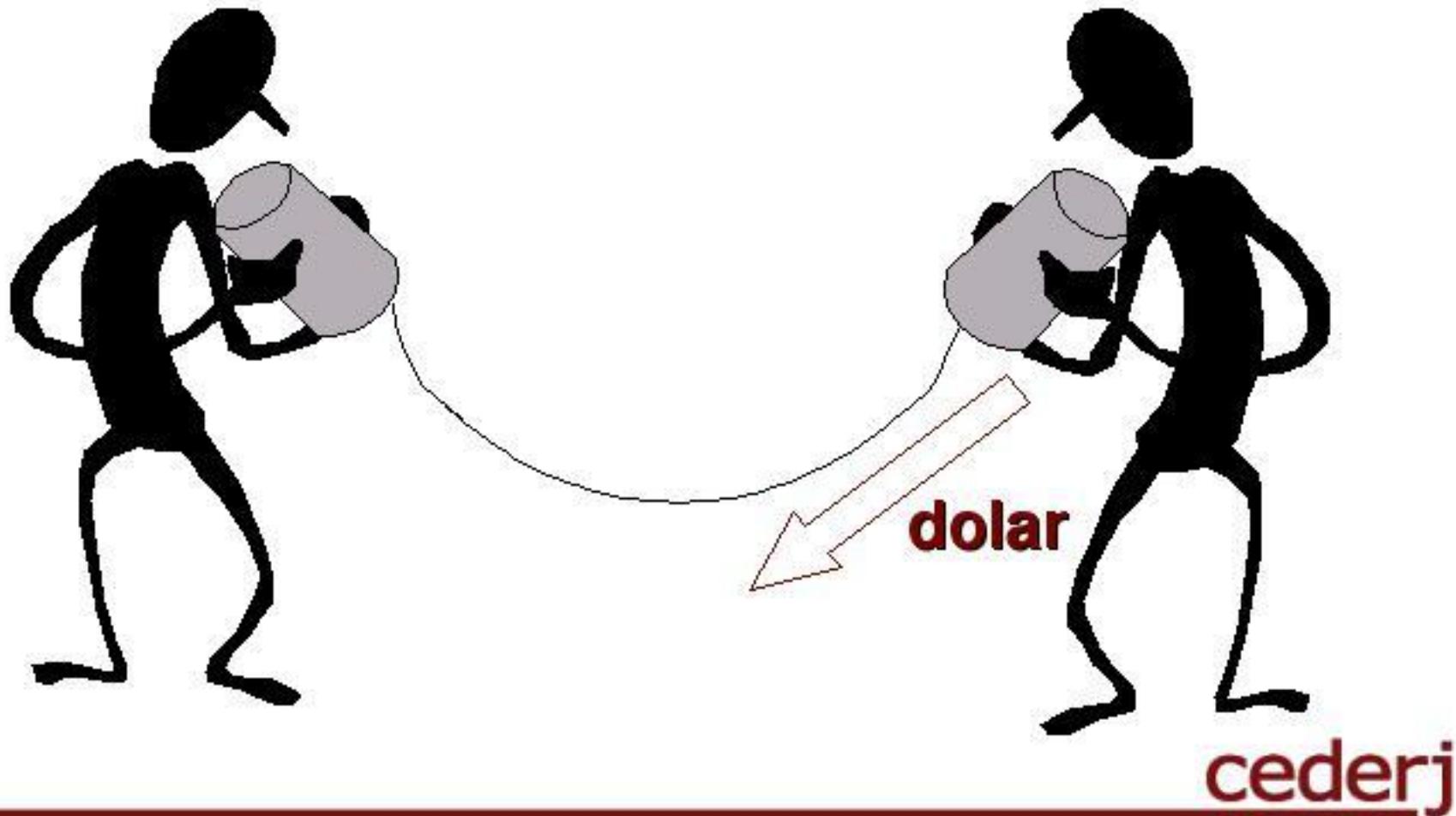
# *O procedimento Converte*



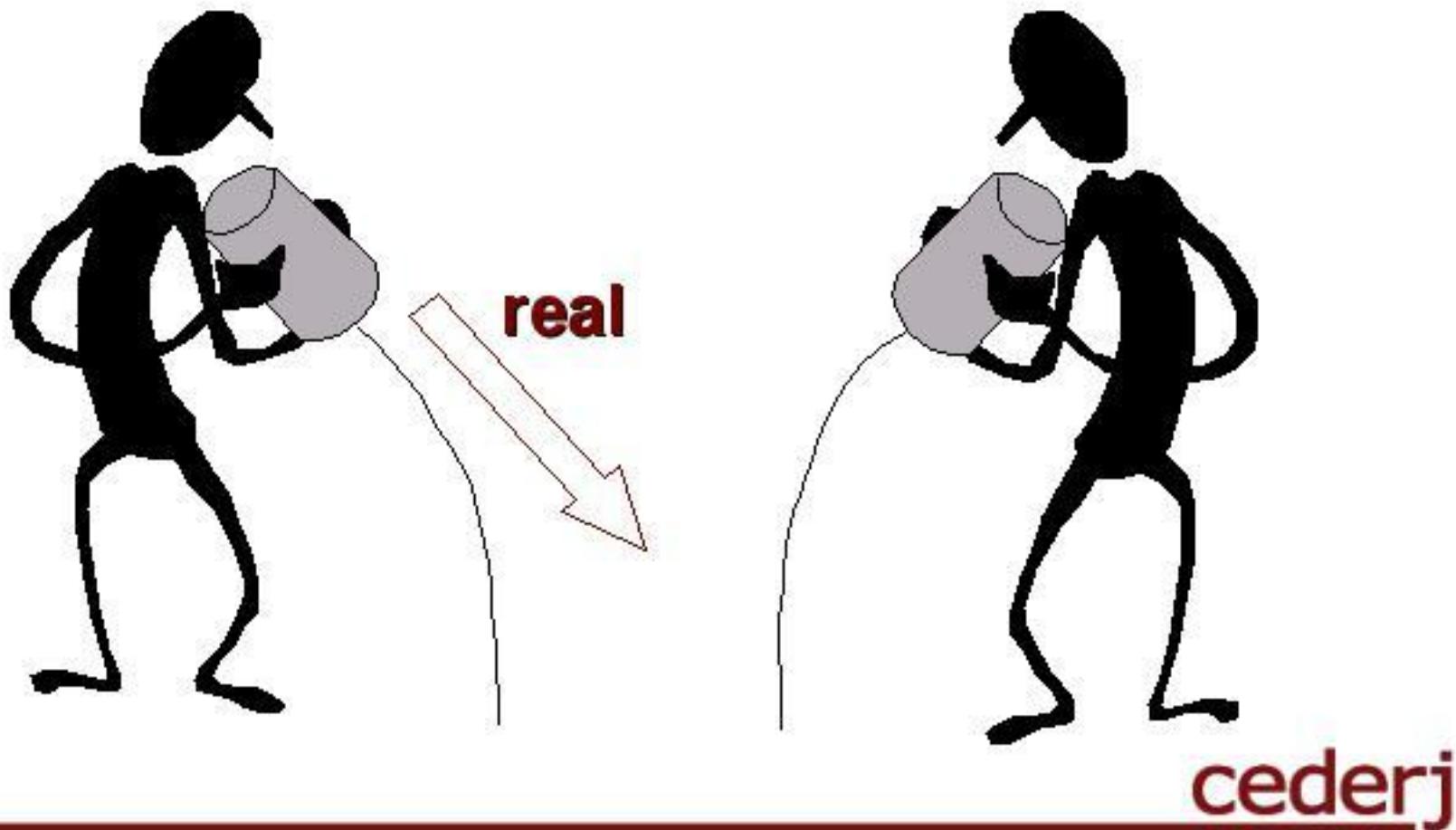
## **Passagem de Parâmetros**

- *O uso de nomes diferentes nas chamadas à procedure e na definição dos procedimentos não apresenta problemas → o número de argumentos deve ser o mesmo.*
- *Os parâmetros são sobrepostos no instante da chamada: o primeiro com o primeiro, o segundo com o segundo, etc.*

**O que acontece se eu declarar todos os  
parâmetros como parâmetros de  
entrada?**



**O que acontece se eu declarar todos os  
parâmetros como parâmetros de  
entrada?**



## **Um exemplo: O procedimento swap**

```
procedimento swap (entradas: x, y)
```

```
inicio
```

```
    imprima 'ponto2> x:',x,' y:',y
```

```
