UNISUL – Universidade do Sul de Santa Catarina

APOSTILA DE LÓGICA DE PROGRAMAÇÃO II

Professores: Hélton Ribeiro Nunes (helton.nunes@unisul.br)

Adriana Zanini (adriana.zanini@unisul.br)

1 Índice

1 Índice....2
2 Introdução....3
3 Os 10 mandamentos para aprender programação....4
4 Variáveis Indexadas....5
4.1 Definição de Variáveis Indexadas....5
4.2 Operações Básicas com Variáveis Indexadas....6
4.2.1 Atribuição....6
4.2.2 Leitura....7
4.2.3 Escrita....7
4.3 Mais Considerações Sobre Variáveis Unidimensionais (vetores)....8
4.4 Variáveis Indexadas Bidimensionais....10
4.4.1 Atribuição, Leitura e Escrita.....11
5 Sub-rotinas....12
5.1 Procedimentos....13
5.2 Função....15
5.3 Variáveis Globais e Locais....16
5.4 Passagem de Parâmetros....18
5.5 Passagem de Parâmetros por Valor e por Referencia....21
5.6 Recursividade....23

2 Introdução

Neste material utilizado na disciplina de Programação 2 de Sistemas de Informação e de Ciência da Computação, daremos continuidade aos conteúdos inicializados na disciplina anterior (Programação 1).

A forma de trabalho neste semestre não será alterada em relação ao semestre anterior, tanto formato das aulas como de avaliações, o que é vantajoso para os alunos.

Aconselho ainda a procurarem estudar em horário extraclasse, tanto fazendo os algoritmos existentes aqui, como lendo este material e tentar entender cada vez mais os conteúdos.

Todo o conteúdo aqui encontrado foi desenvolvido usando muitos materiais recolhidos na própria internet como apostilas listas de exercícios de outros professores e materiais colhidos na própria biblioteca.

3 Os 10 mandamentos para aprender programação

Abaixo seguem os nossos já famosos mandamentos, na verdade são conselhos para melhorar o aprendizado nas disciplinas de programação.

1) Assistir todas as aulas (em sala e em laboratório).
2) Perguntar sempre que tiver dúvida durante a explicação de um novo assunto ou resolução de exercício. Nunca leve uma dúvida para casa.
3) Nas aulas de laboratório, procure sentar próximo a alguém que tenha menos dificuldade. O professor nem sempre consegue atender a todos os alunos individualmente durante a aula. Consulte o colega. Mas cuidado, não peça para que ele resolva o exercício para você. Pergunte onde você errou.
4) Procure o professor nos horários de disponibilidade ou via e-mail para tirar dúvidas. Dificilmente o professor conseguirá corrigir todos os exercícios propostos durante as aulas.
5) Organize grupos de estudo para resolução dos exercícios.
6) Procure vir nas monitorias. Os monitores estão preparados para lhe ajudar a resolver os exercícios e revisar algum conceito que não tenha ficado claro.
7) Faça pesquisas na Internet. Apesar de muita informação duvidosa, existem matérias excelentes que podem ser usados como apoio. Dê preferência a apostilas e sites de professores de outras universidades.
8) Procure criar uma rotina de estudos. Por exemplo: as aulas são as terças-feiras. Você estuda durante as aulas. Na quinta-feira, reveja rapidamente a teoria e resolva pelo menos 5 exercícios. No sábado venha à monitoria ou estude com os colegas.
9) Os assuntos são encadeados. O assunto da primeira aula será utilizado na segunda. Os assuntos são acumulativos! Se você ainda tiver dúvidas do assunto anterior quando o professor estiver explicando um novo tópico poderá ter sérios problemas, pois tudo aquilo que foi visto anteriormente, será necessário e utilizado no novo tópico.
10) A única forma de aprender algoritmos é fazendo algoritmos. Não adianta apenas assistir a todas as aulas, sem resolver exercícios, muitos exercícios. Portanto resolva todos os exercícios propostos e complementares, sempre começando pelos mais fáceis. Se tiver tempo procure nos livros da bibliografia indicada exercícios sobre o assunto que está estudando.

4 Variáveis Indexadas

Como já foi estudado, uma variável simples é uma entidade criada para permitir o acesso a uma posição de memória onde se armazena uma informação, de um determinado tipo de dado, pela simples referência a um nome simbólico.

