Lógica de Programação

Matrizes, Arquivos, Procedimentos e Funções

Diretor Executivo
DAVID LIRA STEPHEN BARROS
Gerente Editorial
CRISTIANE SILVEIRA CESAR DE OLIVEIRA
Projeto Gráfico
TIAGO DA ROCHA
Autoria
IZABELLY MORAIS DE MORAIS
LEANDRO C. CARDOSO
MAX ANDRÉ DE AZEVÊDO SILVA

AUTORIA

Izabelly Morais de Morais

Sou licenciada em Ciência da Computação pela Universidade Federal da Paraíba (UFPB), e mestre em Ciência da Computação com ênfase em Engenharia de Software e Linguagens de Programação pela Universidade Federal de Pernambuco (UFPE). Leciono como professora formadora no Instituto Federal de Pernambuco (IFPE) e na Faculdade Pitágoras (João Pessoa, na Paraíba), onde tenho a oportunidade de transmitir minha experiência na área de Tecnologia e Educação. Por isso fui convidada pela Editora Telesapiens a integrar seu elenco de autores independentes. Estou muito feliz em poder ajudar você nesta fase de muito estudo e trabalho. Conte comigo!

Leandro C. Cardoso

Sou graduado em Comunicação Social com habilitação em Design Digital, e mestre em Tecnologias da Inteligência e Design Digital pela Pontifícia Universidade Católica de São Paulo (PUC-SP), com mais de 20 anos de experiência em direção de arte e criação. Passei por empresas como a Laureate International Universities – FMU/Fiam-Faam, a Universidade Anhembi Morumbi e o Centro Paula Souza (Fatec-Etec). Já atuei como analista de desenvolvimento pedagógico sênior, coordenador de curso técnico de Design Gráfico e revisor técnico e validador para curso EAD para clientes Laureate International Universities, DeVry Brasil, Unef, FAESF, Faculdade Positivo, Uninter e Platos Soluções Educacionais S.A. (Krotonn – Universidade Anhanguera). Além disso, sou autor de mais de 24 livros didáticos e um dos organizadores da Maratona de Criação e Design do Curso de Comunicação Visual da Etec Albert Einstein. Sou apaixonado pelo que faço e adoro transmitir minha experiência de vida àqueles que estão iniciando em suas profissões. Por isso fui convidado pela Editora Telesapiens a integrar seu elenco de autores independentes. Estou muito feliz em poder ajudar você nesta fase de muito estudo e trabalho. Conte comigo!

Max André de Azevêdo Silva

Sou formado em Ciência da Computação, pela Universidade Federal da Paraíba (UFPB), e mestre em Ciência da Computação com ênfase em Engenharia de Software pela Universidade Federal da Paraíba (UFPB). Tenho experiência na área de Desenvolvimento de Sistemas Web, Mobile e Jogos Eletrônicos. Atualmente, trabalho como analista de sistemas. Por isso fui convidado pela Editora Telesapiens a integrar seu elenco de autores independentes. Estou muito feliz em poder ajudar você nesta fase de muito estudo e trabalho. Conte comigo!

ICONOGRÁFICOS

Olá. Esses ícones irão aparecer em sua trilha de aprendizagem toda vez que:

OBJETIVO:
para o início do desenvolvimento de uma nova competência;

NOTA:
quando forem necessários observações ou complementações para o seu conhecimento;

EXPLICANDO MELHOR:
algo precisa ser melhor explicado ou detalhado;

SAIBA MAIS:
textos, referências bibliográficas e links para aprofundamento do seu conhecimento;

ACESSE:
se for preciso acessar um ou mais sites para fazer download, assistir vídeos, ler textos, ouvir podcast;

ATIVIDADES:
quando alguma atividade de autoaprendizagem for aplicada;

DEFINIÇÃO:
houver necessidade de se apresentar um novo conceito;

IMPORTANTE:
as observações escritas tiveram que ser priorizadas para você;

VOCÊ SABIA?
curiosidades e indagações lúdicas sobre o tema em estudo, se forem necessárias;

REFLITA:
se houver a necessidade de chamar a atenção sobre algo a ser refletido ou discutido sobre;

RESUMINDO:
quando for preciso se fazer um resumo acumulativo das últimas abordagens;

TESTANDO:
quando o desenvolvimento de uma competência for concluído e questões forem explicadas;

SUMÁRIO

Matrizes **10**

Conceito de Matriz 10

Problemas Envolvendo Matrizes 14

Arquivos de Dados **18**

Conceito de Arquivos de Dados 18

Instruções de Manipulação de Arquivos de Dados 22

Procedimentos **29**

Conceito de Procedimentos 29

Passagem de Parâmetros 33

Funções **36**

Conceito de Função 36

Declaração de uma Função 38

UNIDADE 04

INTRODUÇÃO

Você sabia que a área que trabalha com matrizes e modulação de algoritmos, como procedimentos e funções, é considerada uma área avançada da lógica de programação? Isso mesmo. Para tanto, é necessário ampliarmos o conceito de vetores para matrizes, bem como estruturas de dados heterogêneas, como arquivos sequenciais. Esses conhecimentos facilitam o desenvolvimento de aplicações comerciais mais corriqueiras, além de modular programas por meio de procedimentos e funções externas. Entendeu? Ao longo desta unidade letiva você vai mergulhar neste universo!

OBJETIVOS

Olá. Seja muito bem-vindo à Unidade 4. Nosso objetivo é auxiliar você no desenvolvimento das seguintes competências profissionais até o término desta etapa de estudos:

1. Definir o conceito e as aplicações de matrizes em soluções algorítmicas.
2. Trabalhar com arquivos de dados na algoritmização das soluções comerciais mais corriqueiras.
3. Modular algoritmos por meio de procedimentos.
4. Criar e utilizar funções externas à linguagem para solucionar problemas específicos.

Matrizes

OBJETIVO:

Ao término deste Capítulo você será capaz de entender o conceito e as aplicações de matrizes em soluções algorítmicas. Isso será fundamental para o exercício de sua profissão. E então? Motivado para desenvolver esta competência? Então vamos lá. Avante!.