    temp ← x
```

```
    x ← y
```

```
    y ← temp
```

```
    imprima 'ponto3> x:',x,' y:',y
```

```
fim
```

## **Exemplo de utilização da função swap**

```
programa teste
  inicio
    x ← 5
    y ← 10
    imprima 'ponto1> x:',x,' y:',y
    swap(x,y)
    imprima 'ponto4> x:',x,' y:',y
  fim
```

## ***Exemplo de utilização da função swap***

**Saída:**

```
ponto1> x:5  y:10
```

```
ponto2> x:5  y:10
```

```
ponto3> x:10 y:5
```

```
ponto4> x:5  y:10
```

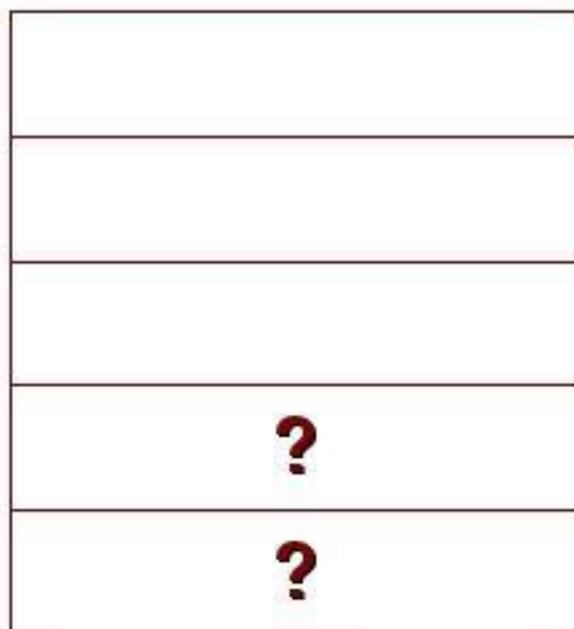
*Por que isto aconteceu???*



cederj

## ***Esquema de alocação em memória***

*Antes de iniciar-se a execução do programa o sistema operacional aloca em memória espaço para as variáveis utilizadas pelo programa principal*



