ALGORITMOS

Prof. Lourival
lourival@dimap.ufrn.br

TERCEIRA PARTE

CAPÍTULO 8: SUBALGORITMOS

8.1 – INTRODUÇÃO

Sempre é possível dividir problemas grandes e complicados em problemas menores e de solução mais simples. A decomposição de um problema é fator determinante para a redução da complexidade. A complexidade é sinônimo de variedade, ou seja, a quantidade de situações diferentes que um problema pode apresentar. Assim, quando decompomos um problema em subproblemas, estamos invariavelmente dividindo também a complexidade e, por conseqüência, simplificando a resolução. Outra grande vantagem da decomposição é que permite focalizar a atenção em um problema pequeno de cada vez, o que ao final produzirá uma melhor compreensão do todo. Cada um desses pequenos problemas será solucionado através de subalgoritmos. Nesse caso, o algoritmo completo é dividido num algoritmo principal e diversos subalgoritmos (tantos quantos forem necessários ou convenientes). O algoritmo principal é aquele por onde começa a execução, e chama, eventualmente, os demais subalgoritmos.

Subalgoritmo é um algoritmo que, geralmente, resolve um pequeno problema, e que está subordinado a um outro algoritmo. Esta subordinação deve-se ao fato de que o subalgoritmo só será acionado se solicitado pelo algoritmo principal. É possível que um subalgoritmo chame outro subalgoritmo.

Em resumo, os subalgoritmos são importantes:

- na subdivisão de algoritmos complexos, facilitando o seu entendimento;
- na estruturação de algoritmos, facilitando, principalmente, a detecção de erros e a documentação de sistemas;
- na modularização de sistemas, que facilita a manutenção de softwares e reutilização de subalgoritmos já implementados.

8.2 – ELEMENTOS DE UM SUBALGORITMO

A definição de um subalgoritmo consta de:

- cabeçalho, onde estão definidos o ***nome*** e o ***tipo*** do subalgoritmo, bem como os seus ***parâmetros*** e suas ***variáveis locais***;
- corpo do subalgoritmo, onde se encontram as instruções, que serão executadas cada vez que ele é chamado.

O *nome de um subalgoritmo* é um nome simbólico pelo qual ele é chamado por outro algoritmo.

As *variáveis locais* são aquelas definidas dentro do próprio subalgoritmo e só podem ser utilizadas pelo mesmo.

Os *parâmetros* são canais por onde os dados são transferidos pelo algoritmo chamador a um subalgoritmo, e vice-versa. Para ser executado, às vezes, um subalgoritmo precisa receber dados do algoritmo que o chamou, e ao terminar sua tarefa, precisa fornecer os resultados ao mesmo. Esta comunicação bidirecional é chamada de ***passagem de parâmetros***.

O tipo de um subalgoritmo é definido em função do número de valores que o subalgoritmo retorna ao algoritmo que o chamou.

Os subalgoritmos podem ser de dois tipos:

- as funções, que retornam um, e somente um valor ao algoritmo chamador;
- os procedimentos, que retornam vários valores, ou nenhum, ao algoritmo chamador.

8.3 – FUNÇÕES

O conceito de funções é originário da idéia de função matemática, onde um valor é calculado a partir de outro(s) valor(es) fornecido(s) à função.

Forma geral de uma função (sintaxe):

Função <nome> (<parâmetros>)

<declaração_de_variáveis>

Início

<instruções>

Fim

8.3.1 – INSTRUÇÃO Retorne

A instrução Retorne é um comando simples usado apenas nas funções e tem o efeito de parar a execução da função e enviar um valor para o algoritmo principal ou para outro subalgoritmo que o tenha chamado. Toda função deve ter em seu corpo de instruções, pelo menos, uma instrução Retorne.

Sintaxe:

Retorne (<expressão>)

Exemplos:

```
Retorne ( 1 )          Retorne ( .V. )
Retorne ( area )       Retorne ( pi*r*r )
Retorne ( nome1 )      Retorne ( 3.*x**2 + 4.*x – 1.5 )
```

Exemplo de uma função:

```
Função quad(w)
        Real: w, z
Início
        z ← w*w
        Retorne (z)
Fim
```

8.3.2 – CHAMADA DE UMA FUNÇÃO

A chamada de uma função por parte do algoritmo principal ou por outro subalgoritmo é feita pelo simples aparecimento de seu nome, seguido pelos respectivos parâmetros entre parênteses, dentro de uma expressão. Os parâmetros utilizados no algoritmo principal, no ato da chamada da função (chamados parâmetros reais) devem manter uma correspondência (número, ordem e tipo) com os parâmetros definidos na função (chamados parâmetros formais).