Conceito de Matriz

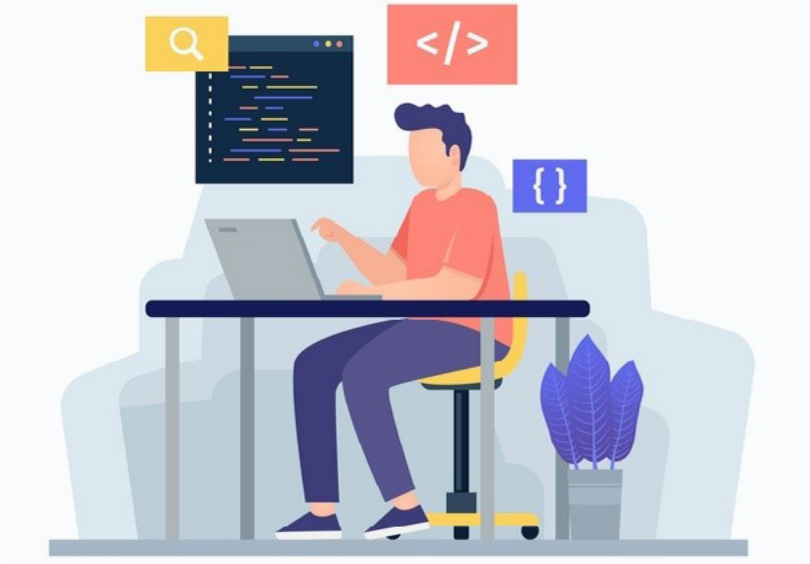
A Matemática nos ensinou que uma matriz é um agrupamento de números organizados por linhas e colunas. Em função disso, muito nos foi ensinado, como determinantes, diagonais, matriz transposta, entre outras operações algébricas.

Figura 1 – Na Matemática, matriz é um agrupamento de números organizados por linhas e colunas

Fonte: Freepik

Na Ciência da Computação, o conceito de matriz não é muito diferente, a diferença é que, além de números, todo e qualquer tipo de dado poderá ser armazenado em suas células.

Figura 2 – Na Ciência da Computação, o conceito de matrizes vai além de números, todo e qualquer tipo de dado poderá ser armazenado em suas células



Fonte: Freepik

Assim como nas planilhas eletrônicas, em cada uma dessas células podem ser armazenadas informações, porém, no caso das matrizes aplicáveis a algoritmos, esses dados precisam ter um único tipo. Essa característica torna as matrizes variáveis bidimensionais homogêneas.

DEFINIÇÃO:

Matrizes são variáveis de memória heterogêneas capazes de armazenar mais de um dado simultaneamente, indexando-os quanto ao seu posicionamento relativo em sua estrutura por meio de duas dimensões: linha e coluna, por isso são conhecidas como "variáveis bidimensionais"..

Assim, por exemplo, se quisermos armazenar as notas que um determinado aluno obteve ao longo de um ano letivo, juntamente com seus respectivos pesos para efeito de cálculo da média ponderada, teremos o que é apresentado na Tabela 1 (MORAIS; AZEVEDO, 2017c).

Tabela 1 – Exemplo de matriz contendo notas obtidas por um aluno (linha 1) e seus respectivos pesos (linha 2)

	1	2	3	4	5	6	7	8
1	10,0	9,5	6,5	7,0	8,5	4,0	10,0	6,5
2	1	1	1	2	1	1	1	2

Fonte: Morais e Azevedo (2017c).

No entanto, qual é a utilidade prática de uma matriz como essa? Assim como o uso de um vetor economiza a declaração de variáveis de memória, a utilização de uma matriz pode economizar a declaração de dois vetores, um para armazenar as notas e o outro para armazenar os pesos. Desse modo, vamos imaginar que o nosso algoritmo deva coletar as notas e os pesos de um aluno e, depois, calcular a média ponderada que esse aluno conseguiu ao final do último período. Você sabe o que é uma média ponderada? Essa grandeza matemática é bastante utilizada para obter uma média em que alguns números devem ter mais importância que outros. Assim, no caso das notas obtidas pelo aluno, sempre as notas finais devem valer o dobro das anteriores, por isso precisamos aplicar a média ponderada em vez da média aritmética.

$$\text{MÉDIA} = \frac{\text{NOTA1} * \text{PESO1} + \cdots + \text{NOTAn} * \text{PESOn}}{\text{PESO1} + \cdots + \text{PESOn}}$$

Na sintaxe da declaração de matrizes, o VisuAlg chama matrizes de "vetores", com a diferença de adicionar mais uma dimensão para a armazenagem de dados. Em termos práticos, uma matriz de L linhas e C colunas é declarada da seguinte forma em VisuAlg:

<nome da variável>: VETOR [1..L, 1..C] DE <tipo da variável>

Exemplos:

```
VAR
NOTASEPESOS: VETOR [1..2, 1..8] DE REAL
```

Assim como no caso dos vetores, uma vez declarada, uma matriz pode ser utilizada livremente, da mesma maneira que uma variável simples, tendo, porém, o cuidado de sempre se referir a um de seus elementos, e não a ela como um todo.

IMPORTANTE:

Sendo uma variável homogênea, a matriz não pode ter elementos de diferentes tipos. Em outras palavras, uma vez declarada como numérica inteira, todos os seus elementos só poderão receber valores numéricos inteiros. No entanto, em algumas linguagens de programação, como o Visual Basic, as matrizes são tratadas como variáveis heterogêneas, podendo receber dados de diferentes tipos para seus elementos.

Assim, para atribuirmos a nota 8,5 e o peso 1,0 aos elementos localizados na quinta coluna e na primeira e segunda linhas, respectivamente, da matriz NOTASEPESOS, teremos o seguinte (MORAIS; AZEVEDO, 2017c):

```
NOTASEPESOS[1,5] ← 8.5
NOTASEPESOS[2,5] ←1.0
```

Problemas Envolvendo Matrizes

Retomando o nosso exemplo anterior, vamos incrementar o algoritmo aplicando a média ponderada no lugar da aritmética? Para isso, precisamos definir a matriz NOTASEPESOS para armazenar as oito notas na primeira linha e os oito pesos das respectivas notas na segunda linha. Observe atentamente a lógica utilizada e tente identificar pontos de melhoria neste algoritmo proposto. Veja a seguir o algoritmo para o cálculo de média ponderada usando matriz em vez de vetores (MORAIS; AZEVEDO, 2017c).

```
ALGORITMO
VAR
IND: INTEIRO
MED, PESO: REAL
NOTASEPESOS: VETOR [1..2, 1..8] DE REAL
PARA IND DE 1 ATÉ 8 FAÇA
        LEIA NOTASEPESOS [1, IND]
        LEIA NOTASEPESOS [2, IND]
        MED ← MED + NOTASEPESOS [1, IND] * NOTASEPESOS [2, IND]
        PESO ← PESO + NOTASEPESOS [2, IND]
FIMPARA
ESCREVA "A média ponderada é: ", MED/PESO
FIMALGORITMO
```

Vamos fazer pequenas adaptações para testar no VisuAlg, que resultará o apresentado na Figura 3.