x: teste

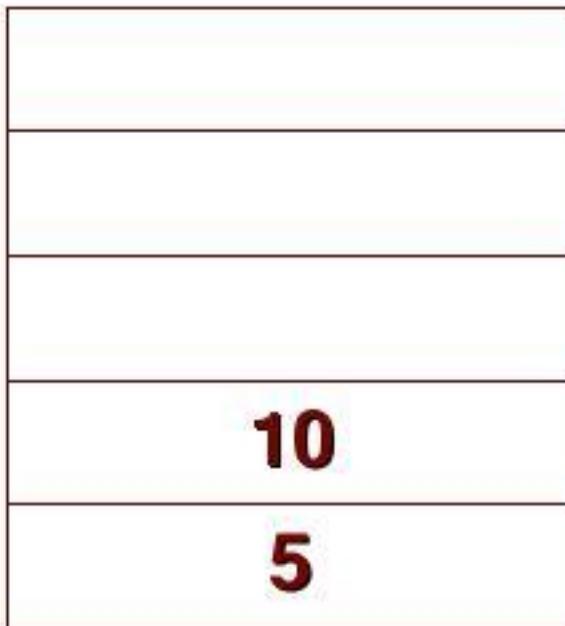
y: teste

# *Esquema de alocação em memória*

*Começa o programa principal*

$x \leftarrow 5$

$y \leftarrow 10$



y:teste

x:teste

**cederj**

## ***Esquema de alocação em memória***

*chamada do procedimento swap*

swap (x, y)

## **Esquema de alocação em memória**

*Antes de começar a executar o procedimento swap,  
o sistema operacional aloca espaço em memória  
para os parâmetros e para as variáveis locais*

10
5
?
10
5

y:swap

x:swap

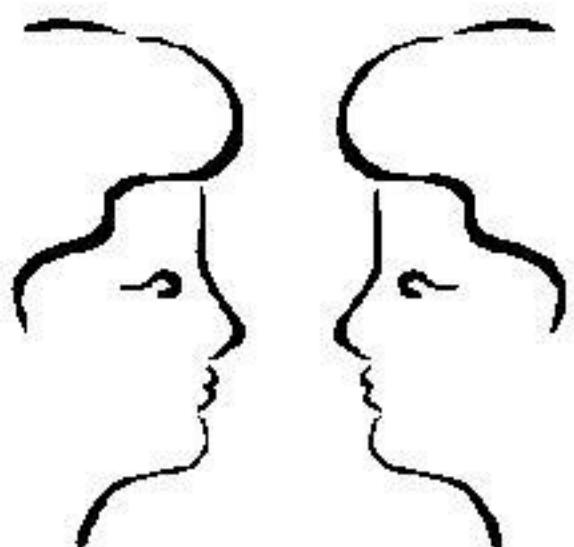
temp

y:teste

x:teste

## ***Esquema de alocação em memória***

*Encare a passagem de parâmetros por valor  
(parâmetros de entrada) como uma cópia xerox do  
dado original.*