Porém, são muito comuns as situações práticas em que se necessita referenciar um grupo de variáveis do mesmo tipo pelo mesmo nome simbólico. Para satisfazer a esta necessidade foi criado o conceito de variáveis indexadas, que é um conjunto de variáveis do mesmo tipo, referenciáveis pelo mesmo nome e individualizadas entre si através de sua posição dentro desse conjunto.

O termo **indexada** provém da maneira como esta individualização é feita: por meio de **índices**. Uma variável indexada pode ser definida como tendo um ou mais índices. No caso em que um único índice é usado a variável indexada é chamada de **vetor**. Quando uma variável indexada possui dois índices, ela é chamada de **matriz**. Estes termos (vetor e matriz) são adotados em analogia aos conceitos usados na Matemática e na Física. Teoricamente não há qualquer restrição quanto à existência de variáveis indexadas com três ou mais índices, mas na prática sua ocorrência é muito pouco freqüente.

Ao número de índices necessários à localização de um componente (elemento) dentro da variável indexada dá-se o nome de **dimensão**.

4.1 Definição de Variáveis Indexadas

A sintaxe do comando de definição de variáveis indexadas é a seguinte:

Variáveis

VETOR [dim1] **de** <tipo de dado> <nome da variável>

Onde:

<nome_da-variável> é o nome simbólico pelo qual o conjunto (variável indexada) é identificado;

A palavra-reservada **VETOR**, seguida da dimensão da variável entre colchetes (“[“ e “]”), determina que se trata de uma variável indexada;

dim1 é o valor máximo do índice da variável na dimensão;

tipo_de_dado é o tipo de dado correspondente aos elementos da variável indexada.

Exemplos de definição de variáveis indexadas:

Variáveis

VETOR [10] **de inteiro** NUM

VETOR [5] **de real** SALARIOS

No exemplo acima temos as variáveis:

NUM, que é um conjunto de dez componentes inteiros, ou seja, uma variável capaz de armazenar dez números inteiros simultaneamente;

SALARIOS, com capacidade de armazenar um total de cinco números reais;

4.2 Operações Básicas com Variáveis Indexadas

Do mesmo modo que acontece com variáveis simples, também é possível operar com variáveis indexadas. Contudo, não é possível operar diretamente com o conjunto completo, mas com cada um de seus componentes isoladamente. Como já foi dito, o acesso individual a cada componente de um conjunto é realizado pela especificação de sua posição no mesmo por meio de um ou mais índices. No exemplo do item anterior foi definida a variável NUM, capaz de armazenar números inteiros (**VETOR** [10] de inteiro). Para acessar um elemento deste conjunto deve-se fornecer o nome do mesmo (NUM) e o índice da componente desejada do conjunto (um número de 0 a 9). Por exemplo, NUM [2] especifica o terceiro elemento do vetor NUM. Portanto, não é possível operar diretamente sobre conjuntos como um todo, mas apenas sobre seus componentes, um por vez. Por exemplo, para somar dois vetores é necessário somar cada um de seus componentes dois a dois. Da mesma forma, as operações de atribuição, leitura e escrita de conjuntos devem ser feitas elemento a elemento, como será mostrado a seguir. a única operação permitida de ser feita com um vetor inteiro é a passagem de parâmetros.

4.2.1 Atribuição

Até então, a sintaxe da instrução primitiva de atribuição foi definida como:

<nome da variável> ← <expressão>

No caso de variáveis indexadas, além do nome da variável deve-se necessariamente fornecer também o(s) índice(s) da componente do conjunto onde será armazenado o resultado da avaliação da expressão.

Naturalmente, também é possível usar conjuntos no meio de expressões, como se faz com variáveis simples, desde que seja especificado o elemento do conjunto que servirá como operando.

Exemplos:

NÚMEROS [2] ← 10

SALÁRIOS [3] ← 100.00

4.2.2 Leitura

A leitura de um conjunto é feita passo a passo, um componente por vez, usando a mesma sintaxe da instrução primitiva de entrada de dados (Leia nome_de_variável). Mais uma vez, além do nome do conjunto, deve ser explicitada a posição do componente lido.

Algoritmo Exemplo1
Variáveis
    VETOR [10] **de Inteiro** NUMEROS
    Inteiro I
Inicio
    Para I de 0 **Até** 9 **Faça**
        Leia NÚMEROS [I]
    fimpara
Fim.

- Algoritmo para leitura de um vetor de números inteiros de 10 posições.