Figura 3 – Exemplo 1 em VisuAlg

```
Área dos programas ( Edição do código fonte ) -> Nome do arquivo: [AULA 13 - EXEMPLO 1.ALG]
 1 Algoritmo "Aula 13 - Exemplo 1"
 2 VAR
 3 IND: INTEIRO
 4 MED, PESO: REAL
 5 NOTASEPESOS: VETOR [1..2, 1..8] DE REAL
 6 INICIO
 7 PARA IND DE 1 ATÉ 8 FAÇA
 8       ESCREVA ("Digite a nota ", IND, ": ")
 9       LEIA (NOTASEPESOS [1, IND])
10       ESCREVA ("Digite o peso da nota ", IND, ": ")
11       LEIA (NOTASEPESOS [2, IND])
12       MED <- MED + NOTASEPESOS [1, IND] * NOTASEPESOS [2, IND]
13       PESO <- PESO + NOTASEPESOS [2, IND]
14 FIMPARA
15 ESCREVA ("A média ponderada é: ", MED/PESO)
16 FimAlgoritmo
```

Fonte: Reprodução do software VisuAlg.

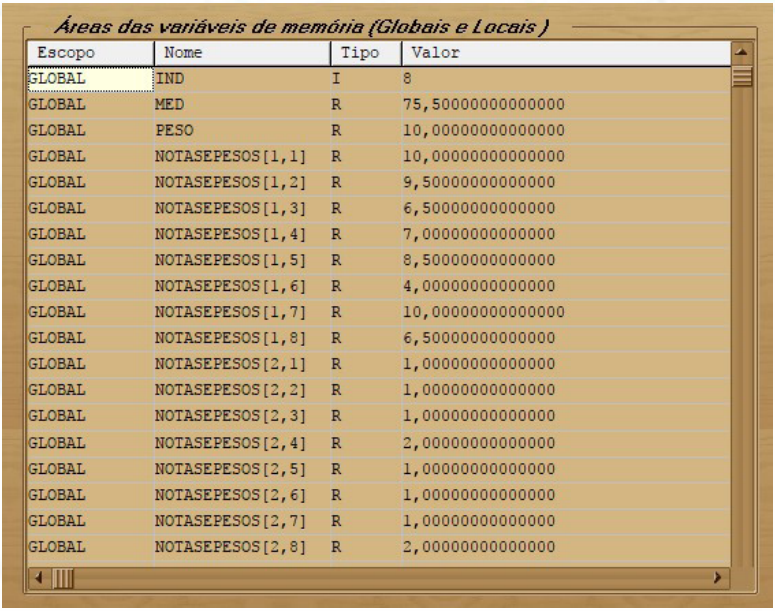
Executando o algoritmo no VisuAlg, obteremos o resultado ilustrado na Figura 4. Como podemos observar, o laço que implementamos pedirá a nota e o peso da nota a cada looping. Note, ainda, que utilizamos duas variáveis acumuladoras, uma para a soma dos produtos entre as notas e seus respectivos pesos, e a outra somente para acumular a soma dos pesos. No final do laço, a média pôde ser obtida dividindo o acumulador MED pelo PESO.

Figura 4 – Resultado do teste

Fonte: Reprodução do software VisuAlg.

Para visualizar o conteúdo da matriz que acabamos de alimentar com esse algoritmo, basta olhar para o painel localizado no canto superior direito da tela do VisuAlg. Lá você poderá conferir o valor inserido em cada uma das células.

Figura 5 – Painel de monitoramento das variáveis de memória



Escopo	Nome	Tipo	Valor
GLOBAL	IND	I	8
GLOBAL	MED	R	75,50000000000000
GLOBAL	PESO	R	10,00000000000000
GLOBAL	NOTASEPESOS[1,1]	R	10,00000000000000
GLOBAL	NOTASEPESOS[1,2]	R	9,50000000000000
GLOBAL	NOTASEPESOS[1,3]	R	6,50000000000000
GLOBAL	NOTASEPESOS[1,4]	R	7,00000000000000
GLOBAL	NOTASEPESOS[1,5]	R	8,50000000000000
GLOBAL	NOTASEPESOS[1,6]	R	4,00000000000000
GLOBAL	NOTASEPESOS[1,7]	R	10,00000000000000
GLOBAL	NOTASEPESOS[1,8]	R	6,50000000000000
GLOBAL	NOTASEPESOS[2,1]	R	1,00000000000000
GLOBAL	NOTASEPESOS[2,2]	R	1,00000000000000
GLOBAL	NOTASEPESOS[2,3]	R	1,00000000000000
GLOBAL	NOTASEPESOS[2,4]	R	2,00000000000000
GLOBAL	NOTASEPESOS[2,5]	R	1,00000000000000
GLOBAL	NOTASEPESOS[2,6]	R	1,00000000000000
GLOBAL	NOTASEPESOS[2,7]	R	1,00000000000000
GLOBAL	NOTASEPESOS[2,8]	R	2,00000000000000

Fonte: Reprodução do software VisuAlg.

Os dois últimos exemplos que algoritmizamos utilizaram variáveis compostas (vetores e matrizes) como meio de armazenar vários dados sem perdê-los. Porém, a cada digitação de dados para os elementos dessas variáveis, nós acumulamos seus valores para, somente depois, calcularmos a média. Poderíamos não ter usado vetores e matrizes para resolver os problemas anteriores? A resposta é sim, uma vez que não recorremos, posteriormente, aos conteúdos dos elementos do vetor ou da matriz (MORAIS; AZEVEDO, 2017c).

SAIBA MAIS:

Confira o material a seguir para um aprofundamento em vetores e matrizes. Para acessar, clique aqui.

No entanto, para processar os dados digitados após o término do laço, vetores e matrizes são, sim, importantes, pois sem eles os dados digitados serão atualizados a cada novo looping.

RESUMINDO:

E então? Gostou do que lhe mostramos? Aprendeu mesmo tudinho? Agora, só para termos certeza de que você realmente entendeu o tema de estudo deste Capítulo, vamos resumir tudo o que vimos. Você deve ter aprendido que é importante saber manipular dados em vetores, de modo a entender como funcionam as matrizes de maneira mais fácil. Diferentemente de um vetor, que só armazena dados em uma única direção, as matrizes são capazes de organizar esses dados em duas dimensões: linhas e colunas. Para isso, é importante a forma de manipulação quanto à atribuição de valores e envolvimento em operações de toda natureza, que é praticamente a mesma empregada nos vetores.

Arquivos de Dados

OBJETIVO:

Ao término deste Capítulo você será capaz de trabalhar com arquivos de dados na algoritmização das soluções comerciais mais corriqueiras. Isso será fundamental para o exercício de sua profissão. E então? Motivado para desenvolver esta competência? Então vamos lá. Avante!.