## ***Esquema de alocação em memória***

*começa a execução do procedimento swap*

`temp ← x`

10
5
5
10
5

y:swap

x:swap

temp

y:teste

x:teste

**cederj**

## *Esquema de alocação em memória*

$x \leftarrow y$

10
10
5
10
5

Por isso preciso da variável auxiliar

y:swap

x:swap

temp

y:teste

x:teste

cederj

## ***Esquema de alocação em memória***

$y \leftarrow temp$

5
10
5
10
5

y:swap

x:swap

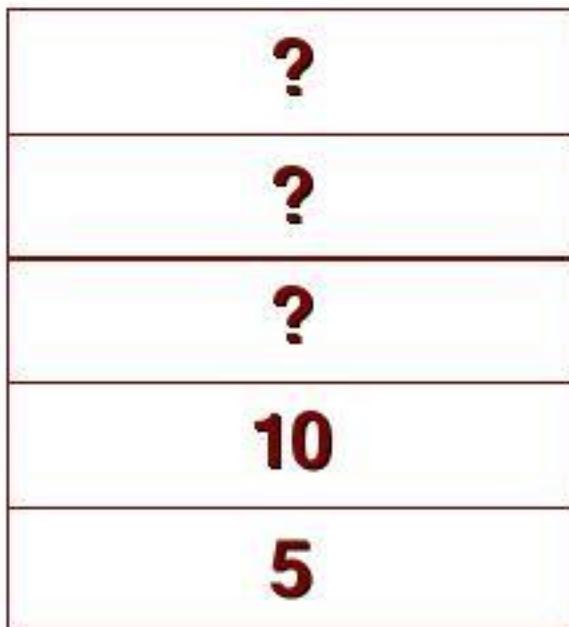
temp

y:teste

x:teste

# ***Esquema de alocação em memória***

*termina o procedimento swap*



y:teste

x:teste

*As variáveis x e y do programa teste permanecem  
inalteradas!!!*

**cederj**

***Qual é a solução???***

*Passar o próprio dado e não uma cópia deste.*



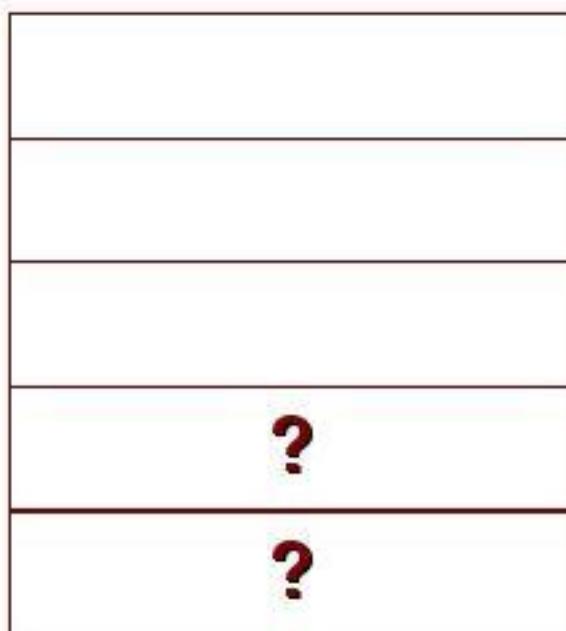
**cederj**

## Solução

```
procedimento swap (saidas: x, y)
inicio
    imprima 'ponto2> x:', x, ' y:', y
    temp ← x
    x ← y
    y ← temp
    imprima 'ponto3> x:', x, ' y:', y
fim
```