Uma observação importante a ser feita é o uso da construção PARA_FAÇA a fim de efetuar a operação de leitura dez vezes, em cada uma delas lendo um determinado componente do vetor. De fato, o uso desta construção é muito comum quando se opera com conjuntos, devido à necessidade de se realizar uma mesma operação com as diversas componentes dos mesmos. Na verdade, são raras as situações em que se deseja operar isoladamente com uma única componente do conjunto.

4.2.3 Escrita

A escrita de um conjunto obedece à mesma sintaxe da instrução primitiva de saída de dados (Escreva nome_de_variável). Mais uma vez, convém lembrar que, além do nome do conjunto, deve-se também especificar por meio de seu(s) índice(s) qual(is) a(s) componente(s) do vetor a ser(em) escrita(s).

Algoritmo Exemplo2

```
Variáveis
    VETOR [10] de Inteiro NÚMEROS
    Inteiro I
Início
    Para I de 0 Até 9 Faça
        Leia NUMEROS [I]
        Escreva NÚMEROS [I]
    fimpara
Fim.
```

- Algoritmo para escrita de um vetor de números inteiros de 10 posições.

Um exemplo mais interessante é o que vem a seguir, onde um conjunto de dez números é lido e guardado no vetor números. Paralelamente, a soma destes números é calculada e mantida na variável SOMA, que posteriormente é escrita.

```
Algoritmo Exemplo3
Variáveis
    VETOR [10] De Real NUMEROS
    Real SOMA
    Inteiro I
Início
    SOMA ← 0.0
    Para I de 0 Até 9 Faça
        Leia NUMEROS [I]
        SOMA ← SOMA + NUMEROS [I]
    fimpara
    Escreva "Soma =", SOMA
Fim.
```

- Algoritmo para somar dez números reais.

4.3 Mais Considerações Sobre Variáveis Unidimensionais (vetores)

Existem muitos problemas que se tornam difíceis de serem resolvidos através dos modelos de variáveis que tínhamos visto antes das variáveis unidimensionais (vetores) e variáveis bidimensionais (matrizes). Suponha a situação em que tenhamos que fornecer ao usuário uma lista dos 100 funcionários de uma

determinada empresa ordenados alfabeticamente. Uma solução é demonstrada no algoritmo abaixo:

Algoritmo pessoas;
Variáveis
    Literal Nome1, nome2,..., Nome100
Início
    Leia nome1
    Leia nome2
    :
    :
    Leia nome100
Fim.

Para o exemplo acima, tivemos que definir 100 variáveis e ler uma a uma, causando um desperdício de tempo para os programadores e tornando o programa pouco flexível. Se a empresa tivesse 10.000 funcionários, teríamos que definir e ler 10.000 variáveis.

Uma variável unidimensional, também conhecida como vetor, é um tipo especial de variável, onde alocamos um número determinado de posições de memória para uma única variável, e acessamos estas posições através de índices, que é um valor numérico inteiro.

Vejamos o exemplo abaixo, utilizando um vetor ao invés de 100 variáveis:

Algoritmo pessoas
Variáveis
    VETOR [100] **De Literal** Nome
    Inteiro Cont
Início
    Para cont **de** 0 **Até** 9 **Faça**
        Leia nome [cont]
        :
        :
    fimpara

Fim.

4.4 Variáveis Indexadas Bidimensionais

Também conhecida como “Matriz”. Uma variável bidimensional, como o próprio nome já indica, possui duas dimensões sendo que é possível definir variáveis com qualquer tipo de dados válidos.

Definição

Algoritmo Define_Matriz

Variáveis

Matriz [Fim1][Fim2] **De** <Tipo> <Nome>

Início

<Comandos>

Fim.

Observações:

O valor Fim1 corresponde ao índice final da primeira dimensão, enquanto que o valor e Fim2 corresponde ao índice final da segunda dimensão, ou como normalmente iremos tratar, tais valores correspondem as linhas e colunas da matriz;

Uma variável indexada deve ser de apenas de um tipo de dado.

Exemplo:

Definir uma matriz de inteiros, com 10 linhas e 5 colunas.

Algoritmo exemplo

Variáveis

Matriz [10][5] **De Inteiro** M

Início

<Comandos>

Fim.

No exemplo anterior, após a definição da variável, a memória estará como mostrada na tabela abaixo.