A função é executada, e ao seu término, o trecho do comando que a chamou é substituído pelo valor retornado pela mesma dentro da expressão em que se encontra, e a avaliação desta prossegue normalmente.

Exemplo de um algoritmo principal utilizando a função do exemplo anterior:

```
Algoritmo Quadrado
        Real: x, y
Início
        Escreva (“Digite um número: “)
        Leia ( x )
        y ← quad (x)
        Escreva (“ y = “ , y )
Fim
```

8.3.3 – EXEMPLOS DE ALGORITMOS COM FUNÇÕES

1) Faça uma função para calcular o fatorial de um número inteiro. Utilize esta função num algoritmo principal para calcular o número de combinações de m elementos tomados p a p, sendo m e p dados.

```
Função fatorial (x)
        Inteiro: x, f, i
Início
        f ← 1
        Para i de 2 até x faça
                f ← f*i
        Fim_para
        Retorne ( f )
Fim
Algoritmo Combinação
        Inteiro: m, p, comb
Início
        Escreva (“Digite dois números para obter as combinações, ”)
        Repita
                Escreva(“valores não-negativos: “)
                Leia (m, p)
        Até ( m >= 0 .E. p >= 0 .E. m >= p )
        comb ← fatorial(m) / ( fatorial(m-p)*fatorial(p) )
        Escreva (“Número de combinações = “, comb )
Fim
```

2) Faça uma função para determinar se um número inteiro é par ou não. Utilize esta função para calcular dois somatórios: um com os números pares e outro com os números ímpares, dentre n números inteiros positivos dados.

```
Função par ( p )
        Inteiro: p
Início
        Se (p/2*2 = p) então
                Retorne (.V.)
        senão
                Retorne (.F.)
        Fim_se
Fim
Algoritmo Soma_par_impar
        Inteiro: n, x, i, sp, si
Início
        Repita
                Escreva(“Quantos números? “)
                Leia (n)
        Até (n>0)
        sp ← 0
        si ← 0
        Escreva (“Digite os números “)
        Para i de 1 até n faça
                Repita
```

```
                    Escreva (“positivos.”)
                    Leia (x)
            Até (x>0)
            Se (par  (x)) então
                    sp ← sp + x
            senão
                    si ←  si + x
            Fim_se
        Fim_para
        Escreva (“Soma dos pares =  “, sp)
        Escreva (“Soma dos ímpares = “, si)
Fim
```

3) Faça uma função para calcular o produto escalar entre dois vetores de mesmas dimensões e a utilize para calcular o cosseno do ângulo entre dois vetores dados.

```
Função prodesc (x, y, n)
        Real: x[20], y[20], pe
        Inteiro: n, i
Início
        pe ←  0
        Para i de 1 até n faça
                pe ←  pe + x[i]*y[i]
        Fim_para
        Retorne (pe)
Fim
Algoritmo Cosseno
        Real: a[20], b[20], cos, pab, ma, mb
        Inteiro: n, i
Início
        Repita
                Escreva(“Digite a dimensão do vetor: “)
                Leia (n)
        Até (n>0 .E. n<=20)
        Escreva (“Digite os números do prim. Vetor:”)
        Para i de 1 até n faça
                Leia (a[i])
        Fim_para
        Escreva (“Digite os números do seg. vetor:”)
        Para i de 1 até n faça
                Leia (b[i])
        Fim_para
        pab ←  prodesc (a, b, n)
        ma ←  prodesc (a, a, n)**0.5
        mb ←  prodesc (b, b, n)**0.5
        cos ←  pab/(ma*mb)
        Escreva (“Cosseno do ângulo entre a e b = “, cos)
Fim
```

8.3.4 – EXERCÍCIOS PROPOSTOS DE FUNÇÕES

1) Faça uma função para calcular o mmc entre dois números inteiros.

2) Faça uma função para informar se um número inteiro é primo ou não. Faça um algoritmo principal para imprimir um determinado número par dado, diferente de dois, como a soma de dois números primos (isto é sempre possível). Por exemplo: 28 = 23 + 5 ou 28 = 17 + 11.

3) Faça uma função para calcular o valor absoluto de um número.

4) Faça uma função para calcular o módulo de um vetor com n números.

5) Faça uma função para verificar se um vetor possui elementos repetidos.

6) Faça uma função para verificar se um caractere pertence a uma cadeia de caracteres.

7) Faça uma função para converter uma letra minúscula em uma letra maiúscula.

