

Psedocódigo com Portugol

Engenharia de Telecomunicações 2ª Fase

Professor: Cleber Jorge Amaral



Algoritmos – breve revisão

► Definição

- Sequência ordenada de passos para resolução de um dado problema. Voltado a automação, mas no geral pode ser uma receita de bolo ou a rotina de lançamento de um foguete

► Problema bem definido

- Possuem objetivos, caminhos de solução e expectativa de solução claros. Contra-exemplo: rota da SC401 a BR101, não especifica claramente o objetivo, nem caminhos ou solução esperada.

► Descrição narrativa

- Texto livre bom se organizado em tópicos. Ex.: Trocar um pneu.

► Fluxograma

- Operações são representadas por formas geométricas. Ex.: Cálculo da média.



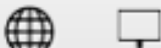





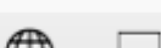
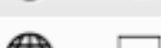
Transformando uma sequencia de passos da linguagem humana para as máquinas

- ▶ Um algoritmo na forma de narrativa escrito em língua portuguesa, por exemplo, não seria compreendido por um computador.
- ▶ A máquina basicamente é capaz de executar operações matemáticas, lógicas e de armazenamento e leitura de dados
- ▶ As imagens, letras e cores deste slide, por exemplo, é o resultado de operações de leitura de memória, cálculos matemáticos e testes lógicos diversos
- ▶ Para realizar esta tradução são utilizadas, portanto, linguagens intermediárias (ou linguagens de programação) onde o homem possa expressar soluções lógicas e que ao mesmo tempo possua certo rigor sintático para que a máquina possa interpretar e transformar em comandos de máquina

Linguagens de programação

- ▶ Há diversas linguagens de programação, cada uma com uma “gramática” própria.
- ▶ As linguagens são equipadas com bibliotecas de funções para realização de certas tarefas, como escrever uma frase na tela ou realizar o cálculo de um seno, por exemplo.
- ▶ O problema das linguagens de programação na aprendizagem de lógica computacional é que normalmente a estrutura estão escritas em inglês, muitas vezes a documentação da linguagem é muito ampla se tornando confusa pra o básico e a sintaxe muitas vezes pouco inteligível

Algumas Linguagens (Ranking IEEE 2016)

Language Rank	Types	Spectrum Ranking
1. C		100.0
2. Java		98.1
3. Python		98.0
4. C++		95.9
5. R		87.9
6. C#		86.7
7. PHP		82.8
8. JavaScript		82.2
9. Ruby		74.5
10. Go		71.9

Pseudocódigos

- ▶ Definição
 - É a escrita por meio de uma regra pré-definida de um algoritmo.
- ▶ Não é uma linguagem de programação, é uma forma de expressar algoritmos para uso didático onde o mais importante é expressão lógica de uma solução e não exatamente a elaboração de um programa para determinada aplicação
- ▶ Há diversas formas de se escrever códigos em pseudocódigo, há portanto “sintaxes” mas via de regra são formas de escrita mais naturais ao ser humano

Para que serve?

- ▶ Fins didáticos:
 - Quando deseja-se escrever um algoritmo de uma forma genérica sobre um algoritmo, sem se referir a nenhuma linguagem de programação formalmente
- ▶ Expressão genérica:
 - Quando deseja-se representar ideias de encadeamento lógico sem preocupação com o rigor da sintaxe de uma linguagem compilável (em um brainstorming, por exemplo)
- ▶ Ferramenta que utilizaremos:
 - Portugol por Antônio Medeiros. Atenção: Há diversas sintaxes para o pseudocódigo chamado Portugol, a sintaxe que utilizaremos aqui é a descrita no trabalho TCC disponibilizado no link a seguir:

<https://vinyanalista.github.io/portugol/>

Estrutura e sintaxe

- ▶ Possui “palavras reservadas” que só podem ser utilizadas para sua própria finalidade.
- ▶ Estrutura básica (Sintaxe do Portugol – de ANTONIO MEDEIROS):

ALGORITMO

//declaracoes de variáveis

//bloco_de_comandos (instruções)

FIM_ALGORITMO.

- ▶ Toda variável deve ter nome e tipo
- ▶ Os nomes de variáveis possuem restrições na elaboração
 - Não coincidir com palavras reservadas
 - Não iniciar com número
 - Não possuir espaço ou caracteres especiais (ç, á, &,...)
 - Exemplos de nomes válidos: i, n1, nomeVariavel, usuario, Temperatura,...