# ***Esquema de alocação em memória***

*Antes de iniciar o programa teste...*



x: teste

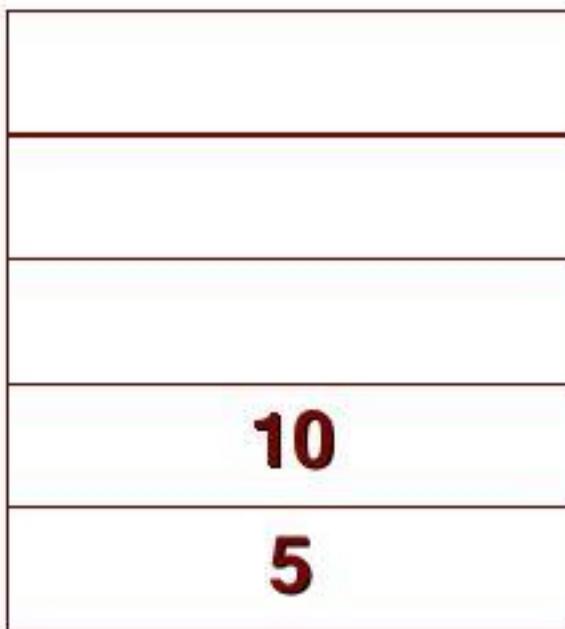
y: teste

# **Esquema de alocação em memória**

*Começa o programa principal*

$x \leftarrow 5$

$y \leftarrow 10$



y:teste

x:teste

**cederj**

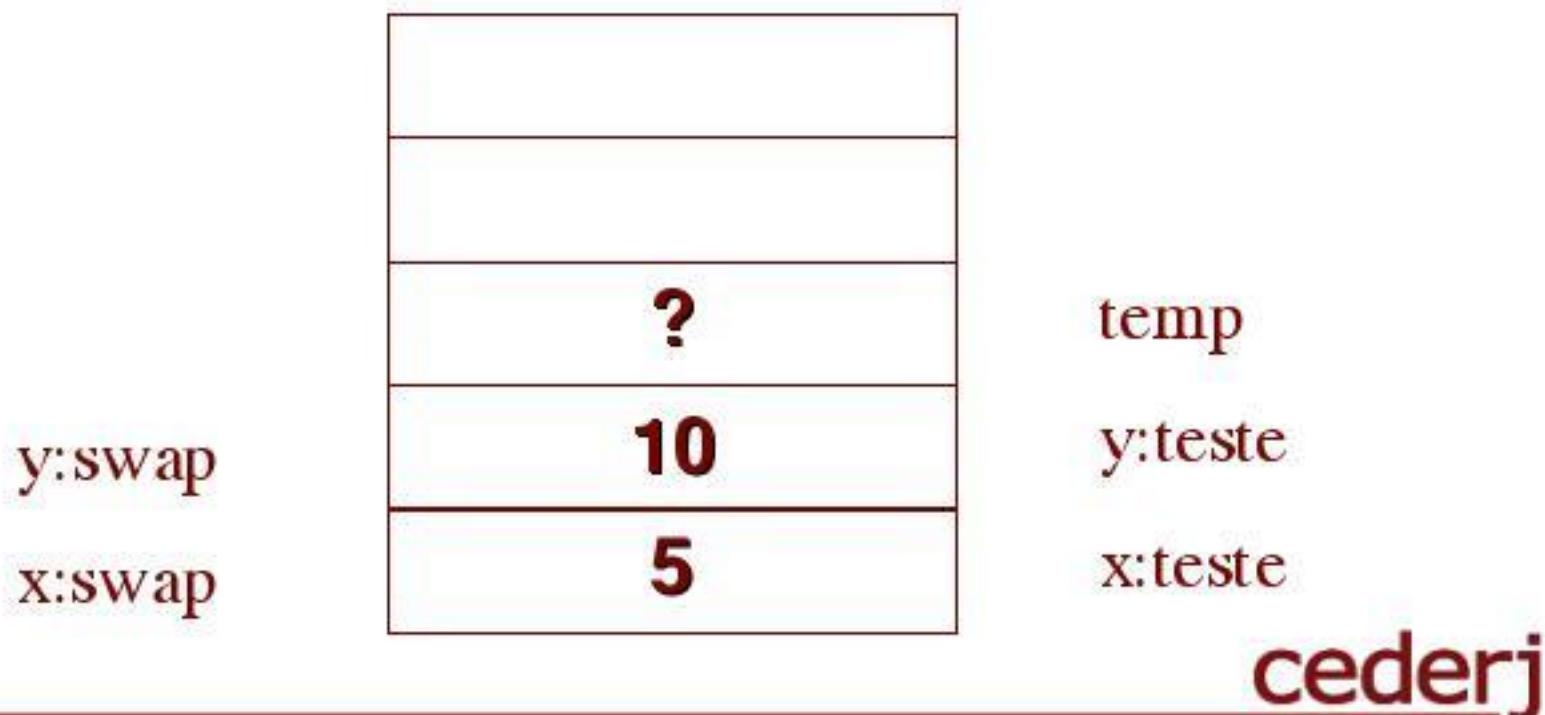
# ***Esquema de alocação em memória***

***chamada do procedimento swap***

swap (x, y)

## **Esquema de alocação em memória**

*Antes de começar a executar o procedimento swap,  
o sistema operacional aloca espaço em memória  
para os parâmetros e para as variáveis locais*



## ***Esquema de alocação em memória***

*Observe como agora o dado é compartilhado...*

*Ao invés de se entregar ao procedimento uma cópia do dado original, entrega-se o próprio dado*

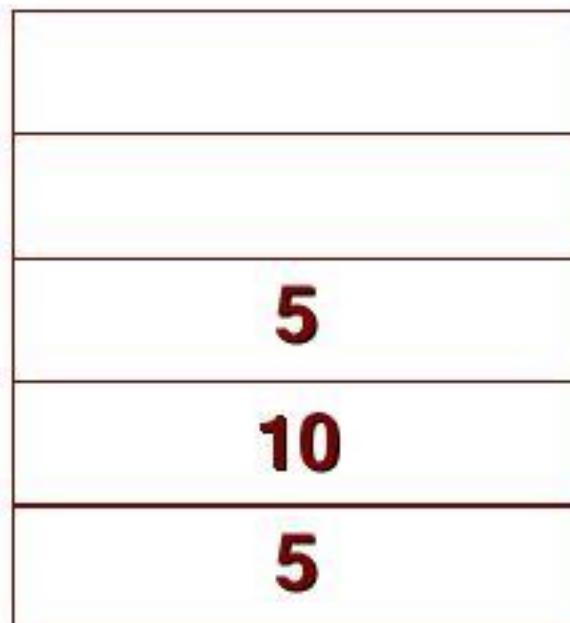
*É verdade, existe um risco no compartilhamento.  
Mas, para dados de saída, é o único jeito...*

## ***Esquema de alocação em memória***

*começa a execução do procedimento swap*

`temp ← x`

y:swap  
x:swap



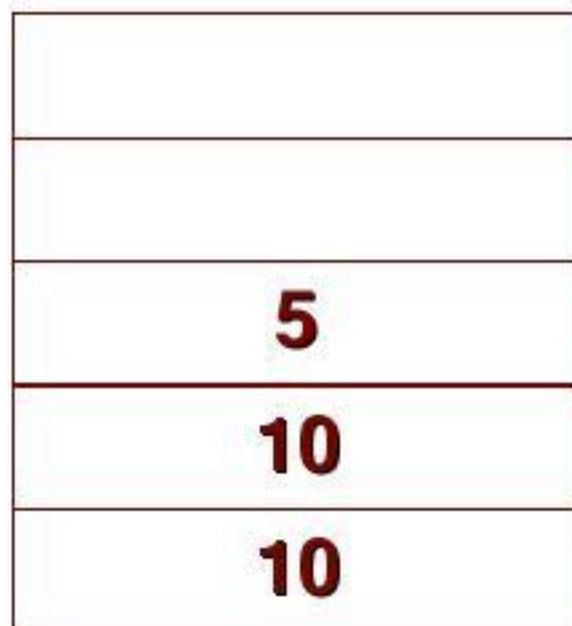
temp  
y:teste  
x:teste

## *Esquema de alocação em memória*

$x \leftarrow y$

y:swap

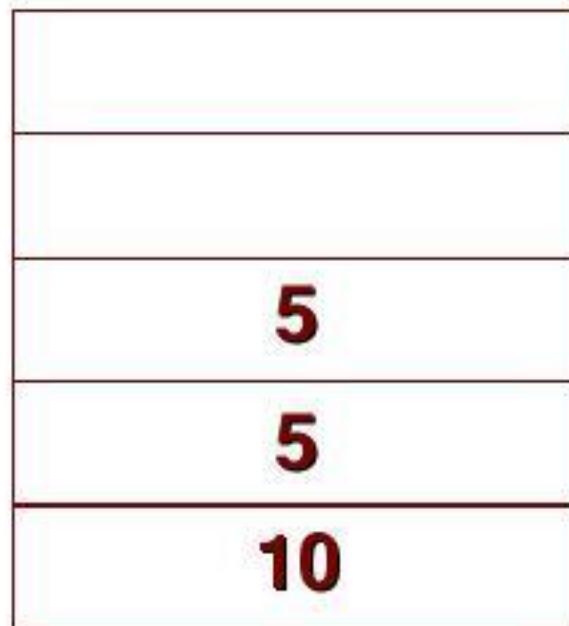
x:swap



## *Esquema de alocação em memória*

$y \leftarrow \text{temp}$

y:swap  
x:swap



## ***Esquema de alocação em memória***

*termina o procedimento swap*

*As variáveis x e y do procedimento swap são destruídas, mas o seu conteúdo permanece acessível através das variáveis do programa teste.*