8.4 – PROCEDIMENTOS

Um procedimento é um subalgoritmo que retorna vários valores, ou nenhum, ao programa principal, ou a outro subalgoritmo que o chame. Estes valores são sempre retornados por meio dos parâmetros, e nunca explicitamente como no caso das funções que usa a instrução Retorne.

Forma geral de um procedimento (sintaxe):

Procedimento <nome> (<parâmetros>)

    <declaração_de_variáveis>

Início

    <instruções>

Fim

Exemplo de um procedimento:

```
Procedimento Retangulo(lado1, lado2, perim, area)
        Real: lado1, lado2, perim, area
Início
        perim ← 2*(lado1 + lado2)
        area ← lado1 * lado2
Fim
```

8.4.1 – CHAMADA DE UM PROCEDIMENTO

A chamada de um procedimento só é feita em comandos isolados dentro do algoritmo principal ou outro subalgoritmo, como os comandos simples, tipo as instruções Leia, Escreva, Retorne, etc., e nunca no meio de expressões ou em atribuições como no caso de funções.

Exemplo de um algoritmo principal com o procedimento anterior:

```
Algoritmo Area_perimetro
        Real: x, y, p, a
Início
        Escreva("Digite dois números: ")
        Leia( x, y )
        Retangulo (x, y, p, a)
        Escreva("Perímetro = ", p, "   Area = ", a)
Fim
```

8.4.2 – MECANISMOS DE PASSAGEM DE PARÂMETROS

Dados são passados pelo algoritmo principal (ou outro subalgoritmo) ao subalgoritmo, ou retornados por este ao primeiro, por meio de parâmetros.

Parâmetros formais são os nomes simbólicos (variáveis) introduzidos no cabeçalho dos subalgoritmos, utilizados na definição dos parâmetros do mesmo. Dentro do subalgoritmo trabalha-se com estes nomes da mesma forma como se trabalha com variáveis locais.

Parâmetros reais são aqueles que substituem os parâmetros formais quando da chamada de um subalgoritmo. Os parâmetros formais são úteis somente na definição do subalgoritmo. Os parâmetros reais podem ser diferentes a cada chamada de um subalgoritmo.

Nos procedimentos os parâmetros são divididos em parâmetros de entrada e parâmetros de saída, já que os valores a serem retornados pelo procedimento (parâmetros de saída) são transferidos através dos parâmetros, enquanto que os dados passados pelo algoritmo principal para o procedimento (parâmetros de entrada) são também através de parâmetros.

Os parâmetros reais substituem os parâmetros formais no ato da chamada de um subalgoritmo. Esta substituição é denominada passagem de parâmetros e pode se dar por dois mecanismos: passagem por valor (ou por cópia) ou passagem por referência.

Na passagem de parâmetros por valor o parâmetro real é calculado e uma cópia de seu valor é fornecida ao parâmetro formal, no ato da chamada do subalgoritmo. As modificações feitas no parâmetro formal não afetam o parâmetro real, durante a execução do subalgoritmo, pois trabalha-se apenas com uma cópia do mesmo. Os parâmetros formais possuem locais de memória exclusivos para que possam armazenar os valores dos parâmetros reais.

Na passagem de parâmetros por referência não é feita uma reserva de espaço de memória para os parâmetros formais. Quando um subalgoritmo com parâmetros passados por referência é chamado, o espaço de memória ocupado pelos parâmetros reais é compartilhado pelos parâmetros formais correspondentes. Assim, as eventuais modificações feitas nos parâmetros formais também afetam os parâmetros reais correspondentes.

Como norma, vamos adotar que, quando a passagem de parâmetros for feita com variáveis simples, a passagem será por valor, e quando for feita com variáveis indexadas, a passagem de parâmetros será por referência. Isto quer dizer que se for do nosso interesse manter intactos vetores ou matrizes após a chamada de subalgoritmos, que os utilizaram e os modificaram como parâmetros formais no ato da chamada dos mesmos, então devemos antes disso guardar cópias dos seus valores.