Tipos de variáveis e operadores

► Tipos de variáveis do Portugol

- numérico: usado para variáveis que devem armazenar números, como -23, -23.45, 0, 98, 346.89;
- lógico: usado para variáveis que devem assumir apenas os valores VERDADEIRO ou FALSO;
- literal: usado para armazenar um ou mais caracteres (letras maiúsculas, minúsculas, números e caracteres especiais) em sequência.

► Operadores aritméticos são:

- (subtrai ou inverte o sinal)
- + (soma ou mantém o sinal)
- * (multiplicação)
- / (divisão)

► Operadores de atribuição:

- <- (atribui um valor a uma variável)

Operadores relacionais e lógicos

- ▶ Operadores relacionais

 - = (igualdade)

 - <> (diferença)

 - < Menor que

 - <= Menor ou igual que

 - > Maior que

 - >= Maior ou igual que

- ▶ Operadores lógicos:

 - OU (lógica OU)

 - E (lógica E)

 - NAO (lógica negação)

Regras de precedência do Portugol

1) + (soma) e - (subtração)

Ordem: Esquerda para direita

2) * (multiplicação) e / (divisão)

Ordem: Esquerda para direita

3) + (positivo) e - (negativo)

Ordem: Direita para esquerda

4) “()” (Parênteses)

Ordem: “Dentro para fora”

Exemplos: declare i,j,k,l numerico

`i <- 1 + 2 * 3` // Resultado: $i \leftarrow 7$

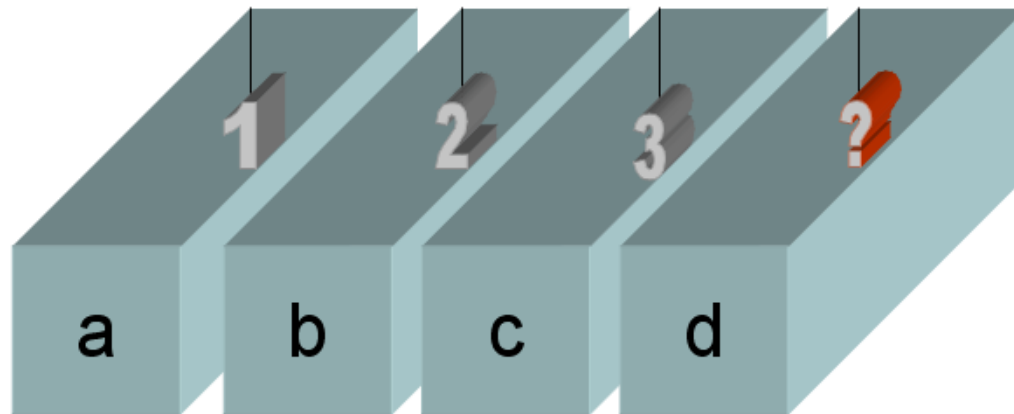
`j <- 2 * 3 + 5 / 2` // Resultado: $j \leftarrow 8.5$

`k <- -3 * 5 + 8 / -4` // Resultado: $k \leftarrow -17$

`l <- -3 * (5 + 8) / -4` // Resultado: $l \leftarrow 9.75$

Entendendo variáveis

- Variável é um espaço de memória que permite armazenar, ler e modificar seu valor desde a compilação do programa até sua execução do programa. Ou seja, pode-se carregar valores iniciais e também manipulá-los durante a execução.



Constantes

- ▶ Constante é um espaço de memória que permite armazenar e ler seu valor que deve ser definido em código. Durante a execução este valor está protegido pelo compilador, não é possível modificá-lo, em suma é “somente leitura” / “read only”.
- ▶ Exemplos:
 - Declaração em código de uma frase a ser exibida em tela: **escreva** “Olá mundo”
 - A maioria das linguagens permite também declarar constantes que podem auxiliar no desenvolvimento e compreensão do código (exemplo $PI = 3.1415$), mas não é suportado pelo português

Sub-rotinas pré-definidas

- ▶ Portugal já possui uma série de sub-rotinas pré-definidas, que vem embarcadas, podemos chamar sem a necessidade de nenhuma parametrização adicional, as mais importantes são de saída e entrada de dados

//concatena uma constante com a variável x

escreva "Conteúdo de x = ", x

//Espera o usuário digitar um valor e coloca em y

leia y

- ▶ Há uma série de funções pré-definidas para cálculos matemáticos entre elas de seno, arredondamento de um número, potencia, raiz quadrada, etc. (Ver MEDEIROS, p58)
- ▶ Há ainda um biblioteca de sub-rotinas de uso geral como limpar tela, obter data do computador, entre outras (Ver MEDEIROS, p59)

Caso CalculaMédia

algoritmo

declare media, num1, num2 **numerico**

//Execução de instruções

escreva "Digite o primeiro número:"

leia num1

escreva "Digite o segundo número:"

leia num2

media <- (num1 + num2) / 2

escreva "A média aritmética é: ",media

fim_algoritmo.