?
5
10

y:teste

x:teste

**cederj**

## **Valor ou referência???**

*Ok então. Preciso então usar parâmetros de saída...*

*Mas e se eu declarar todos os parâmetros como parâmetros de saída? Não funciona? Eu compartilharia os dados referentes a todos os parâmetros e não somente aqueles correspondentes aos parâmetros de saída...*



## **Valor ou referência???**

*Existem 2 bons motivos para usar chamadas por valor:*

- *Privacidade dos dados*
- *A possibilidade de passar constantes como argumentos de funções*

```
x ← raiz(2)
```

## **Outro exemplo: O procedimento CalculaPeso**

Escreva um procedimento que calcule o peso ideal de uma pessoa:

*homens:*

$$\text{peso} = 72.7 * \text{altura} - 58$$

*muitas*

$$\text{peso} = 62.1 * \text{altura} - 44.7$$

# Procedimento CalculaPeso

Vamos determinar o conjunto de entradas e saídas do procedimento CalculaPeso



# Procedimento CalculaPeso

- ***Entradas***

- sexo*

- altura*

- ***Saída***

- peso*

# **Solução utilizando variáveis públicas**

## **Variáveis públicas**

sexo, altura, peso

**procedimento** CalculaPeso

**inicio**

**se** sexo = 'M' **entao**

    peso  $\leftarrow$  72.7 \* altura - 58

**senão**

    peso  $\leftarrow$  62.1 \* altura - 44.7

**fim se**

**fim**

**Contrato**

**cederj**

## Um programa que use CalculaPeso

```
programa teste
  inicio
    imprima 'Entre com a altura: '
    leia altura
    imprima 'Entre com o sexo: '
    leia sexo
    CalculaPeso
    imprima 'Peso ideal: ', peso
  fim
```

## **Solução utilizando parâmetros**

```
procedimento CalculaPeso (
    entradas: sexo, altura
    saídas: peso)
inicio
    se sexo = 'M' entao
        peso ← 72.7*altura - 58
    senão
        peso ← 62.1*altura - 44.7
    fim se
fim
```

**Contrato**

**cederj**

## Um programa que use CalculaPeso

```
programa teste
  inicio
    imprima 'Entre com a altura: '
    leia a
    imprima 'Entre com o sexo: '
    leia s
    CalculaPeso (s, a, p)
    imprima 'Peso ideal: ', p
  fim
```

# *Funções*

cederj

# **Objetivos**

*Definição*

*Exemplos*

*Necessidade e benefícios na utilização de funções*

*Passagem de parâmetros para funções (esquema de alocação de memória)*

*Vetores e funções*

**cederj**

# **Funções**

*Funções são blocos de programas que retornam um valor.*

```
x := sqrt(y);
```

```
linha := FloatToStr(x);
```

```
y := 4 * abs(x);
```

# Funções

*O bloco de comandos deve conter pelo menos uma instrução que atribui o valor de retorno da função.*

## Sintaxe:

```
função nome(entradas: parâmetros de entrada  
            saídas: parâmetros de saída)  
inicio  
    ...  
    resultado ← ...  
fim
```

## **Funções – Exemplo**

```
função noPatas(entradas: N1,N2)
    inicio
        resultado ← 4 * (N1 + N2)
    fim

programa teste
    inicio
        caes ← 4
        gatos ← 3
        imprima 'Total de patas: ', noPatas(caes, gatos)
    fim
```

## *A função noPatas em Pascal*

```
function noPatas(N1, N2 : integer): integer;
begin
    result := 4 * (N1+N2);
end;

var
    caes, gatos : integer;
begin
    caes := 4;
    gatos := 3;
    writeln('Total de patas: ',
            noPatas(caes, gatos));
end.
```

## **A função noPatas em Java**

```
public class Patas {  
    public static int noPatas(int N1, int N2) {  
        return 4*(N1+N2);  
    }  
  
    public static void main(String args[]) {  
        int caes = 4;  
        int gatos = 3;  
        System.out.println("Total de patas: " +  
            noPatas(caes, gatos));  
    }  
}
```

# **Funções**

*Mas é necessário que eu use funções em meus programas?*



*Sim e Não. ?!?!?!?!*

**cederj**

# **Funções**

*É necessário que eu use funções em meus programas?*

*Sim, se as funções foram escritas por algum outro programador!*

*Se você está escrevendo seus próprios módulos de software (procedimentos ou funções), o uso de funções não é obrigatório, ainda que ele apresente algumas vantagens.*

**cederj**

## ***noPatas como um Procedimento***

```
procedimento noPatas (entradas: N1,N2  
                      saidas: patas)
```

```
inicio
```

```
    patas ← 4 * (N1 + N2)
```

```
fim
```

```
programa teste
```

```
inicio
```

```
    caes ← 4
```

```
    gatos ← 3
```

```
    noPatas(caes, gatos, pes)
```

```
    imprima 'Total de patas: ', pes
```

```
fim
```