8.4.3 – EXEMPLOS DE ALGORITMOS COM PROCEDIMENTOS

1) Faça um procedimento que soma duas matrizes numéricas. Faça um outro procedimento que obtém a transposta de uma matriz. Faça um algoritmo principal para somar uma matriz quadrada dada com a sua transposta utilizando os procedimentos.

```
Procedimento Soma_matriz (a, b, m, n, soma)
        Real: a[10, 10], b[10, 10], soma[10, 10]
        Inteiro: m, n, i, j
Início
        Para i de 1 até m faça
                Para j de 1 até n faça
                        soma[i, j] ← a[i, j]+b[i, j]
                Fim_para
        Fim_para
Fim
Procedimento Transposta (x, p, q, y)
        Real: x[10, 10], y[10, 10]
        Inteiro: p, q, i, j
Início
        Para i de 1 até q faça
                Para j de 1 até p faça
                        y[i, j] ← x[j, i]
                Fim_para
```

```
        Fim_para
Fim
Algoritmo Matriz_quadrada
        Real: mat[10, 10], t[10, 10], s[10, 10]
        Inteiro: n, i, j
Início
        Repita
                Escreva("Digite a ordem da matriz quadrada <=10:")
                Leia (n)
        Até (n>0 .E. n<=10)
        Escreva("Digite os números da matriz: ")
        Para i de 1 até n faça
                Para j de 1 até n faça
                        Leia (mat[i, j])
                Fim_para
        Fim_para
        Transposta (mat, n, n, t)
        Soma_matriz (mat, t, n, n, s]
        Escreva("Matriz resultante: ")
        Para i de 1 até n faça
                Para j de 1 até n faça
                        Escreva( s[i, j])
                Fim_para
        Fim_para
Fim
```

2) Faça um procedimento para calcular o somatório de n números. E utilize este procedimento para calcular o desvio padrão de n números dados. (Essa questão também pode ser feita com função.)

```
Procedimento Soma_vetor (v, n, som)
        Real: v[30], som
        Inteiro: n, i
Início
        som ← 0.0
        Para i de 1 até n faça
                som ← som + v[i]
        Fim_para
Fim
Algoritmo Desvio_padrão
        Real: x[30], desvio, y[30], s, m
        Inteiro: n, i
Início
        Repita
                Escreva("Quantos números?")
                Leia (n)
        Até (n>0 .E. n<=30)
        Escreva("Digite os números: ")
        Para i de 1 até n faça
                Leia (x[i])
        Fim_para
        Soma_vetor (x, n, s)
        m ←  s/n
        Para i de 1 até n faça
```

```
            y[i] ← (x[i] – m)**2
        Fim_para
        Soma_vetor (y, n, s)
        desvio ← ( s /(n-1))**0.5
        Escreva("Desvio padrão = ", desvio)
Fim
```

3) Faça um procedimento para calcular a soma dos elementos de um vetor com n números inteiros. Faça um outro procedimento para gerar todos os divisores próprios de um número inteiro. Utilize estes procedimentos para imprimir todos os números "perfeitos" menores que um certo número dado.

```
Procedimento Soma (x, n, som)
        Inteiro: x[30], n, som, i
Início
        soma ← 0
        Para i de 1 até n faça
                som ← som + x[i]
        Fim_para
Fim
Procedimento Divisores (k, v, n)
        Inteiro: k, v[30], n, i
Início
        n ← 0
        Para i de 1 até k/2 faça
                Se (k/i*i = k) então
                        n ← n + 1
                        v[n] ← i
                Fim_se
        Fim_para
Fim
Algoritmo Perfeito
        Inteiro: num, per, x[30], q, s
Início
        Repita
                Escreva ("Digite um número inteiro positivo:")
                Leia (num)
        Até (num > 0)
        Para per de 2 até num-1 faça
                Divisores (per, x, q)
                Soma (x, q, s)
                Se (per = s) então
                        Escreva ("Número perfeito = ", per)
                Fim_se
        Fim_para
Fim
```

8.4.4 – EXERCÍCIOS PROPOSTOS DE SUBALGORITMOS

1) Faça um procedimento para calcular o maior e o menor número de um conjunto de números inteiros dados. Faça um outro procedimento para imprimir se um número inteiro é par ou ímpar. Faça um algoritmo principal para ler n números inteiros e imprimir o maior e o menor deles informando se cada um deles é par ou ímpar.

2) Faça uma função para calcular o maior elemento de um conjunto numérico. Utilize esta função para calcular os maiores elementos de cada coluna de uma matriz qualquer dada.

3) Faça um subalgoritmo para obter quantos e quais são os divisores de um número inteiro positivo. Faça um outro subalgoritmo que calcula o maior número de um conjunto de números inteiros e a sua posição dentro do conjunto. Utilize estes subalgoritmos num algoritmo principal para ler um conjunto de números inteiros positivos e imprimir cada um deles juntamente com o número de divisores respectivo, e ainda informar qual o número do conjunto que possui o maior número de divisores. Por exemplo, dado o conjunto {4, 17, 36, 55}, teremos como solução:

```
4     possui 3 divisores
17    possui 2 divisores
36    possui 9 divisores
55    possui 4 divisores
36 é o número que possui o maior número de divisores.
```

CAPÍTULO 9 – ESTRUTURA DE DADOS HETEROGÊNEOS

9.1 – INTRODUÇÃO

Existem quatro tipos de dados simples, chamados tipos básicos, que o computador manipula, que já conhecemos e utilizamos (são eles: os reais, os inteiros, os literais e os lógicos). Com eles podemos definir novos tipos que possibilitam agrupar um conjunto de dados homogêneos (mesmo tipo) sob um único nome (a exemplo dos vetores e matrizes), ou de tipos heterogêneos (tipos diferentes). A esse novo tipo de dados heterogêneos damos o nome de registros.