Início do algoritmo

Declaração de variáveis

comentário

Instruções

Fim do algoritmo

Estrutura condicional

- ▶ Permite a montagem de condicionais que podem gerar diferentes trajetórias de operação do programa.
- ▶ A estrutura clássica é o “Se... então” mas podemos ter arranjos complexos de múltiplas condicionais e valores possíveis.
- ▶ Operadores lógicos podem ser utilizados nas expressões para combinar condições: E, OU e NÃO
- ▶ Operadores relacionais também são aceitos: = (igual), <> (diferente), < (menor que), <= (menor ou igual que), > (maior que) e >= (maior ou igual que)

Estrutura condicional – simples e composta

- ▶ Bloco condicional simples com “Se... Então”

...

se idade >= 65 **entao**

 escreva "Considerado idoso"

...

- ▶ Bloco condicional “Se... então... senão...”

...

se idade >= 18 **entao**

 escreva "Considerado adulto"

senao

 escreva "Considerado de menor"

...

Estrutura condicional encadeada

- Bloco condicional “Se... então... senão Se... senão”

```
...  
se idade >= 65 entao  
    escreva "Considerado idoso"  
senao se idade >= 18 entao  
    escreva "Considerado adulto"  
senao  
    escreva "Considerado de menor"  
...
```

Blocos de instruções (início / fim)

...

se idade \geq 16 **entao**

inicio

IMC <- peso / **potencia** (altura,2)

se IMC < 17 **entao**

escreva "Muito abaixo do peso"

senao se IMC < 25 **entao**

escreva "Peso normal"

senao se IMC < 30 **entao**

escreva "Acima do peso"

senao se IMC < 35 **entao**

escreva "Obesidade I"

senao se IMC < 40 **entao**

escreva "Obesidade II (severa)"

fim

...

Caso VaiChover

algoritmo

declare VaiChover **logico**

//Execução de instruções

escreva "Pela previsão vai chover hoje? VERDADEIRO ou FALSO?"

leia VaiChover

se VaiChover **entao**

escreva "Leve o guarda-chuva"

fim_algoritmo.

Caso VaiChoverFrio

algoritmo

declare VaiChover, VaiFazerFrio **logico**

escreva "Pela previsão vai chover hoje? VERDADEIRO ou FALSO?"

leia VaiChover

escreva "Vai fazer frio? VERDADEIRO ou FALSO?"

leia VaiFazerFrio

se VaiChover **e** VaiFazerFrio **entao**

escreva "Leve o guarda-chuva e o casaco!"

senao se VaiChover **entao**

escreva "Leve o guarda-chuva!"

senao se VaiFazerFrio **entao**

escreva "Leve o casaco!"

