Estruturas de dados homogêneas

Vetores Multidimensionais (continuação)

Exercício 30:

Com base no exercício anterior, faça um algoritmo que declare duas matrizes 2x3 de reais, as inicialize e efetue a soma das duas matrizes retornando a matriz resultante na saída padrão com o layout retro aludido.

UNIVASF

Estruturas de dados homogêneas

Vetores Multidimensionais (continuação)

Exercício 31:

Com base no exercício 38, faça as alterações necessárias para que o algoritmo efetue a multiplicação de duas matrizes (2x3 e 3x2, respectivamente) inicializadas, através da entrada padrão, e retorne a matriz resultante da multiplicação, com layout apropriado, na saída padrão.

UNIVASF

Modularização

Com o avanço do estudo sobre algoritmos os problemas a serem solucionado aumentam em complexidade.

Um método de resolução de problemas complexos é visualizá-los como compostos por problemas menores e tratar um a um os sub-problemas identificados.

Podemos, desta forma, construir um algoritmo composto por subalgoritmos denominados módulos.

Esta metodologia é denominada top-down (de cima para baixo), ou melhor, do genérico para o específico.

UNIVASF

Modularização

Desta forma ao nos depararmos com um problema complexo devemos buscar visualizá-lo como um conjunto de problemas mais simples.

Trataremos agora de como construir um algoritmo composto por módulos.

Assim como um algoritmo, em geral, possui um conjunto de entradas, efetua um processamento e gera um conjunto de saídas. Um módulo também funciona da mesma forma.

UNIVASF

Modularização

Assim como foi definida uma estrutura para representação de algoritmos em pseudocódigo, também é especificada uma estrutura para construção de módulos.

Trabalharemos com dois tipos de módulos:

- a função;
- e o procedimento.

Iniciaremos nossas análise pelo conceito de função.

Uma função é um módulo que possui **ou não** um conjunto de entradas, efetua a execução de um conjunto de instruções e **sempre** gera um valor como saída, também denominado retorno.

Modularização

A estrutura de uma função é a seguinte:

funcao <*nome_da_função*> ([<*sequência-de-declarações-de-parâmetros*>]) : <*tipo_de_dado*>

// Seção de Declarações Internas

inicio

// Seção de Comandos

fimfuncao

Obs.: A declaração de uma função deve estar entre o final da declaração de variáveis e a linha contendo **inicio** no algoritmo principal.

Modularização

O *tipo_de_dado* é o tipo do valor que a função vai retornar.

A *sequência-de-declarações-de-parâmetros* é uma lista com a seguinte forma geral:

identificador1: tipo_de_dado; identificador2: tipo_de_dado; ...; identificadorN: tipo_de_dado

Repare que o *tipo_de_dado* deve ser especificado para cada uma das N variáveis definidas como parâmetros independente de mais de uma variáveis ser do mesmo *tipo_de_dado*. É na declaração de parâmetros que informamos quais serão as entradas da função (assim como informamos a saída no *tipo_de_dado* associado à função).

Modularização

Em um módulo, assim como em um algoritmo, podemos declarar variáveis que serão utilizadas nas manipulações efetuadas. Para esta finalidade existe a **Seção de Declarações Internas**.

As regras para construção do *nome_da_função* são as mesmas aplicadas na definição dos identificadores de variáveis.

Por fim, é na **Seção de Comandos** também denominado corpo da função que as entradas são processadas, saídas são geradas e/ou outras operações são feitas.

UNIVASF

Modularização

- Comando retorne

Forma geral:

retorne valor_de_retorno

Quando se executa um comando **retorne** a função é encerrada imediatamente e o valor de retorno é retornado pela função. É importante lembrar que o valor de retorno fornecido tem que ser compatível com o tipo de retorno declarado para a função.

Modularização

Exemplo de função:

```
algoritmo "exemplo1 função"
var     num:inteiro
        funcao quadrado (a: inteiro): inteiro
        inicio
                retorne (a*a)
        fimfuncao
inicio
  escreva ("Entre com um número inteiro: ")
  leia (num)
  num <- quadrado(num)
  escreva ("O seu quadrado vale: ",num)
fimalgoritmo
```

Modularização

Exemplo de função:

```
algoritmo "exemplo2 função"
var     num:inteiro
        funcao quadrado (a: inteiro): inteiro
        inicio
                retorne (a*a)
        fimfuncao
inicio
  escreva ("Entre com um número: ")
  leia (num)
  escreva ("O quadrado de ",num)
  escreva (" é ", quadrado(num))
fimalgoritmo
```