Registros são conjuntos de dados logicamente relacionados, mas de tipos diferentes.

As matrizes são tipos de variáveis que agrupam dados similares, enquanto que os registros agrupam, geralmente, dados desiguais. Aos itens de dados de um registro dá-se o nome de “membros” ou “componentes”, enquanto que aos itens de uma matriz (ou vetor) dá-se o nome de elementos”.

9.2 – REGISTROS

Registros correspondem a conjuntos de posições de memória conhecidos por um mesmo nome e individualizados por identificadores associados a cada conjunto de posições.

Sintaxe para criação de um registro:

Registro = <nome_do_registro>

<componentes_do_registro>

Fim_registro

O <nome_do_registro> é um nome escolhido pelo programador e será considerado como um novo tipo de dados, como os tipos básicos.
<componentes_do_registro> é o local onde o programador definirá as variáveis desse novo tipo, se valendo de outros tipos já definidos.

A definição de um registro deve figurar no cabeçalho do algoritmo (ou subalgoritmo) antes da declaração das variáveis relativas a ele. Um registro pode ser definido em função de outros registros já definidos.

Exemplo 1:

```
Registro = Data
        Inteiro: dia
        Literal[10]: mês
        Inteiro: ano
Fim_registro
Data: natal, aniversario
```

Exemplo 2:

```
Registro = Aluno
        Inteiro: matricula
        Real: nota[3], media
Fim_registro
Aluno: Ana
```

Exemplo 3:

```
Registro = Dados
        Literal[40]: nome
        Inteiro: idade
        Lógico: sexo
        Real: salario
Fim_registro
Dados: x, y
```

A especificação no exemplo 3 corresponde a criação de duas áreas de memória (x e y), capazes de conter, cada uma, quatro subdivisões de tipos diferentes.

Uma ficha cadastral, tipo a que preenchemos em bancos, supermercados, lojas etc., é o melhor modelo para compararmos com um registro e o que acontece na memória do computador:

Nome:_____

Idade:_____ Sexo: Masc. ☐ Fem. ☐

Salário:_____

Observe que as variáveis que irão assumir valores desse tipo (registro) estão sendo declaradas como Dados: x, y .

A referência ao conteúdo de um dado componente do registro será indicada por:

<variável_registro>.<variável_componente>

Exemplo: x.nome ← “Paulo César do Nascimento”
x.idade ← 32
x.sexo ← .V.
x.salario ←2500.00
y.nome ← “Maria Clara Machado”
y.idade ← 25
y.sexo ← .F.
y.salario ← 3500.00

Exemplo: Registro com registro.

```
Registro = Ender
        Literal[40]: rua
        Inteiro: num
        Literal[10]: cep
Fim_registro
Registro = Funcionario
        Literal[40]: nome
        Ender: endereco
        Literal[20]: cidade, estado
        Real: salario
Fim_registro
Funcioario: x
```

Nesse caso a referência ao conteúdo de um componente do registro Funcionário será indicado, por exemplo, como:

```
x.nome ← “Carla Pontes de Lima”
x.endereco.rua ← “Bumba Meu Boi”
x.endereco.num ← 3125
x.endereco.cep ← “72341-530”
x.cidade ← “Patos”
x.estado ← “PB”
x.salario ← 2458.75
```

CAPÍTULO 10 – ARMAZENAMENTO DE DADOS PERSISTENTES

10.1 – INTRODUÇÃO

Variáveis simples ou compostas (variáveis indexadas e registros) são entidades normalmente usadas para declarar estruturas de dados alocadas no mesmo ambiente do programa. Entretanto, pode não haver espaço suficiente para armazenar um volume muito grande de dados, ou então haver necessidade de armazenamento de um conjunto de informações por um período muito longo de tempo. A alternativa para isto é o arquivo, com a vantagem dele poder ser usado por vários outros programas.

Exemplos de arquivos com volume muito grande de dados e necessidade de armazenamento por período longo, são os arquivos de assinantes de uma companhia telefônica, ou de clientes de um banco, ou dos contribuintes da Receita Federal, etc.