senao

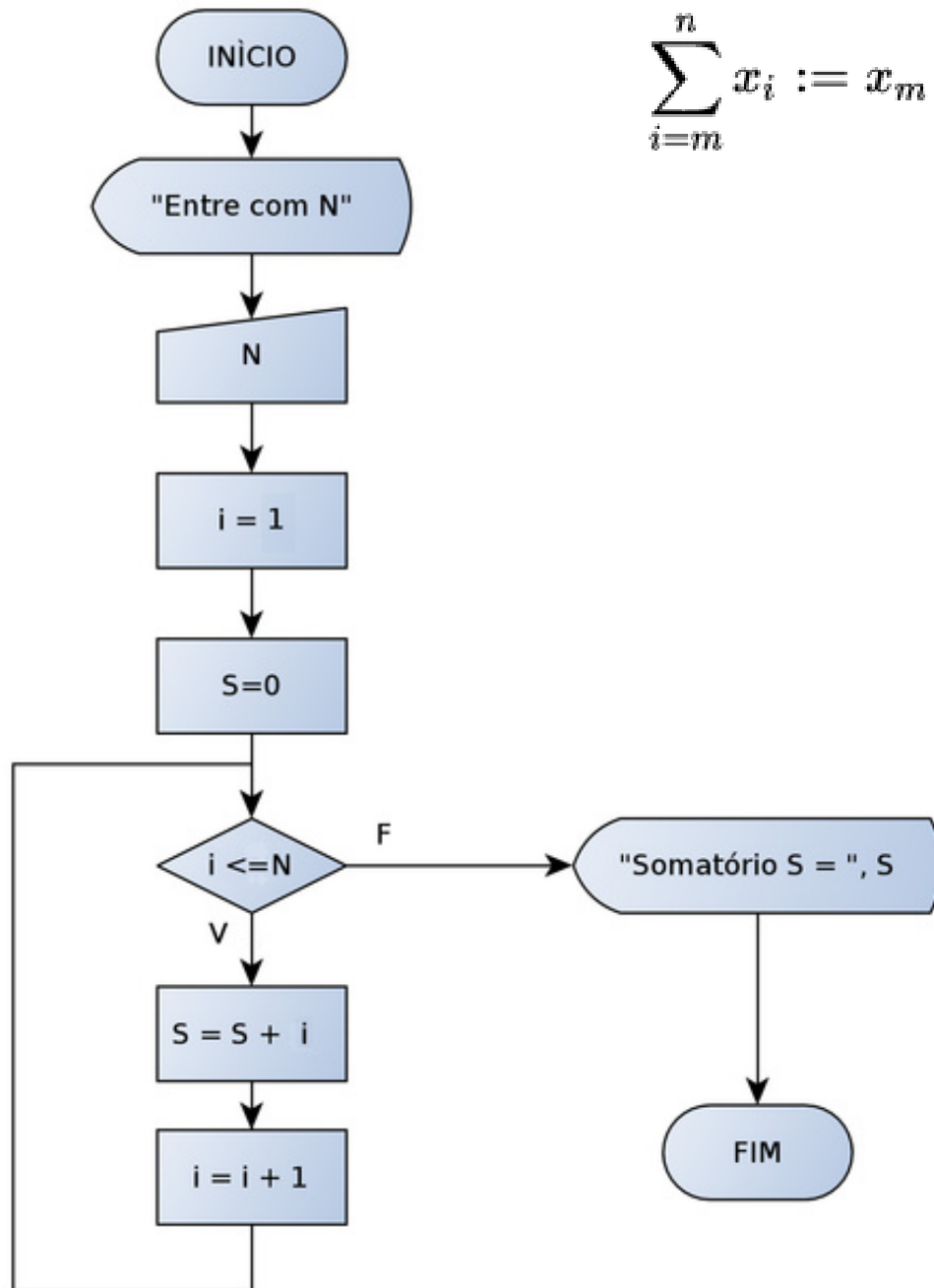
escreva "Aproveite este belo dia!"

fim_algoritmo.

Estruturas de repetição

$$\sum_{i=m}^n x_i := x_m + x_{m+1} + \cdots + x_n$$

Faça algo N vezes ou
faça algo enquanto
determinada condição
for verdadeira



Estruturas de repetição no portugol

- ▶ As estruturas de repetição permitem o que talvez seja a aplicação mais importante dos sistemas computacionais: repetir rotinas executando com mesmos critérios e quantas vezes for necessário
- ▶ Para realização desta tarefa normalmente se realiza uma contagem e se finaliza o processo ao atingir a quantidade de repetições desejada ou se repete um determinado procedimento enquanto uma certa condição está ocorrendo
- ▶ No portugol há três estruturas de repetição
 - ENQUANTO
 - REPITA
 - PARA

ENQUANTO

- ▶ Utilizado especialmente quando não se conhece a quantidade de vezes se deseja repetir uma instrução, podendo até nem vir a ser executado
- ▶ Estrutura:

ENQUANTO condicao **FACA**

comando

- **ENQUANTO**: palavra-chave de inicio do bloco de repetição, seguida de uma condição que quando verdadeira será executado o comando após **FACA**
- **FACA**: demarca o comando ou bloco de comandos que serão executados enquanto a condição se mantiver verdadeira

Caso Somatorio

algoritmo

declare N,i,S **numerico**

escreva "Entre com N"

leia N

i <- 1

S <- 0

enquanto i <= N **faca**

inicio

S <- S + i

i <- i + 1

fim

escreva "Somatório S = ",S

fim_algoritmo

Lembre-se
o programa
Sempre
Executa de
cima para
baixo

**Entendendo o que o
Portugol faz num
laço enquanto:**

1: Testa se i <= N (se sim vai p/ 2, se não vai para 3)

2: executa o comando neste caso
tem um bloco inicio/fim

3: Prossegue com a execução do programa

Caso AcerteCaractere

algoritmo

declare caractere **literal**

escreva "Acerte o caractere para sair do programa:"

leia caractere

enquanto caractere <> "q" **faca**

inicio

escreva "Caractere incorreto!"

leia caractere

fim

escreva "Saindo..."

fim_algoritmo.