# **Funções**

*O uso de funções traz, no entanto, algumas vantagens:*

- *É possível utilizar a chamada da função diretamente em expressões:*

`imprima 'Total de patas: ', noPatas(caes, gatos)`

*ao invés de:*

`noPatas(caes, gatos, pes)`

`imprima 'Total de patas: ', pes`

## **Vantagens na utilização de funções**

*Além disso, em construções do tipo:*

`y ← raiz (x)`

*é mais fácil perceber que y é a raiz de x do que na construção:*

`raiz (x, y)`

*onde é impossível dizer, sem conhecer a função, se y é raiz de x ou x é raiz de y*

**cederj**

## **Outro exemplo: A função converte**

```
função Converte(entradas: Fer)
início
    resultado ← (Fer - 32) * 5 / 9
fim

programa teste
início
    imprima 'Fahrenheit: '
    leia Far
    Cel ← Converte(Far)
    imprima 'Celsius: ', Cel
fim
```

# **A função converte**

**Saída:**

Fahrenheit: 212

Celsius: 100.00

## **A função Converte em Pascal**

```
function converte(Fer : real): real;
begin
    result := (Fer - 32) * 5 / 9;
end;

var
    Far, Cel : real;
begin
    write ('Fahrenheit: ');
    readln(Far);
    Cel := converte(Far);
    writeln('Celsius: ', Cel:0:2);
end.
```

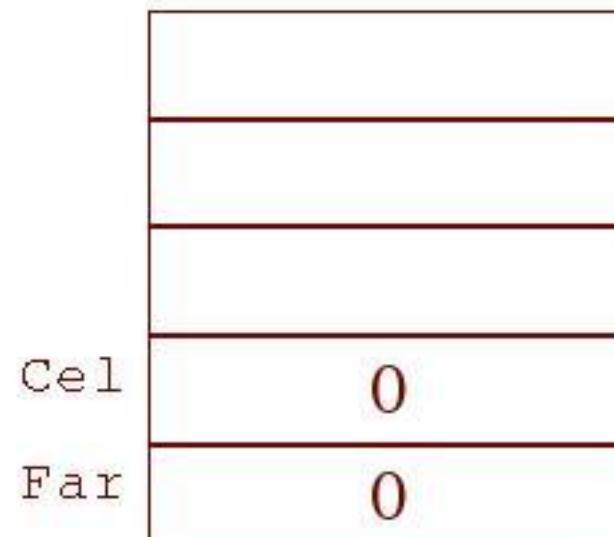
## **A função converte em Java**

```
public class IO extends EasyIn {
    public static float converte(float Fer) {
        return (Fer - 32) * 5 / 9;
    }

    public static void main(String args[]) {
        float Far, Cel;
        System.out.print("Fahrenheit: ");
        Far = getFloat();
        Cel = converte(Far);
        System.out.println("Celsius: " + Cel);
    }
}
```

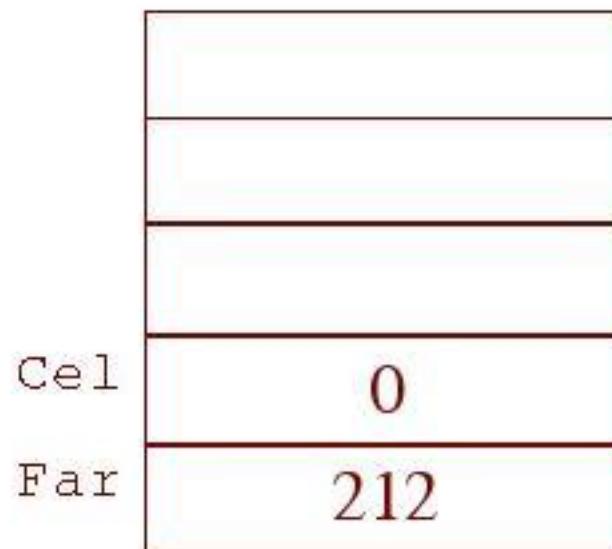
## **A função converte: alocação de memória**

```
var  
    Far, Cel : real;
```