	0	1	2	3	4
0	M [0][0]				

1					M [1][4]
2		M [2][1]			
3					
4					
5			M [5][2]		
6		M [6][1]			
7				M [7][3]	
8					
9					

- Representação de uma matriz

Os valores numéricos apresentados acima correspondem aos índices da variável.

4.4.1 Atribuição, Leitura e Escrita.

<Nome>[<linha>][<coluna>] . valor

Leia <nome>[<linha>][<coluna>]

Escreva <nome>[<linha>][<coluna>]

Exemplo:

```
Algoritmo TesteMatriz
Variáveis
    Matriz [5][5] De Literal Nomes
    Inteiro I, J
Início
    Para I de 0 Até 4 Faça
        Para J de 0 Até 4 Faça
            Leia Nomes[I][J]
            Escreva "Nome ",I, J, " : ",Nomes[I][J]
        fimpara
    fimpara
Fim.
```

5 Sub-rotinas

Um matemático uma vez disse que um grande problema se resolve dividindo-o em pequenas partes e resolvendo tais partes em separado. Estes dizeres servem também para a construção de programas. Os profissionais de informática quando necessitam construir um grande sistema, o fazem, dividindo tal programa em partes, sendo então desenvolvido cada uma em separado, onde mais tarde elas serão acopladas para formar o sistema. Estas partes são conhecidas por vários nomes. Nós adotaremos uma destas nomenclaturas: sub-rotinas.

Podemos dar um conceito simples de subrotina dizendo ser um pedaço de código computacional que executa uma Função bem definida, sendo que esta subrotina pode ser utilizada várias vezes no algoritmo.

A utilização de subrotinas é uma técnica de programação que consiste basicamente na divisão de um algoritmo complexo em diversos módulos simples com funções bem definidas gerenciadas pelo modulo principal.

Essa divisão deve-se à existência de algoritmos muito complexos, que:

- Dificultam a compreensão do problema por parte do autor e, principalmente, de outros programadores;
- Tornam o algoritmo pouco confiável;
- Facilitam a ocorrência de erros;
- Incentivam a redundância de códigos;
- Dificultam a manutenção do sistema.

Já trabalhando com subrotinas:

- O algoritmo torna-se de fácil compreensão;
- Aumenta-se a sua flexibilidade;
- Diminuem-se as ocorrências de erros;
- Barateia-se e facilita-se a manutenção do sistema.
- Para permitir o reaproveitamento de código já construído(por você ou por outros programadores);
- Para evitar que um trecho de código que seja repetido várias vezes dentro de um mesmo programa;

- Para permitir a alteração de um trecho de código de uma forma mais rápida. Com o uso de uma função é preciso alterar apenas dentro da função que se deseja;
- Para que os blocos do programa não fiquem grandes demais e, por conseqüência, mais difíceis de entender;
- Para facilitar a leitura do programa-fonte de uma forma mais fácil;
- Para separar o programa em partes (blocos) que possam ser logicamente compreendidos de forma isolada.

Alem disso, aumenta a produtividade do algoritmo, uma vez que os módulos de uso geral podem ser agrupados em bibliotecas para serem reaproveitados no futuro, sem a necessidade de um novo desenvolvimento e de uma nova bateria de testes para a eliminação de erros, pois os mesmos já foram eliminados quando da criação inicial dos módulos.

Quando criamos sub-rotinas devemos buscar ter objetivos:

- Reusabilidade: Devemos procurar criar sub-rotinas sempre que temos algum código que poderá ser utilizado mais de uma vez, evitando a redundância de rotinas.
- Facilidade de detecção de erros: Quando estamos trabalhando no desenvolvimento de um software que esta muito grande, cada vez fica mais difícil encontrar o local correto onde ocorrem erros. Se, por exemplo, para a realização de um determinado calculo criarmos uma sub-rotina e surgir erros na execução do programa, saberemos com mais facilidade onde procurar o problema.
- Organização: Com a criação de sub-rotinas o código torna-se mais limpo e fácil de ser entendido.

Nesta disciplina iremos tratar de dois tipos de Subrotinas: PROCEDIMENTO e FUNÇÃO.

Podemos adiantar que a diferença básica é que a função retorna obrigatoriamente um valor como resultado e o procedimento não.

5.1 Procedimentos

Um procedimento é um bloco de código precedido de um cabeçalho que contém o Nome do procedimento e seus parâmetros. Com isto, podemos fazer

referência ao bloco de código de qualquer ponto do algoritmo através do seu nome e passando os seus parâmetros.