Caso DigiteNotaValida

algoritmo

declare nota **numerico**

escreva "Digite uma nota de 0 a 10:"

leia nota

enquanto ((nota < 0) ou (nota > 10)) **faca**

inicio

escreva "Nota inválida! Digite uma nota de 0 a 10:"

leia nota

fim

escreva "Nota válida digitada!"

fim_algoritmo.

REPITA

- ▶ Semelhante a estrutura ENQUANTO esta é utilizada especialmente quando não se conhece a quantidade de vezes se deseja repetir uma instrução, porém aqui no REPITA ao menos uma vez será executada
- ▶ Estrutura:

REPITA

comando

ATE condicao

- **REPITA**: palavra-chave de inicio do bloco de repetição, da lista de comandos (não precisa declarar o bloco inicio/fim)
- **ATE**: expressão que descreve a condição que deve ocorrer para finalizar o bloco de repetição

Caso AcerteCaractereComRepita

algoritmo

declare caractere **literal**

repita

escreva "Acerte o caractere para sair:"

leia caractere

ate caractere = "q"

escreva "Saindo..."

fim_algoritmo.

Entendendo o que o Portugol faz num laço repita

1: executa o bloco de instruções

2: Testa se caractere = "q" (sim vai p/ 1, não vai p/ 3)

3: Prossegue com a execução do programa

Caso UmADezComRepita

algoritmo

declare numero **numerico**

escreva "Escreva os números de 1 a 10:"

numero <- 1

repita

escreva "Numero: ",numero

numero <- numero + 1

ate numero > 10

fim_algoritmo.

PARA

- ▶ Serve quando se conhece a quantidade de vezes se deseja repetir algo
- ▶ Estrutura:

PARA indice <- valor_inicial **ATE** valor_final **FACA** [**PASSO** n]
comando

- **PARA**: palavra-chave de inicio do bloco de repetição, seguida de uma operação inicial que será executada apenas no início do bloco para carga inicial da variável de controle
- **ATE**: determina o valor que a variável deve alcançar para encerrar a repetição, quando esta condição for verdadeira será encerrada a operação deste PARA
- **FACA**: demarca o comando ou bloco de comandos que serão executados até que a variável de controle alcance o valor em ATE (esta será sua última execução)
- **PASSO**: O incremento padrão é 1, aqui pode-se determinar um outro incremento ou decremento da variável de controle.

Caso UmADezcomPARA

Entendendo o que o
Portugol faz num
laço para

algoritmo

declare i numerico

para i <- 1 ate 10 faca

inicio

escreva "O valor de i é: ",i

fim

escreva "Execução finalizada"

fim_algoritmo.

1: variável "i" recebe valor 1

2: Testa se $i < 10$ (sim vai p/ 3, não vai p/ 4)

3: Executa comando (neste caso bloco inicio/fim), se $i+1 < 10$ incrementa i, vai para 2

4: Prossegue com a execução do programa

Caso EscreveImpares

algoritmo

declare

i numerico

numero numerico

escreva "Informe um inteiro positivo p/ verificar os ímpares menores ou igual a este"

leia numero

para i <- 1 **ate** numero **faca passo** 1

inicio

se resto(i, 2) <> 0 **entao**

escreva i, " é ímpar!"

fim

fim_algoritmo.

Caso Fatorial

algoritmo

declare fat,i,numero **numerico**

leia numero

inicio

fat <- 1

para i <- 1 **ate** numero **faca**

inicio

fat <- fat * i

fim

escreva "Fatorial: ",fat

fim

fim_algoritmo

Laço principal do programa

- ▶ É comum que os programas fiquem rodando por tempo indeterminado, aguardando comandos do usuário ou realizando tarefas de leitura de dados, etc.
- ▶ Isso se dá utilizando uma estrutura em laço que contém a operação principal do software.
- ▶ Um exemplo de laço principal pode ser montado utilizando as estruturas repita e enquanto dos exemplos acerte o caractere, ou seja, o programa executará algo até que o usuário digite um dado caractere que encerra a aplicação.
- ▶ Esta estrutura também pode ser chamada de loop infinito, ainda que não seja exatamente infinito já que há formas de encerrá-la.