Conceito de Arquivos de Dados

Os arquivos de dados podem ser entendidos como uma evolução das matrizes. Três características diferem umas das outras, a primeira é que enquanto as matrizes são declaradas com tamanho fixo, sendo-lhes atribuída uma quantidade finita de linhas e colunas, os arquivos de dados não têm um tamanho previamente definido, o que os torna elásticos (MORAIS; AZEVEDO, 2017a).

Figura 6 – Os arquivos de dados não têm um tamanho previamente definido, sendo elásticos

Fonte: Freepik

A segunda característica é que enquanto as matrizes são estruturas de dados homogêneas, os arquivos de dados são estruturas heterogêneas, ou seja, podem armazenar dados de diferentes tipos.

DEFINIÇÃO:

Arquivos de dados são estruturas heterogêneas, capazes de armazenar um número ilimitado de dados, que são organizados por registros e campos.

A terceira e última característica é que as matrizes são estruturas de dados voláteis, armazenadas na memória RAM, enquanto os arquivos residem na memória auxiliar.

Figura 7 – As matrizes são estruturas de dados voláteis, armazenadas na memória RAM

Fonte: Freepik

A forma de organização dos arquivos de dados é semelhante à das matrizes, ou seja, os dados são organizados por registros (equivalentes às linhas da matriz) e campos de dados (equivalentes às colunas da matriz). Diferentemente das matrizes, os dados armazenados em um arquivo não são referenciados por linha e coluna, mas sim pelo nome do campo de

dado que o contém. Esse campo de dado se repete em todos os registros do arquivo, e o dado acessado sempre deve ser referenciado pelo nome do campo que diz respeito ao registro corrente (MORAIS; AZEVEDO, 2017a).

IMPORTANTE:

Ao acessar o telefone de um aluno em um arquivo de dados intitulado ALUNOS, primeiramente, o arquivo deve ter todos os seus registros lidos até chegar ao aluno desejado. Feito isso, o telefone daquele aluno estará contido no campo de dado de nome TELEFONE, por exemplo. Se o próximo registro for lido, o conteúdo do campo de dado TELEFONE não mais será o do aluno anterior, mas sim o do aluno corrente.

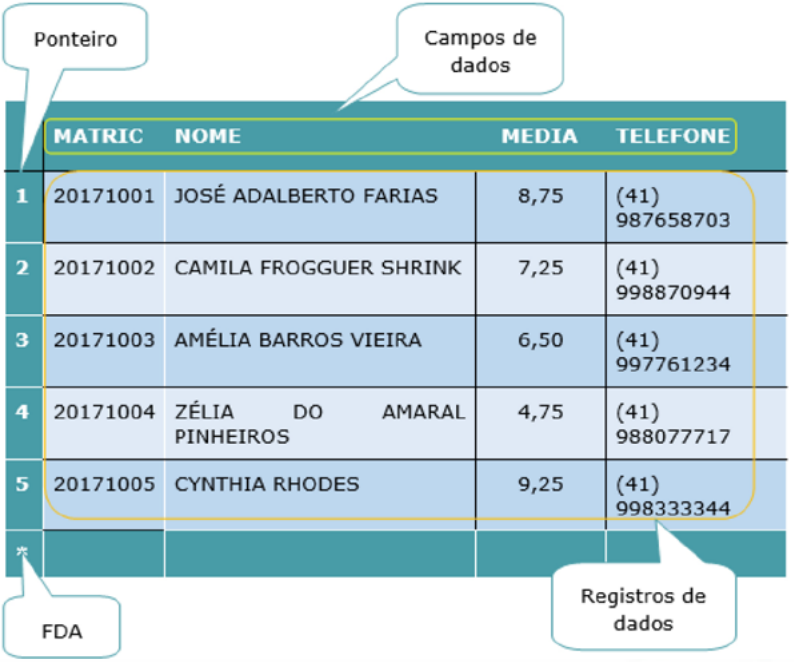
Pelo que explicamos até o momento, percebemos o surgimento de algumas novas definições, como a de arquivo de dados e a de registros de dados, que são as linhas de um arquivo de dados, os quais devem ser lidos de forma sequencial até que um determinado item seja localizado.

IMPORTANTE:

Apenas um registro de um mesmo arquivo pode ser lido de cada vez.

Já sobre campos de dados, estes são variáveis que assumem automaticamente os dados do último registro lido de um arquivo de dados. A Tabela 2 ilustra um exemplo de arquivo de dados, com a indicação das definições que acabamos de enunciar.

Tabela 2 – Exemplo de um arquivo de dados



	MATRIC	NOME	MEDIA	TELEFONE
1	20171001	JOSÉ ADALBERTO FARIAS	8,75	(41) 987658703
2	20171002	CAMILA FROGGUER SHRINK	7,25	(41) 998870944
3	20171003	AMÉLIA BARROS VIEIRA	6,50	(41) 997761234
4	20171004	ZÉLIA DO AMARAL PINHEIROS	4,75	(41) 988077717
5	20171005	CYNTHIA RHODES	9,25	(41) 998333344
*				

Fonte: Morais e Azevedo (2017a)

Para entender melhor, "ponteiro" é um termo usado para se referir ao registro corrente, ou seja, ao último registro lido do arquivo de dados. Em outras palavras, a instrução de leitura de um registro é, na realidade, um comando para o ponteiro se dirigir ao próximo registro. Já FDA é a sigla normalmente utilizada para se referir ao "fim do arquivo". Trata-se de um registro imaginário localizado logo após o último registro gravado. Para efeito do algoritmo, essa sigla consiste em uma função lógica interna do interpretador, que será FALSO se o registro corrente for um registro válido, e VERDADEIRO se o registro corrente se tratar da marca de fim de arquivo.

Instruções de Manipulação de Arquivos de Dados

Agora que já entendemos o que é e para que serve um arquivo de dados, inclusive conhecendo a sua estrutura lógica de organização, vamos enunciar alguns comandos para declaração, criação, atualização e leitura de arquivos de dados.

IMPORTANTE:

O VisuAlg não dá suporte a arquivos de dados, pelo menos até a versão 3, portanto, as instruções e as estruturas de dados abordadas aqui não poderão ser testadas nele. É importante sempre estar atento às atualizações das aplicações.