Sintaxe:

Procedimento Nomedoprocedimento (*parâmetros*)
Variáveis
Inicio
    <Comandos>
Fim

Um “PROCEDIMENTO” é um tipo de subrotina que é ativada através da colocação de seu Nome em alguma parte do programa. Desta forma, assim que o Nome de um “PROCEDIMENTO” é encontrado, ocorre um desvio no programa, para que os comandos da subrotina sejam executados. Ao término da subrotina, a execução retornará ao ponto subseqüente a chamada do “PROCEDIMENTO”.

Exemplos:

Algoritmo um
Variáveis
    Inteiro Número, N
Inicio
    LEIA N
    EscreveNoVideo
    Escreva “fim”
Fim
Procedimento EscreveNoVideo
Inicio
    Para Número **de** 1 **até** N **Faça**
        Escreva Número
    Fimpara
Fim

Algoritmo dois
Variáveis
    Literal N

Inteiro I

Caracter S

Inicio

Ler

Escrever

Fim

Procedimento ler

Inicio

Escreva “Nome”

Leia N

Escreva “Idade”

Leia I

Escreva “Sexo”

Leia S

Fim

PROCEDIMENTO Escrever

Inicio

Escreva “Nome”, N

Escreva “Idade”, I

Escreva “Sexo”, S

Fim

5.2 Função

Uma função é semelhante a um procedimento, sendo que esta deve retornar, obrigatoriamente, um valor em seu nome, desta forma, é necessário declarar, no cabeçalho da função, qual o seu tipo.

Sintaxe:

Função NomeDaFuncao (*parâmetros*) : Tipo_da_Funcao

Variáveis

Inicio

<Comandos>

NomeDaFuncao <Expressão de retorno>

Fim

Ao criarmos uma função e definirmos o tipo do retorno, poderemos utilizar todos os tipos de variáveis vistos até agora (inteiro, literal, lógico,...).

Outra diferença entre um procedimento e uma função é sobre a forma de referencia dentro do algoritmo. Um procedimento, para ser chamado, basta em uma linha de comando do algoritmo fazermos a chamada direta dele, uma função funciona diferente, porque ela tem um valor como retorno. Com isso iremos trabalhar a chamada da função semelhante a uma variável (exceto que não poderá receber um valor).

Exemplos:

A ← NomedaFuncao

A ← NomedaFuncao * 2

Escreva “Resultado:”,NomedaFuncao

Algoritmo Exemplo

Variaveis

Real X, Y

Início

Escreva “Digite um número”

Leia X

Y←QUAD

Escreva “Y =“, Y

Fim.

Função QUAD: **Real**

Início

QUAD ← X * X

Fim

5.3 Variáveis Globais e Locais

Variáveis globais são aquelas declaradas no início de um algoritmo. Estas variáveis são visíveis (isto é, podem ser usadas) no algoritmo principal e por todas as subrotinas.

Variáveis locais são aquelas definidas dentro de uma subrotina e, portanto, somente visíveis (utilizáveis) dentro do mesmo. Outras subrotinas, ou mesmo o algoritmo principal, não podem utilizá-las.

Exemplo:

```
Algoritmo Teste
Variáveis
    Inteiro I
Inicio
    Para i de 1 até 10 Faça
        EscreveNoVídeo
    Fimpara
Fim
Procedimento EscreveNoVídeo
Variáveis
    Inteiro Número, N
Inicio
    Leia N
    Para número de1 até N Faça
        Escreva Número
        Escreva I
    Fimpara
Fim
```

É importante notar no exemplo anterior que:

- A variável I é global, e é visível em todo o algoritmo, inclusive dentro do procedimento.
- As variáveis Número e N são locais do procedimento, ou seja, passam a existir no inicio da execução dele e deixar de existir no momento de sua finalização.

A utilização de variáveis destas formas é muito interessante, pois além de tornar a organização das variáveis mais fácil, ajuda na economia de memória, pois se estivermos executando uma subrotina, as variáveis das outras ainda não estarão utilizando espaço ainda.