Vetores

- ▶ Um vetor é uma variável composta unifilar que permite o acesso a suas posições de memória através de um índice
- ▶ Os dados de um vetor serão todos do mesmo tipo conforme declaração
- ▶ Em memória um vetor é armazenado sequencialmente
- ▶ A declaração deve ser feita na sessão DECLARE juntamente com outras variáveis

DECLARE nome[tamanho] tipo

- NOME: identificador do vetor, um nome qualquer seguindo as mesmas restrições de um nome de uma variável comum
- TAMANHO: é a quantidade de variáveis que serão instanciadas em memória
- TIPO: o tipo deste conjunto de variáveis, sendo aceitos os mesmo para variáveis comum

Exemplo de vetor

DECLARE x[5] NUMERICO

- ▶ Indica que 5 posições de memória para armazenamento de variáveis tipo NUMERICO foram reservadas em sequencia e inicializadas com zero
- ▶ Para acessar cada um espaço de memória utiliza-se o indice correspondente, por exemplo:
 - `x[1] <- 45` //Carrega na primeira posição do vetor x o número 45
 - `x[5] <- 128` //Carrega na quinta (e última) posição de x o número 128

Caso ImprimeVetorFixo

algoritmo

declare x[5] **numerico**

x[1] <- 45 //Carrega na primeira posição do vetor x o número 45

x[5] <- 128 //Carrega na quinta (e última) posição de x o número 128

escreva x[1], "-",x[2], "-",x[3], "-",x[4], "-",x[5]

fim_algoritmo

Caso Obtem5Numeros

algoritmo

declare

i, x[5] numerico

para i <- 1 ate 5 faca

inicio

escreva "Digite o ", i, "º número"

leia x[i]

fim

para i <- 1 ate 5 faca

inicio

escreva "O ", i, "º número digitado foi ", x[i]

fim

fim_algoritmo

Matrizes

- ▶ Uma matriz é uma variável composta multidimensional que permite o acesso a suas posições de memória através de combinações de índices
- ▶ Os dados de um vetor serão todos do mesmo tipo conforme declaração
- ▶ Em memória um vetor é armazenado sequencialmente
- ▶ A declaração deve ser feita na sessão DECLARE juntamente com outras variáveis

DECLARE nome[dimensao_1, dimensao_2, ..., dimensao_n] tipo

- NOME: identificador do vetor, um nome qualquer seguindo as mesmas restrições de um nome de uma variável comum
- dimensao_1, dimensao_2, ..., dimensao_n: é a quantidade de variáveis que serão instanciadas em memória em cada dimensão da matriz
- TIPO: o tipo deste conjunto de variáveis, sendo aceitos os mesmo para variáveis comum

Exemplo de matriz

DECLARE x[3,5] NUMERICO

- ▶ Indica que 15 (3 vezes 5) posições de memória para armazenamento de variáveis tipo NUMERICO foram reservadas em sequencia na memória e inicializadas com zero
- ▶ Para acessar cada um espaço de memória utilizam-se os índices correspondentes, por exemplo:
 - `x[3,1] <- 4` //Carrega a célula posicionada na linha 3, coluna 1 da matriz “x” com o número 4
 - `X[1,5] <- 8` //Carrega a célula posicionada na linha 1, coluna 5 da matriz “x” com o número 8

Caso ImprimeMatrizFixo

algoritmo

declare x[3,5] **numerico**

x[3,1] <- 4 //Carrega a célula da linha 3, coluna 1 com o número 4

x[1,5] <- 8 //Carrega a célula da linha 1, coluna 5 com o número 8

escreva "Linha 1: ",x[1,1], "-",x[1,2], "-",x[1,3], "-",x[1,4], "-",x[1,5]

escreva "Linha 2: ",x[2,1], "-",x[2,2], "-",x[2,3], "-",x[2,4], "-",x[2,5]

escreva "Linha 3: ",x[3,1], "-",x[3,2], "-",x[3,3], "-",x[3,4], "-",x[3,5]

fim_algoritmo

Caso ObtemNumerosMatriz

algoritmo

declare i, j, x[3,5] **numerico**

para i <- 1 **ate** 3 **faca**

inicio

para j <- 1 **ate** 5 **faca**

inicio

escreva "Digite o número da linha ", i, " e coluna ", j

leia x[i,j]

fim

fim

para i <- 1 **ate** 3 **faca**

para j <- 1 **ate** 5 **faca**

inicio

escreva "O número da linha ", i, " e coluna ", j, " é: ", x[i,j]