10.2 – ARQUIVOS

Um arquivo é um conjunto de registros armazenados em um dispositivo de memória (disco, fita, disquete, etc.). Portanto, uma estrutura de dados, no qual cada registro não ocupa uma posição fixa dentro da estrutura, não possuindo tamanho preestabelecido. Pode ser criado, consultado, processado e removido por algoritmos distintos. Pode-se comparar um arquivo com aqueles móveis antigos de aço, chamados arquivos de pastas, tão comuns em repartições públicas (escolas, bibliotecas, delegacias, etc.).

As operações básicas que podem ser feitas em um arquivo através de um algoritmo são:

- obtenção de um registro (Leitura);
- inserção de um novo registro (Gravação);
- modificação de um registro (Alteração);
- exclusão de um registro (Exclusão).

Os arquivos podem ser organizados de duas formas, quanto à sua criação ou acesso. São elas:

- Seqüencial, onde os registros são obtidos ou inseridos no arquivo um após o outro;
- Direta, onde o acesso ao registro é feito em ordem aleatória.

No algoritmo, o arquivo deve ser declarado, no cabeçalho, e aberto, no corpo de instruções, antes que qualquer operação possa ser feita.

O acesso a um arquivo dentro do algoritmo é feito através da leitura e escrita de registros. No final do algoritmo, ou quando houver necessidade, o arquivo deve ser fechado.

Sintaxe para declaração de arquivos:

Arquivo: <nome_arquivo><organização><nome_registro>

Onde:

<nome_arquivo> é o nome que será usado para referenciar o arquivo;

<organização> indica o tipo de organização do arquivo que pode ser Seqüencial ou Direta;

<nome_registro> é o nome do registro que será usado para se ter acesso ao arquivo.

Exemplo:

```
Registro = Ender
        Literal[30]: nome, rua
        Inteiro: numero
Fim_registro
Arquivo: Agenda Sequencial Ender
Ender: end1, end2
```

Neste exemplo foi declarado um arquivo de nome Agenda, composto de registros que contém nome, rua e número. Esquematicamente pode se ter o seguinte:

ARQUIVO AGENDA:

NOME	RUA	NÚMERO
Roberto Carlos	Liberdade	48
Nicete Bruno	Bumba Meu Boi	113
Carlos Chagas	Ulisses Caldas	95
.	.	.
.	.	.
.	.	.
Pedro Pimenta	Serra do Doutor	215

Sintaxe para abertura/fechamento de arquivos:

Abra (<nome_arquivo>, Leitura)
Abra (<nome_arquivo>, Escrita)
Abra (<nome_arquivo>, Leitura/Escrita)
Feche (<nome_arquivo>)

10.2.1 – ORGANIZAÇÃO SEQÜENCIAL

A principal característica da organização seqüencial é a de que os registros são armazenados contiguamente, isto é, um após o outro. Assim sendo, o acesso aos registros do arquivo, tanto na leitura como na escrita, são feitos seqüencialmente, ou seja, a leitura de um registro só é possível após a leitura de todos os registros anteriores, e a escrita de um registro só é possível após o último registro.

Geralmente não sabemos quantos registros têm num arquivo que desejamos acessá-lo, nem tão pouco sabemos o conteúdo do último registro. Para isto vamos supor que no final de cada arquivo sempre inserimos um dado conhecido em algum dos campos do último registro. Será portanto um registro posto de propósito para sabermos até onde pesquisar (critério de parada num laço).

As operações de Entrada/Saída de um arquivo de organização seqüencial são feitas nos algoritmos da seguinte forma:

Leitura:	Leia (<nome_arquivo>, <variável_registro>)
Escrita:	Escreva (<nome_arquivo>, <variável_registro>)

No exemplo anterior teríamos:

```
Leia (Agenda, end1)
Escreva (Agenda, end2)
```

10.2.2 – EXEMPLO DE ALGORITMO COM ARQUIVO SEQUENCIAL

Supondo que existe um arquivo de nome Pessoal com registros com campos: nome, sexo e salário, fazer um algoritmo que leia este arquivo e crie um outro arquivo de nome Homens composto de registros com nome, sexo e salário só dos homens do arquivo Pessoal.

```
Algoritmo Cópia_homens
        Registro = Dados
                Literal[30]: nome
                Literal[2]: sexo
                Real: salario
        Fim_registro
        Arquivo: Pessoal Sequencial Dados
        Arquivo: Homens Sequencial Dados
        Dados: var1
Início
        Abra( Pessoal, Leitura)
        Abra (Homens, Escrita)
        Leia (Pessoal, var1)
        Enquanto(var1.nome <> “Fim de Arquivo”)faça
                Se (var1.sexo = “M”) então
                        Escreva (Homens, var1)
                Fim_se
```

```
            Leia(Pessoal, var1)
        Fim_enquanto
        Feche (Pessoal)
        Feche (Homens)
Fim
```

10.2.3 – ORGANIZAÇÃO DIRETA

Ao criar um arquivo, podemos utilizar um algoritmo que expresse um padrão de comportamento rígido, com o objetivo de estruturar o arquivo para facilitar sua manipulação.