## **A função converte: alocação de memória**

```
readln(Far);
```



## **A função converte: alocação de memória**

```
// chamada da função  
Cel := converte(Far);
```

result:Converte	?
Fer:Converte	212
Cel	0
Far	212

## **A função *converte*: alocação de memória**

```
result := (Fer - 32) * 5 / 9;
```

result:Converte

100

Fer:Converte

212

Cel

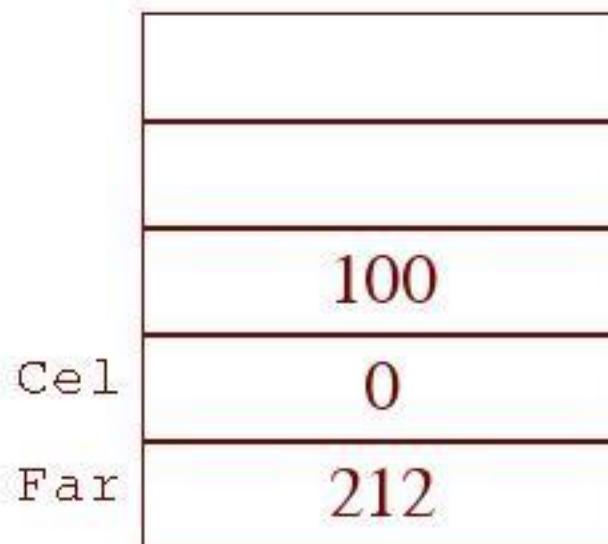
0

Far

212

## **A função converte: alocação de memória**

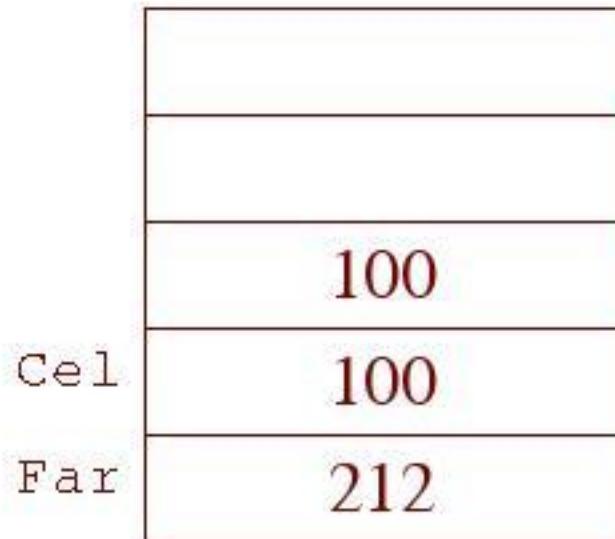
```
// término da função  
    result := (Fer - 32) * 5 / 9;  
end;
```



## **A função converte: alocação de memória**

```
// retorno da função
```

```
Cel := converte(Far);
```



## **Outro exemplo: A função area**

```
função CalculaArea(entradas: l, c)
    inicio
        resultado ← l * c
    fim

programa teste
    inicio
        imprima 'Largura do jardim? '
        leia larg
        imprima 'Comprimento do jardim? '
        leia comp
        imprima 'Area: ', CalculaArea(larg, comp)
    fim
```

## **A função area**

**Saída:**

Largura do jardim? 100

Comprimento do jardim? 50

Area: 5000

## **A função area em Pascal**

```
function CalculaArea(l, c: integer) : integer;
begin
    result := l * c;
end;

var
    larg, comp: integer;
begin
    write('Largura do jardim? ');
    readln(larg);
    write('Comprimento do jardim? ');
    readln(comp);
    writeln('Area: ', CalculaArea(larg, comp));
end.
```

## **A função area em Java**

```
public class IO extends EasyIn {
    public static int CalculaArea(int l, int c)    {
        return l*c;
    }

    public static void main(String args[]) {
        int larg, comp;
        System.out.print("Largura do jardim? ");
        larg = getInt();
        System.out.print("Comprimento do jardim? ");
        comp = getInt();
        System.out.println("Area: " +
                           CalculaArea(larg, comp));
    }
}
```

## Vetores e funções

*Um vetor pode ser passado como parâmetro usando-se apenas o nome do mesmo na chamada à função ou procedimento*

```
inicio
    // inicializa o vetor
    imprima 'Maior: ', achaMaior(vetor, pos)
    imprima 'posicao: ', pos
fim
```

# Vetores e funções

```
inicio
    // inicializa o vetor
    imprima 'Maior: ', achaMaior(vetor, pos)
    imprima 'posicao: ', pos
fim
```

*vetor: vetor de números inteiros*

*achaMaior: função que devolve o maior elemento  
de um vetor e a posição onde ele foi  
encontrado*

*pos: posição do maior elemento do vetor*

**cederj**

## Vetores e funções

```
função achaMaior (entradas: tab
                     saídas: pos)
inicio
    resultado ← tab[0]
    pos ← 0
    para i ← 1 até tamanho(tab) faça
        se tab[i] > resultado então
            resultado ← tab[i]
            pos ← i
        fim se
    fim para
fim
```

# Vetores e funções: Alocação em memória

Uma cópia de cada um dos elementos do vetor é criada na pilha.

	6	resultado:achaMaior
	3	tab[3]:achaMaior
	6	tab[2]:achaMaior
	2	tab[1]:achaMaior
	6	
vetor[3]	3	
vetor[2]	6	
vetor[1]	2	
pos	1	pos:achaMaior

## **Vetores e funções**

*A passagem de um vetor por valor pode ser uma operação lenta, se o vetor tiver muitos elementos!*



## **Vetores e funções: passagem por referência**

```
função achaMaior (saídas: tab
                     saídas: pos)

inicio
    resultado <- tab[0]
    pos <- 0
    para i <- 1 até tamanho(tab) faça
        se tab[i] > resultado então
            resultado <- tab[i]
            pos <- i
        fim se
    fim para
fim
```

# **Vetores e funções: passagem por referência**

## *Alocação em memória*

	6	resultado:achaMaior
	6	
vetor[3]	3	tab[3]:achaMaior
vetor[2]	6	tab[2]:achaMaior
vetor[1]	2	tab[1]:achaMaior
pos	1	pos:achaMaior

## **Vetores e funções: passagem por referência**

*A passagem de vetores por referência pode ser vantajosa, se os vetores forem grandes, e se o tempo de processamento for um dado crítico no programa.*

*Evita-se assim a cópia dos valores dos vetores na pilha.*