fim

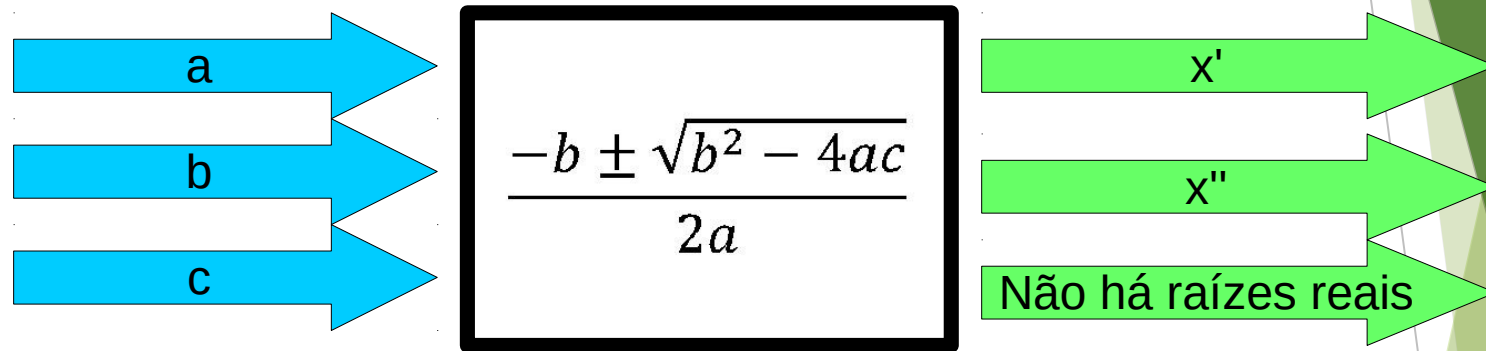
fim_algoritmo

Sub-rotinas ou sub-programas

- ▶ São blocos de instruções que realizam algum processamento bem definido
- ▶ É interessante para “encapsular” a solução de dados problemas em estruturas mais enxutas e bem testadas
- ▶ Torna o programa mais simples de entender pois códigos de atividades específicos ficam descritos em áreas específicas e o problema como um todo fica dividido em implementações menores
- ▶ Em algumas linguagens também podem ser chamados de funções, métodos ou procedimentos
- ▶ Dentro de uma sub-rotina pode haver declarações de variáveis que serão apenas de utilização local (dentro da sub-rotina)
 - não podendo ser acessada de outras sub-rotinas ou do programa principal.
 - Estas variáveis são destruídas no final da execução da sub-rotina
- ▶ A sub-rotina pode receber parâmetros (como entradas) e retornar valores (como saídas).
 - Um parâmetro aqui no português sempre é passado por sua referência então na prática pode funcionar além de entrada como uma saída

Sub-rotinas vs função matemática

- O conceito pode ser compreendido como de uma função matemática
 - Imaginemos uma função que calcula as raízes de uma equação de segundo grau ($ax^2 + bx + c = 0$)



- Espera-se entrar com a, b e c, e obter x' e x'', ou uma indicação de que não há raízes reais
- Uma função matemática como potência, fatorial, raízes de equações de segundo grau, derivada e integral são operações bem definidas que esperam entradas e retornam saídas
- Uma sub-rotina pode ou não receber entradas e pode ou não retornar saídas

Caso ImprimeSaudacao

algoritmo

declare tecla **literal**

enquanto tecla <> "q" **faca**

inicio

imprime_saudacao()

//Chamadas para realização de operações segmentadas

leia tecla

fim

fim_algoritmo

sub-rotina imprime_saudacao()

escreva "Bem vindo ao programa"

escreva "Digite ? para ?"

escreva "Digite ?? para ??"

escreva "Digite q para sair"

fim_sub_rotina imprime_saudacao

A idéia de um programa como este é ter um menu principal e realizar operações diversas. Aqui está sendo proposto colocar em uma sub-rotina as funções de escrita do menu

Programa invoca a sub-rotina `imprime_saudacao()`

sub-rotina `imprime_saudacao()`
Não recebe nenhum parâmetro
Não retorna nada, apenas
Imprime mensagens em tela

Caso ParidadePar

algoritmo

declare num **numerico**

escreva "Entre com um inteiro qualquer"

leia num

paridade_par(num)

fim_algoritmo

sub-rotina paridade_par(n NUMERICO)

declare x **numerico**

x <- resto (n, 2)

se (x = 1) **entao**

escreva "O número é ímpar"

senao

escreva "O número é par"

fim_sub_rotina paridade_par

A paridade é um valor binário (verdadeiro ou falso). Pode-se ter paridade par ou ímpar. Se o número é par, sua paridade par é verdadeira e sua paridade ímpar é falsa.