5.4 Passagem de Parâmetros

Até agora vimos que para ativar uma subrotina bastaria colocar o seu Nome em alguma parte do programa. Mas isto nem sempre significa que o trabalho de escrever o programa irá diminuir. Com o que vimos até agora, dependendo da tarefa a ser realizada pela subrotina, o trabalho de um programador pode até ser bem complicado. Por Exemplo, como faríamos para calcular o somatório de 10 números 2 a 2, sem perde-los? Poderíamos, por Exemplo, criar 5 subrotinas, uma para cada somatório. Isto sem dúvida resolveria esta situação, mas, e se fossem 100 somatórios? Ou 1000? Seria realmente uma tarefa muito trabalhosa ter de escrever 100, ou 1000 subrotinas. Como já foi dito, as subrotinas foram criadas para serem genéricas o bastante para se adaptarem a qualquer situação, visando justamente à possibilidade de reutilização do código. Para realizar esta "mágica", foi criado o conceito de passagem de parâmetros, ou seja, passar informações para serem tratadas dentro da Subrotina.

Agora temos uma espécie de declaração de variáveis no cabeçalho da subrotina, sendo que elas não precisam ser redeclaradas, mas não nos impede de criarmos outras variáveis dentro da subrotina, e serão as variáveis utilizadas como parâmetros. As variáveis do mesmo tipo são separadas por vírgulas (,) e variáveis de tipos diferentes são separadas por ponto e vírgula (;). Esses conceitos são validos tanto para procedimentos como para funções.

```
Algoritmo Um
Variáveis
    Inteiro Número
    Literal Funcionário
Inicio
    Leia Número
    Leia Funcionário
    EscreveNome (Número, Funcionário)
Fim
Procedimento EscreveNome (Inteiro N; Literal Nome)
Variáveis
    Inteiro I
```

```
Inicio
    Para i de 1 até n Faça
        Escreva Nome
    Fimpara
Fim
```

O mesmo exemplo acima pode ser resolvido da forma a seguir, onde não são utilizadas variáveis globais, apenas locais.

```
Algoritmo Dois

Inicio
    Inicial
Fim

Procedimento Inicial
Variáveis
    Inteiro Número
    Literal Funcionário
Inicio
    Leia Número
    Leia Funcionário
    EscreveNome (Número, Funcionário)
Fim
Procedimento EscreveNome (Inteiro N; Literal Nome)
Variáveis
    Inteiro I
Inicio
    Para i de 1 até n Faça
        Escreva Nome
    Fimpara
Fim
```

No Exemplo acima, o valor contido em “Número” será passado para o parâmetro “N”, da mesma forma que o valor contido na variável “Funcionário” será passada para o parâmetro “Nome”. Note que os nomes são diferentes.

Deve-se tomar cuidado com os tipos utilizados nas variáveis, pois eles devem ser iguais (numero é do mesmo tipo que N e Funcionário é do mesmo tipo que Nome), e também com a quantidade de parâmetros, no exemplo acima o procedimento possui dois, então sempre que for chamado deveremos enviar 2 também.

No Exemplo abaixo é feito somatório de 4 valores informados pelo usuário de forma diferente, onde primeiramente soma-se n1 e n2, depois n3 e n4, para somente depois adicionar os dois resultados onde finalmente será feito o calculo final.

```
Algoritmo Três
Inicio
    Inicial
Fim

Procedimento Inicial
Variáveis
    Real n1, n2, n3, n4
Inicio
    Leia n1
    Leia n2
    Leia n3
    Leia n4
    Leia Funcionário
    Escreva Soma(Soma(n1,n2), Soma(n3,n4))
Fim
Função Soma (Real A, B):Real
Inicio
    Soma ← A + B
Fim
```

Importante: Para variáveis ‘Comuns’ utiliza-se o conceito de passagem de parâmetros por Valor, ou seja, o conteúdo original é mantido intocável, há também o conceito de passagem de parâmetros por referencia, que será visto mais adiante, onde as alterações efetuadas no parâmetro reflete no valor original.

5.5 Passagem de Parâmetros por Valor e por Referencia

Já sabemos que ao declarar uma função ou procedimento, é possível declarar um ou mais parâmetros. Ao utilizar uma função ou procedimento, devemos informar os valores para os parâmetros, na mesma ordem em que foram definidos durante a declaração da função ou procedimento. Este processo é conhecido como passagem de parâmetros para a função ou procedimento, ou seja, ao chamar a função ou procedimento, passamos valores que serão utilizados pelo código da função ou procedimento.

Existem duas maneiras diferentes de fazer a passagem dos parâmetros e é importante para o programador, entender exatamente a diferença entre estas duas maneiras:

- Passagem por Valor – Variáveis 'Comuns' (inteiro, real, Literal,...);
- Passagem por Referência – Variáveis indexadas e objetos.