Nas instruções de manipulação de arquivos de dados é importante saber sobre a abertura de arquivos. Em um algoritmo, antes de utilizar um arquivo de dados, faz-se necessário abri-lo. Para isso, podemos utilizar o comando ABRIR.

```
ABRIR <nome do arquivo> ALIAS <apelido do arquivo>
        CAMPOS { <campo-1>, <tipo de dado do campo-1>;
                 <campo-2>, <tipo de dado do campo-2>; ...
                 <campo-N>, <tipo de dado do campo-N> }
```

Assim, antes de utilizar dados do arquivo de alunos que exemplificamos anteriormente, precisamos abri-lo logo no início do algoritmo, como mostra o exemplo a seguir.

```
ALGORITMO
ABRIR "ALUNOS.dat" ALIAS ALUNOS
        CAMPOS {MATRIC, CARACTERE;
                NOME, CARACTERE;
                MEDIA, REAL;
                TELEFONE; CARACTERE}
FIMALGORITMO
```

Note que a cláusula ALIAS permite que seja atribuído um apelido ao arquivo que se deseja abrir. Com isso, não há necessidade de nos referirmos ao nome completo do arquivo, com extensão e endereço da pasta em que está localizado no disco rígido do computador.

SAIBA MAIS:

O VisuAlg não dá suporte a operações com arquivos de dados, no entanto, ele consegue ler e gravar arquivos-textos. Conheça estes recursos no link a seguir. Para acessar, clique aqui.

Depois, a cláusula CAMPOS permite que sejam indicados, entre chaves, todos os campos e respectivos tipos que compõem um registro desse arquivo.

IMPORTANTE:

A ordem com que os campos são declarados é muito importante, pois se houver alguma inversão, os dados serão trocados de um campo para outro quando um registro desse arquivo for lido.

Em relação à criação de arquivos, caso o arquivo que se deseja manipular não exista, ele pode ser criado pelo comando CRIAR, utilizando a seguinte sintaxe:

```
CRIAR <nome do arquivo> ALIAS <apelido do arquivo>
        CAMPOS {<campo-1>, <tipo de dado do campo-1>;
                <campo-2>, <tipo de dado do campo-2>; ...
                <campo-N>, <tipo de dado do campo-N> }
```

Perceba que a sintaxe é a mesma do comando ABRIR. Nesse caso, uma vez criado um arquivo, ele estará automaticamente aberto e disponível para acesso. Veja o algoritmo que cria um arquivo:

```
ALGORITMOCRIAR "ALUNOS.dat" ALIAS ALUNOS
                    CAMPOS {MATRIC, CARACTERE;
                            NOME, CARACTERE;
                            MEDIA, REAL;
                            TELEFONE; CARACTERE}
FIMALGORITMO
```

Sobre a leitura de registros, como dissemos anteriormente, ler o registro de um arquivo é simplesmente posicionar o ponteiro de leitura para cima de um deles. Ao abrir um arquivo, o ponteiro estará posicionado no primeiro registro, assim, os seguintes comandos auxiliam no posicionamento do ponteiro de leitura, a saber:

```
PULE <número de registros a serem saltados> DE <alias do arquivo>

PULE PARA PRIMEIRO DE <alias do arquivo>

PULE PARA ÚLTIMO DE <alias do arquivo>
```

Para compreendermos melhor esse comando, vejamos o exemplo de algoritmo a seguir, que abre, lê e lista o conteúdo de um arquivo até o fim.

```
ALGORITMO
        ABRIR "ALUNOS.dat" ALIAS ALUNOS
                CAMPOS {MATRIC, CARACTERE;
                                NOME, CARACTERE;
                                MEDIA, REAL;
                                TELEFONE; CARACTERE}
        ENQUANTO NÃO FDA("ALUNOS") FAÇA
                ESCREVA NOME, MEDIA
                PULE 1 DE ALUNOS
        FIMENQUANTO
FIMALGORITMO
```

Observe que o laço ENQUANTO FAÇA estará sendo processado enquanto não for atingido o final do arquivo, em outras palavras, enquanto a função lógica FDA() não for verdadeira, o laço estará em looping (MORAIS; AZEVEDO, 2017a).

IMPORTANTE:

A função FDA() é interna da linguagem, e deve ser utilizada em estruturas condicionais e/ou repetitivas para testar se já chegou ao final do arquivo.

Em relação à gravação de registros, quer o arquivo esteja vazio, ou seja, FDA() é verdadeira logo ao abrir o arquivo; quer tenha sido atingido o final do arquivo, poderemos incluir um novo registro. O comando que permite esta instrução é o GRAVE.

```
GRAVE EM <alias do arquivo>
```

Este comando fará com que seja adicionado um novo registro ao arquivo descrito na cláusula <alias do arquivo>. Os dados que serão gravados nesse registro serão aqueles contidos ou atribuídos aos seus campos de dados (MORAIS; AZEVEDO, 2017a). Por exemplo: se acabamos de ler um determinado registro e, de repente, instruímos que seja executado o comando GRAVE, os dados do último registro lido serão replicados naquele que está sendo incluído no arquivo. Para que isso não ocorra, antes de executar o comando GRAVE, devem ser movidos novos

dados aos campos de dados declarados no algoritmo, como mostra o exemplo a seguir, de algoritmo que inclui um registro no arquivo ALUNOS.

```
ALGORITMO
        ABRIR "ALUNOS.dat" ALIAS ALUNOS
                        CAMPOS {MATRIC, CARACTERE;
                                NOME, CARACTERE;
                                MEDIA, REAL;
                                TELEFONE; CARACTERE}
        LEIA MATRIC, NOME, MEDIA, TELEFONE
        ENQUANTO MATRIC <> "99999999" FAÇA
                GRAVE EM ALUNOS
                LEIA MATRIC, NOME, MEDIA, TELEFONE
        FIMENQUANTO
FIMALGORITMO
```

No exemplo acima, começamos por abrir o arquivo "ALUNOS.dat", apelidado de ALUNOS. Executada essa ação, iniciamos a digitação dos dados de um novo aluno nos campos MATRIC, NOME, MEDIA e TELEFONE. Até aqui esses dados estavam apenas na memória, até que o comando GRAVE foi executado, ordenando que todos os dados atribuídos àqueles campos fossem gravados. Sobre a regravação de registros, vimos que o comando GRAVE adiciona um novo registro ao arquivo. Contudo, se quisermos simplesmente alterar os dados de um registro, adotamos a sintaxe a seguir:

```
REGRAVE EM <alias do arquivo>
```

O comando REGRAVE transfere os dados contidos nos campos armazenados em memória para o registro corrente. Vamos analisar como isso ocorre no algoritmo que consulta e altera os dados de um registro do arquivo ALUNOS:

```
ALGORITMO
        ABRIR "ALUNOS.dat" ALIAS ALUNOS
                        CAMPOS {MATRIC, CARACTERE;
                                                NOME, CARACTERE;
                                                MEDIA, REAL;
                                                TELEFONE; CARACTERE}
        VAR MAT: CARACTERE
        LEIA MAT
        ENQUANTO MAT <> MATRIC E NÃO FDA("ALUNOS") FAÇA
                PULE 1 DE ALUNOS
        FIMENQUANTO
        SE FDA("ALUNOS") ENTÃO
                ESCREVA "Matrícula de aluno não encontrada"
        SENÃO
                ESCREVA "Digite os novos dados para este aluno"
                LEIA NOME, MEDIA, TELEFONE
                REGRAVE EM ALUNOS
        FIMSE
FIMALGORITMO
```