Programa invoca a sub-rotina paridade_par(num)
O parâmetro num é uma variável global

Sub-rotina paridade_par
Recebe o parâmetro n
Não retorna nada, apenas
Imprime mensagens em tela.
Para realizar esta lógica a
Sub-rotina utiliza x uma
Variável local

Caso ParidadeParComRetorno

algoritmo

declare num **numerico**

escreva "Entre com um inteiro qualquer"

leia num

se (paridade_par(num)) **entao**

escreva "O número é ímpar"

senao

escreva "O número é par"

fim_algoritmo

sub-rotina paridade_par(n NUMERICO)

declare x **numerico**

x <- **resto** (n, 2)

se (x = 1) **entao**

retorne verdadeiro

senao

retorne falso

fim_sub_rotina paridade_par

Programa invoca a sub-rotina paridade_par(num)
O condicional SE está testando o retorno da sub-rotina

O parâmetro num é uma variável global

Sub-rotina paridade_par
Recebe o parâmetro n
Retorna VERDADEIRO ou FALSO
Para realizar esta lógica a Sub-rotina utiliza x uma Variável local

Registros

- ▶ Registros são estruturas de dados que agregam informações na forma de campos que podem se heterogêneos (combinar múltiplos de dados)
 - Vetores ou matrizes também são variáveis compostas porém homogêneas, ou seja, todos os campos do conjunto é do mesmo tipo (numérico ou literal ou lógico)
- ▶ Aplica-se este recurso para organizar dados na forma de tabelas (planilhas)
- ▶ São declarados no mesmo local das variáveis, vetores e matrizes

DECLARE nome **REGISTRO** (nome_do_campo_1 tipo_do_campo_1, nome_do_campo_2 tipo_do_campo_2, ..., nome_do_campo_n tipo_do_campo_n)

- NOME: identificador do registro, um nome qualquer seguindo as mesmas restrições de um nome de uma variável comum
- nome_do_campo_?: um nome qualquer de campo nos mesmos moldes de declaração de variáveis, vetores e matrizes
- tipo_do_campo_?: o tipo deste do campo referido

Exemplos de registros

DECLARE conta **registro** (num, saldo **NUMERICO** cliente **LITERAL**)

- Estrutura de dados chamada conta, possui três campos: Num e saldo do tipo numérico e Nome do tipo literal

- conta.num <- 5
- conta.saldo <- 850.65
- conta.nome <- "CC do Banco do Brasil"

DECLARE conta[3] **registro** (num, saldo **NUMERICO** nome **LITERAL**)

- Um vetor de registros vai se parecer com uma planilha, aqui são os mesmos campos (colunas) porém são 3 registros (linhas)

- conta.num[2] <- 8
- conta.saldo[2] <- 91.74
- conta.nome[2] <- "POP do Santander"

Caso MediaAluno

algoritmo

declare

aluno **registro** (nome **literal** nota1, nota2 **numerico**)

Declaração do registro

escreva "Digite o nome do aluno"

leia aluno.nome

Escrevendo no campo nome

escreva "Digite a nota 1 do aluno"

leia aluno.nota1

Escrevendo no campo nota1

escreva "Digite a nota 1 do aluno"

leia aluno.nota2

Escrevendo no campo nota2

escreva "A média do aluno ",aluno.nome," é ",(aluno.nota1+aluno.nota2)/2

fim_algoritmo

Consultando campos
para realizar
aritmética

Caso MediaAlunos

algoritmo

declare

i numerico

aluno[3] registro (nome **literal** nota1, nota2 **numerico**)

Declaração do registro

para i <- 1 ate 3 faca Para percorrer o vetor de registros

inicio

escreva "Digite o nome do aluno ",i

leia aluno[i].nome

Escrevendo no campo nome[i]

escreva "Digite a nota 1 do aluno ",i

leia aluno[i].nota1

Escrevendo no campo nota1[i]

escreva "Digite a nota 2 do aluno ",i

leia aluno[i].nota2

Escrevendo no campo nota2[i]

fim

para i <- 1 ate 3 faca Para percorrer o vetor de registros

escreva "A média de ",aluno[i].nome," é ",(aluno[i].nota1+aluno[i].nota2)/2

fim_algoritmo

Obrigado pela
atenção e
participação!