A circunstância de armazenamento que perfaz esse algoritmo é da localização do registro dentro do arquivo ficar diretamente relacionada a uma informação constituinte deste, ou seja, através de um dos campos do registro podemos determinar o lugar onde este está guardado, podendo acessá-lo de modo instantâneo.

Quando utilizamos um arquivo de organização Direta, podemos acessar um registro específico diretamente, sem nos preocuparmos com seus antecessores, utilizando nesse acesso o mesmo campo que determinou sua posição no arquivo no instante da gravação.

O campo que determina a posição do registro no arquivo é denominado **chave**, pois é a informação capaz de acessar o registro. A chave determinada no algoritmo deve ser única, pois nunca podemos armazenar dois registros diferentes em uma mesma localização.

Para exemplificar um arquivo de organização direta, imaginemos a situação de um professor que deseja armazenar informações referentes a uma de suas turmas, como o nome do aluno e suas três primeiras notas. Para tal, ele utiliza como chave o número de matrícula do aluno, informação que também é parte integrante do registro e é única. Dessa forma não será necessário fazer uma pesquisa por todos os registros do arquivo, mas necessitamos de uma nova instrução para que a posição corrente do arquivo passe a ser a indicada pela chave. Essa instrução terá o nome de **Posicione.**

Sintaxe:

Posicione (<nome_arquivo>, <chave>)

Onde: <nome_arquivo> é o nome do arquivo previamente definido;
<chave> é o componente do registro escolhido para indicar a posição desejada.

10.2.4 – EXEMPLO DE ALGORITMO COM ARQUIVO DIRETO

Vejamos o algoritmo que o professor do exemplo acima utiliza para armazenar os dados de seus alunos.

```
Algoritmo Alunos
        Registro = Ficha
                Literal[30]: nome
                Inteiro: mat
                Real: n1, n2, n3, med
        Fim_registro
        Arquivo: Cadastro Direta Ficha
        Ficha: alun
Início
        Abra (Cadastro, Escrita)
        Repita
                Escreva (“Digite o nome do aluno e sua matrícula:”)
                Leia (alun.nome, alun.mat)
                Se ( alun.mat > 0) então
                        Escreva(“Digite as três notas:”)
                        Leia (alun.n1, alun.n2, alun.n3)
                        alun.med ← ( 4*alun.n1+5*alun.n2+6*alun.n3)/15.
                        Posicione (Cadastro, alun.mat)
                        Escreva (Cadastro, alun)
                Fim_se
        Até (alun.mat <= 0)
        Feche (Cadastro)
        Abra (Cadatro, Leitura)
        Escreva (“Digite uma matrícula válida:”)
        Leia (alun.mat)
        Posicione (Cadastro, alun.mat)
        Leia (Cadastro, alun)
        Escreva (alun.nome, “possui média igual a  “, alun.med)
        Feche (Cadastro)
Fim
```

10.3 – EXEMPLOS RESOLVIDOS DE ALGORITMOS COM ARQUIVOS

1) Uma empresa resolveu diminuir sua folha de pagamento. Para tal, mandou criar um arquivo, a partir do cadastro de todos os seus funcionários, apenas com os funcionários que recebem mais de 40 salários mínimos. Sabendo-se que os registros possuem os campos mostrados na figura abaixo, escreva um algoritmo para criar o arquivo pedido.