Este conceito é muito importante e vamos entendê-lo através de alguns exemplos simples. Inicialmente vamos ver o que acontece quando utilizamos o tipo de passagem por valor.

Passagem de parâmetros por Valor

```
Algoritmo Um
Inicio
    Main
Fim
Procedimento Main
Variáveis
    Inteiro A
Inicio
    A←10
    Muda (A)
    Escreva A
Fim
Procedimento Muda(inteiro B)
Inicio
    B←B*2
    Escreva B
Fim
```

Neste algoritmo acima temos uma variável chamada 'A' que recebe um valor 10. Este valor é enviado por parâmetro para o procedimento 'Muda' onde utilizamos o parâmetro 'B'. Neste momento temos duas variáveis (A e B) com o mesmo valor

(10). Quando realizamos a alteração do conteúdo da variável ‘B’ o valor de ‘A’ permanece inalterado, ou seja, será escrito na tela os valores 20 e 10.

Até este momento nada de anormal, agora veremos a utilização da passagem de um vetor como parâmetro.

Passagem de parâmetros por Referencia

Ao receber um parâmetro por referência, as alterações que a função/procedimento fizer, serão feitas diretamente na variável original, pois agora, a função/procedimento tem acesso ao endereço da variável na memória e não mais apenas uma cópia do seu valor.

Para exemplificar este conceito, considere o código a seguir, onde temos um procedimento chamado **Main**, na qual é declarado um vetor A, do tipo inteiro, inicializamos este vetor.

```
Algoritmo Dois
Inicio
    Main
Fim
Procedimento Main
Variáveis
    vetor [2] de Inteiro A
Inicio
    A[0]←10
    A[1]←20
    Muda (A)
    Escreva A[0]
    Escreva A[1]
Fim
Procedimento Muda(inteiro[] B)
Inicio
    B[0]← B[0] +10
    B[1]← B[1] +10
Fim
```

No exemplo é armazenado no vetor os valores 10 e 20, e passado por parâmetro para o vetor B, como a passagem é feita por referencia, toda alteração feita no vetor B é refletida em A. No caso o vetor B passara a ter os mesmos valores de A (10 e 20), e quando são feitas as alterações em B (20 e 30), esses também serão os novos valores armazenados no vetor A, portanto neste exemplo serão

mostrados na tela os valores 20 e 30. Lembre-se: Todas as passagens de parâmetro em um vetor são feitas por **REFERENCIA.**

```
Algoritmo Tres
Inicio
    Main
Fim
Procedimento Main
Variáveis
    vetor [10] de Nomes A
Inicio
    Ler (A)
    Mostrar (A)
Fim

Procedimento Ler(inteiro[] B)
Variáveis
    Inteiro I
Inicio
    Para i de 0 até 9 Faça
        Leia b[i]
    FimPara
Fim

Procedimento Mostrar (inteiro[]B)
Variáveis
    Inteiro I
Inicio
    Para i de 0 até 9 Faça
        Escreva b[i]
    FimPara
Fim
```

5.6 Recursividade

Diz-se que uma função ou um procedimento é recursivo quando ele chama a si próprio, esta característica pode, a princípio, parecer estranha, ou até mesmo desnecessária devido ao nível de programas o qual estamos trabalhando, mas o uso da recursividade, muitas vezes, é a única forma de resolver problemas complexos.

```
Procedimento Recursão (inteiro A)
Inicio
    Se a > 0 então
        Escreva A
        Recursão (A - 1)
```

```
    Fimse
Fim

Procedimento Recursão (Inteiro A)
Inicio
    Se a > 0 Então
        Recursão (A -1)
        Escreva A
    Fimse
Fim
```

No primeiro exemplo, a saída gerada será a seguinte seqüência de números: 5 4 3 2 1. No segundo exemplo, a saída gerada será a seguinte seqüência de números: 1 2 3 4 5.

```
Algoritmo rec
Variaveis
    Inteiro F
    Função FatRec (inteiro N):Inteiro
    Inicio
        Se N=1 então
            FatRec · 1
        Senão
            FatRec ← N*FatRec (N-1)
        Fimse
    Fim
Inicio
    Escreva “Valor:”
    Leia F
    F ← FatRec(F)
    Escreva "Resultado:”,F
Fim
```