O algoritmo que acabamos de apresentar é capaz de alterar os dados de um determinado aluno, para isso, note que o programa pede que seja digitada a matrícula a ser pesquisada no arquivo. Por meio de uma estrutura repetitiva ENQUANTO FAÇA, os registros do arquivo ALUNOS começam a ser "varridos" enquanto não for localizada a matrícula procurada e não chegue ao final do arquivo. Ao ser localizado o registro procurado, perceba que novos dados serão digitados pelo usuário e, em seguida, regravados no registro corrente. Em relação à deleção de registros, para completar as funcionalidades básicas das operações algorítmicas envolvendo arquivos de dados, só falta conhecermos o comando DELETE.

```
DELETE DE <alias do arquivo>
```

Esse comando fará com que o registro corrente seja excluído do arquivo. Em algumas linguagens de programação, essa exclusão é apenas lógica, ou seja, os dados permanecem gravados no arquivo e podem ser restaurados com comandos específicos.

```
ALGORITMO
        ABRIR "ALUNOS.dat" ALIAS ALUNOS
                        CAMPOS (MATRIC, CARACTERE;
                                                NOME, CARACTERE;
                                                MEDIA, REAL;
                                                TELEFONE; CARACTERE)
        VAR MAT: CARACTERE
        LEIA MAT
        ENQUANTO MAT <> MATRIC E NÃO FDA("ALUNOS") FAÇA
                PULE 1 DE ALUNOS
        FIMENQUANTO
        SE FDA("ALUNOS") ENTÃO
                ESCREVA "Matrícula de aluno não encontrada"
        SENÃO
                DELETE DE ALUNOS
        FIMSE
FIMALGORITMO
```

O algoritmo acima é um exemplo típico de exclusão de registros, ou seja, algoritmo que consulta e deleta um registro do arquivo ALUNOS.

RESUMINDO:

E então? Gostou do que lhe mostramos? Aprendeu mesmo tudinho? Agora, só para termos certeza de que você realmente entendeu o tema de estudo deste Capítulo, vamos resumir tudo o que vimos. Você deve ter aprendido que para, finalmente, chegarmos ao ponto de desenvolvermos algoritmos bem próximos à realidade que teremos no mundo do trabalho, são necessários os conhecimentos estudados aqui. Praticamente, todas as aplicações comerciais consistem na leitura e na atualização de arquivos de dados. Afinal, é lá que as informações são armazenadas em caráter definitivo, uma vez que variáveis de memória, vetores e matrizes são apagadas quando a execução de um programa é encerrada. Assim, é importante saber como podemos processar dados e gerar informações a partir do meio físico em que estarão armazenados.

Procedimentos

OBJETIVO:

Ao término deste Capítulo você será capaz de modular algoritmos por meio de procedimentos. Isso será fundamental para o exercício de sua profissão. E então? Motivado para desenvolver esta competência? Então vamos lá. Avante!.

Conceito de Procedimentos

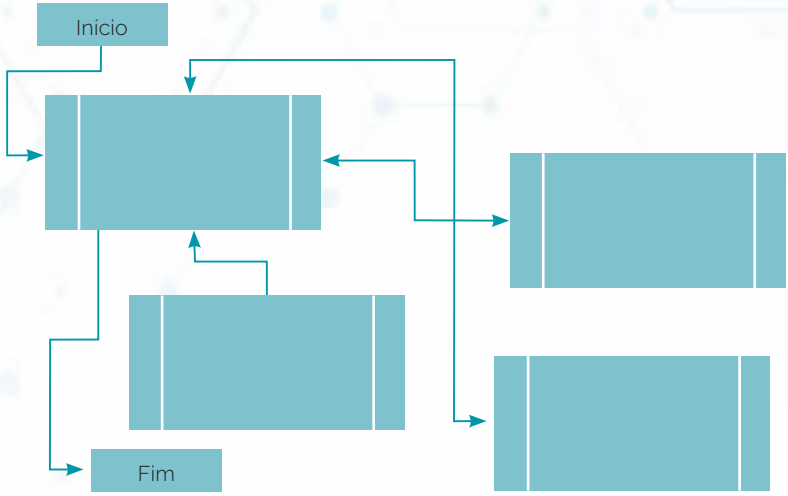
Você já teve a sensação de estar codificando várias vezes as mesmas instruções, para fazer as mesmas coisas, em diferentes algoritmos? E se pudéssemos substituir essas instruções redundantes por uma única instrução? É isso que chamamos de procedimentos, também denominados sub-rotinas (MORAIS; AZEVEDO, 2017d).

DEFINIÇÃO:

Procedimentos ou sub-rotinas são conjuntos de instruções que podem ser invocados a partir de pontos distintos de um algoritmo.

Os procedimentos nem sempre são utilizados apenas para reduzir o nível de redundância das instruções em um programa. Eles também podem servir como subprogramas, a exemplo do que ocorre com sistemas cadastrais, como o ilustrado na Figura 8 (MORAIS; AZEVEDO, 2017d).

Figura 8 – Diagrama de um sistema cadastral



Fonte: Morais e Azevedo (2017d).

No exemplo ilustrado acima, temos um programa-mestre intitulado "Menu". É esperado que esse programa ofereça a opção de o usuário digitar um número para selecionar a ação desejada, que pode ser:

1. Cadastramento.
2. Listagem.
3. Consulta.
4. FIM.

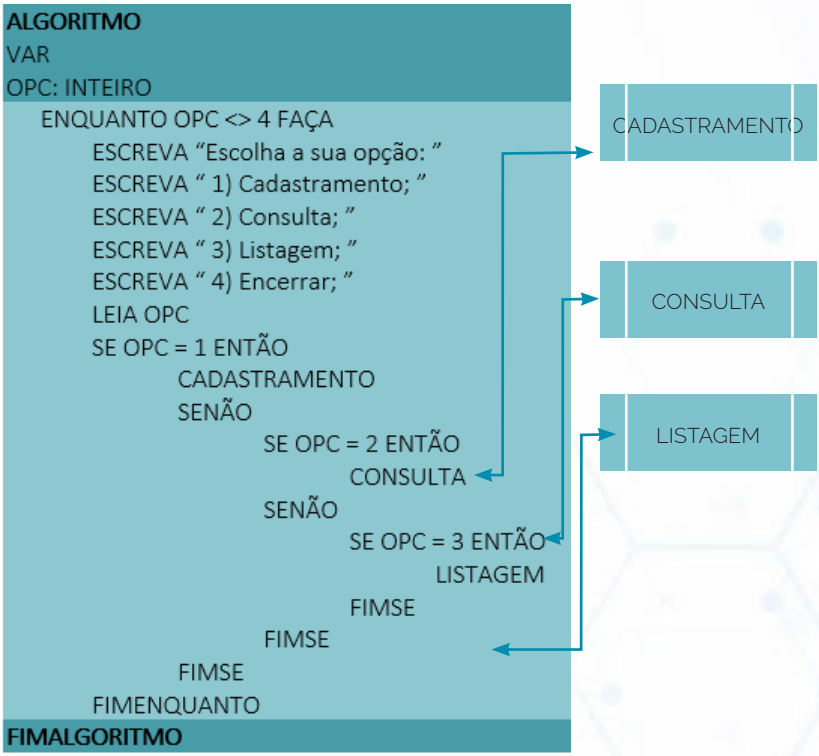
Dependendo da opção que esse usuário digitar, um entre três procedimentos pode ser executado. Nesse caso, tais procedimentos funcionarão como subprogramas, ou seja, o fluxo de execução das linhas de comando do programa "Menu" será desviado para um desses subprogramas, somente retornando quando este chegar ao seu término (MORAIS; AZEVEDO, 2017d) .

REFLITA:

Mas, o que aconteceria se não tivéssemos o recurso dos procedimentos à nossa disposição?

O programa "Menu" ficaria excessivamente longo, dificultando a sua compreensão. E o que é pior: toda vez que tivéssemos a necessidade de executar as instruções de uma das opções digitadas, teríamos que replicar aquelas mesmas instruções em outra parte do programa.

Para exemplificar melhor o uso dos procedimentos, vamos visualizar como ficaria o algoritmo "Menu" no contexto deste pequeno sistema. Veja o exemplo do algoritmo para seleção de procedimentos.



Perceba que, a cada opção digitada, um procedimento é chamado à execução. Após o seu fluxo lógico de instruções ser executado, esse

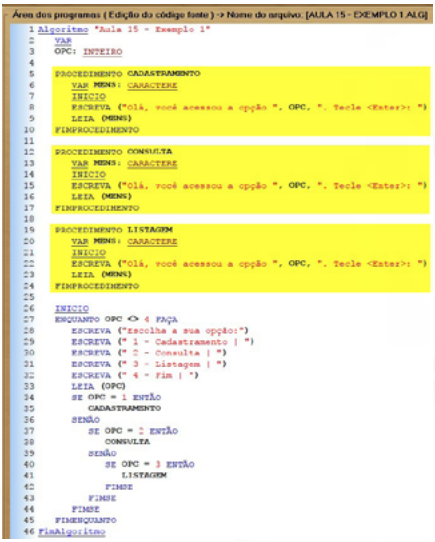
procedimento devolverá o controle de execução ao programa-mestre que o invocou. A linha de comando, imediatamente, após a chamada desse procedimento será executada, o que, nesse caso, fará com que uma nova opção seja requisitada, até que um "4" seja digitado para finalizar o programa.

IMPORTANTE:

Diferentemente de muitas linguagens, o VisuAlg exige que os procedimentos sejam declarados e codificados dentro do próprio programa que os invoca, logo no início do algoritmo, após a declaração das variáveis e antes da cláusula INÍCIO. Para facilitar nossos testes, não vamos algoritmizar os três procedimentos agora, conforme seus objetivos ensejam. Em vez disso, vamos escrever uma breve mensagem, só para sinalizar que eles realmente estão sendo executados corretamente. Mais adiante você pode completar suas codificações.

Vamos adaptar este algoritmo para o formato do VisuAlg. O resultado será mostrado na Figura 9.

Figura 9 – Algoritmo-exemplo em VisuAlg



Fonte: Reprodução do software VisuAlg.

Os três procedimentos chamados pelo algoritmo estão salientados em fundo amarelo na Figura 9, perceba que cada um deles declara uma variável de nome MENS. Essa variável é privada, ou seja, só tem validade para efeito do procedimento que a declara, e isso significa que ela não pode ser tratada pelo algoritmo principal. A recíproca não é verdadeira, ou seja, a variável OPC, declarada e alimentada no algoritmo principal, foi tratada por cada um dos procedimentos (MORAIS; AZEVEDO, 2017d).

Figura 10 – Monitoramento do teste do algoritmo

Fonte: Reprodução do software VisuAlg.

Nesse caso, dizemos que a variável OPC é uma variável global, isto é, tem validade em todos os procedimentos que, porventura, forem criados pelo programa principal.

Passagem de Parâmetros

Além de poderem ser chamados à execução a partir de qualquer ponto de um algoritmo, os procedimentos também podem receber instruções para aplicarem um determinado processamento sobre alguns dados repassados pelo programa principal. Esses dados repassados são chamados de "parâmetros".

DEFINIÇÃO:

Os parâmetros de um procedimento são dados repassados a ele pelo programa principal, que ocupam variáveis especificamente declaradas para este fim.

Para exemplificarmos o uso de parâmetros em um procedimento, vamos retomar o programa-exemplo anteriormente ilustrado. Imagine

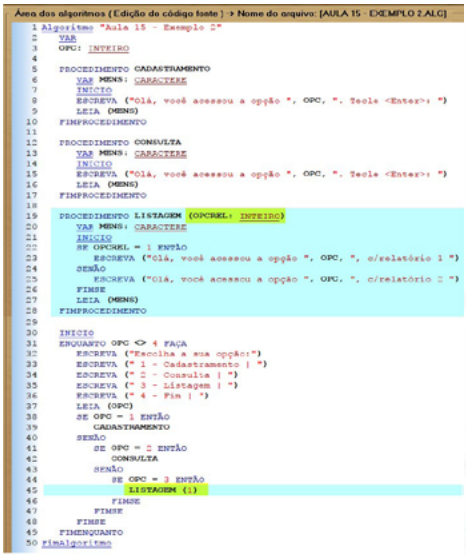
que o procedimento LISTAGEM apresente duas opções de relatório, e que, dependendo do parâmetro repassado pelo programa-principal, o relatório poderá ser um ou outro. Primeiramente, vamos codificar essa solução considerando que a opção será pelo primeiro relatório. Nesse caso, perceba que o algoritmo a seguir está repassando ao procedimento LISTAGEM um parâmetro numérico igual a 1.

SAIBA MAIS:

Acesse a documentação do VisuAlg no que diz respeito à sintaxe da chamada e da declaração de um procedimento. Para acessar, clique aqui.