REGISTRO

NOME	CARGO	SALÁRIO

```
Algoritmo Salário_alto
        Registro = Funcionarios
                Literal[40]: nome
                Literal[20]: cargo
                Real: salario
        Fim_registro
        Arquivo: Cadastro Sequencial Funcionarios
        Arquivo: Bem_pagos Sequencial Funcionarios
        Funcionarios: fun
        Real: sal_min
Início
        Abra (Cadastro, Leitura)
        Abra (Bem_pagos, Escrita)
        Escreva("Digite o valor do salário-mínimo atual:")
        Leia (sal_min)
        Leia (Cadastro, fun)
        Enquanto ( fun.nome <> "Fim_de_arquivo"")faça
                Se (fun.salario > 40*sal_min) então
                        Escreva (Bem_pagos, fun)
                Fim_se
                Leia (Cadastro, fun)
        Fim_enquanto
        Feche (Cadastro)
        Feche (Bem_pagos)
Fim
```

2) Faça um algoritmo para gerar um arquivo, composto de registros com nome, sexo e salário.

```
Algoritmo Gera_arquivo
        Registro = Dados
                Literal[40]: nome
                Literal[2]: sexo
                Real: salario
        Fim_registro
        Arquivo: Cadastro Sequencial Dados
        Dados: p1
        Literal[2]: p2
Início
```

```
        Abra (Cadastro, Escrita)
        p2 ← "s"
        Enquanto (p2 = "s")faça
                Escreva ("Digite um nome:")
                Leia (p1.nome)
                Escreva ("Digite o sexo (M ou F):")
                Leia (p1.sexo)
                Escreva ("Digite o salário:")
                Leia (p1.salario)
                Escreva (Cadastro, p1)
                Escreva ("Digita mais dados (s/n)?")
                Leia (p2)
        Fim_enquanto
        Feche (Cadastro)
Fim
```

3) Leia os seguintes dados de um arquivo relativos a cada funcionário de uma empresa: nome, salário, sexo e número de dependentes. Separe em dois outros arquivos homens e mulheres, apenas com nomes e salários reajustados, os quais deverão ser da seguinte forma:

- funcionários que ganham menos de R$1000,00 e têm mais de 5 dependentes terão reajuste de 20%;
- os que ganham menos de R$1000,00 e têm de 1 a 5 dependentes terão reajuste de 15%;
- os que ganham R$1000,00 ou mais e têm dependentes terão reajuste de 10%;
- as mulheres terão 1% a mais para cada dependente.

```
Algoritmo Salário
    Registro = Dados
            Literal[40]: nome
            Real: salario
            Literal[2]: sexo
            Inteiro: depende
    Fim_registro
    Registro = Reajuste
            Literal[40]: nome1
            Real: salario1
    Fim_registro
    Arquivo: Geral Sequencial Dados
    Arquivo: Homens Sequencial Reajuste
    Arquivo: Mulheres Sequencial Reajuste
    Dados: x
    Reajuste: y
Início
        Abra (Geral, Leitura)
        Abra (Homens, Escrita)
        Abra (Mulheres, Escrita)
        Leia (Geral, x)
        Enquanto (x.nome <> "Fim de Arquivo") faça
                Se (x.depende > 0) então
                        y.nome1 ← x.nome
                        Se (x.sexo="F" .E. x.salario < 1000.0) então
                                Se (x.depende > 5) então
```

```
                                        y.salario1←x.salario*(1.20+0.01*x.depende)
                                    senão
                                        y.salario1 ← x.salario*(1.15+0.01*x.depende)
                                    Fim_se
                                Fim_se
                                Se (x.sexo=”M” .E. x.salario <1000) então
                                    Se (x.depende > 5) então
                                        y.salario1 ← x.salario*1.20
                                    senão
                                        y.salario1 ← x.salario*1.15
                                    Fim_se
                                Fim_se
                                Se (x.salario >= 1000.00) então
                                    Se (x.sexo = “F”) então
                                        y.salario1 ← x.salario*(1.10+0.01*x.depende)
                                    senão
                                        y.salario1 ← x.salario*1.10
                                    Fim_se
                                Fim_se
                                Se (x.sexo = “M”) então
                                    Escreva (Homens, y)
                                Senão
                                    Escreva (Mulheres, y)
                                Fim_se
                            Fim_se
                            Leia (Geral, x)
                        Fim_enquanto
                        Feche (Geral)
                        Feche (Homens)
                        Feche (Mulheres)
Fim
```

10.4 – EXERCÍCIOS PROPOSTOS

1) Faça um subalgoritmo para gerar um arquivo de dados com as seguintes informações de um grupo de pessoas: nome, idade, sexo e estado civil. Faça um algoritmo principal para ler os dados deste arquivo e imprimir em outro arquivo uma lista com nome e idade dos homens solteiros ou divorciados, com idades entre 18 e 40 anos, e imprimir em outro arquivo uma lista com nome e idade das mulheres solteiras ou divorciadas, com idades entre 15 e 35 anos.

2) A revendedora Autocar dispõe de um arquivo com os seguintes dados de todos os seus automóveis: marca, modelo, ano de fabricação e preço. Faça um algoritmo para ler estes dados e imprimir na tela do computador a lista desses itens num dado ano ou numa dada marca.