Como você pode observar, esse parâmetro está ocupando uma variável de memória intitulada OPCREL, que está sendo tratada pelo comando SE, presente nesse procedimento.

Figura 11 – O mesmo algoritmo, porém, passando parâmetros para o procedimento LISTAGEM



Fonte: Reprodução do software VisuAlg.

Observe que o parâmetro passado ao procedimento LISTAGEM foi um número inteiro, exatamente como declarado internamente dentro do procedimento LISTAGEM. Se tivéssemos passado uma constante literal, como "1" (entre aspas), teria ocorrido um erro de execução do programa.

Figura 12 – Teste de execução do programa modificado, passando parâmetros ao procedimento LISTAGEM

Fonte: Reprodução do software VisuAlg.

Da mesma forma que um parâmetro pode ser repassado em forma de constante, este também pode ser uma variável de memória. Veja o resultado da execução desse algoritmo no VisuAlg, conforme Figura 12.

RESUMINDO:

E então? Gostou do que lhe mostramos? Aprendeu mesmo tudinho? Agora, só para termos certeza de que você realmente entendeu o tema de estudo deste Capítulo, vamos resumir tudo o que vimos. Você deve ter aprendido que mesmo que nos preparemos para solucionar diversos problemas com algoritmos, percebemos que alguns blocos de comandos sempre se repetem de um algoritmo para outro. Em particular, quando criamos uma série de programas de um mesmo sistema, podemos identificar, claramente, essas sequências redundantes de comandos. Para simplificar e encurtar algoritmos extensos e complexos, podemos "lançar mão" de procedimentos. É o que chamamos de "modulação de algoritmos".

Funções

OBJETIVO:

Ao término deste Capítulo você será capaz de criar e utilizar funções externas à linguagem para solucionar problemas específicos. Isso será fundamental para o exercício de sua profissão. E então? Motivado para desenvolver esta competência? Então vamos lá. Avante!.

Conceito de Função

Já que aprendemos como funcionam os procedimentos (ou sub-rotinas), ficou mais fácil entendermos o que vem a ser função. Podemos afirmar que se trata do mesmo princípio, mas com uma diferença básica: os procedimentos não recebem parâmetros, e não retornam nada, já as funções retornam algo em função dos parâmetros recebidos.

DEFINIÇÃO:

As funções são conjuntos de instruções que podem ser invocadas a partir de pontos distintos de um algoritmo, cuja finalidade é retornar um valor como resultado do processamento dos parâmetros recebidos por elas..

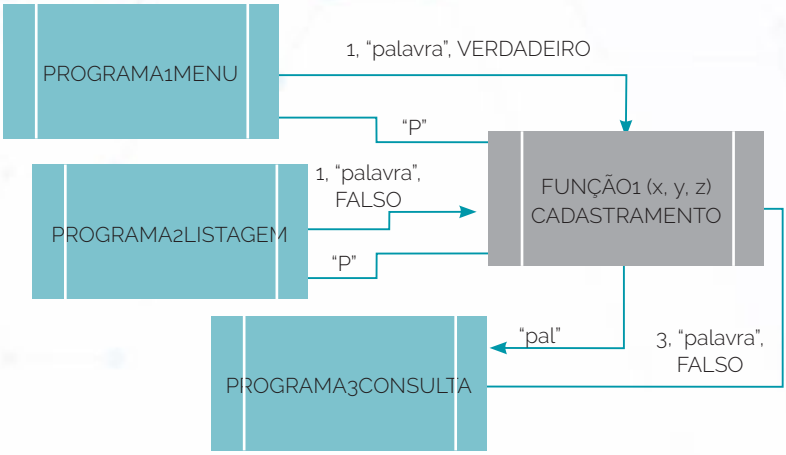
Para exemplificarmos, vamos imaginar que é necessário criar uma função que receba três parâmetros (também chamados de "argumentos"):

1. Um número.
2. Uma cadeia de caracteres.
3. Um sinal lógico (VERDADEIRO ou FALSO).

Claro que essas constantes também podem ser variáveis. A critério do programador, o primeiro argumento, o número, deve significar a quantidade de caracteres que deve ser extraída de uma cadeia de caracteres, informado como segundo parâmetro. Por fim, se o terceiro argumento for verdadeiro, essa parte da cadeia extraída deverá vir em

maiúsculo, e, se for falso, em minúsculo. A Figura 13 traz alguns exemplos de parâmetros que entram na função, bem como os seus respectivos retornos.

Figura 13 – Diagrama de um sistema cadastral



Fonte: Morais e Azevedo (2017b).

No exemplo ilustrado acima, temos vários programas que chamam uma mesma função, passando diferentes dados como parâmetros e obtendo resultados distintos como retorno. Mas, como esses parâmetros são passados? Como o retorno é comunicado? Veja o exemplo a seguir.

```
ALGORITMO "PROGRAMA1"
VAR PALAV: CARACTERE
TAMANHO, OPC: INTEIRO
ESCREVA "Digite uma palavra, a quantidade de caracteres da esquerda para a direita A extrair, e o número 1 se quiser ver em maiúsculo, ou 2 se minúsculo;"
LEIA PALAV, TAMANHO, OPC
SE OPC = 1 ENTÃO
        ESCREVA FUNÇÃO1(TAMANHO, PALAV, VERDADEIRO)
SENÃO
        ESCREVA FUNÇÃO1(TAMANHO, PALAV, FALSO)
FIMSE
FIMALGORITMO
```

Perceba que o algoritmo do exemplo 1 inseriu a função diretamente dentro da sintaxe do comando ESCREVA, o que significa que essa função retorna uma cadeia de caracteres que pode ser impressa diretamente por esse comando.

IMPORTANTE:

Se estivéssemos lidando com um procedimento, nada disso poderia ser feito, uma vez que uma sub-rotina não retorna nada.

Se assim o quiséssemos, poderíamos, em vez de escrever, atribuir o resultado da FUNÇÃO1 diretamente a uma variável, ou inseri-la em um comando ou em uma estrutura condicional.

Declaração de uma Função

Assim como os procedimentos devem ser declarados logo no início do algoritmo, também as funções assim o devem. Em VisuAlg, por exemplo, as funções devem ser declaradas e codificadas dentro do próprio programa que as invoca, logo no início do algoritmo, após a declaração das variáveis e antes da cláusula INÍCIO. A sintaxe da cláusula de declaração de função é a seguinte:

```
FUNCAO <nome-de-função> [(<sequência-de-declarações-de-parâmetros>)]: <tipo-de-dado>
// Seção de Declarações Internas
INICIO
// Seção de Comandos
FIMFUNCAO
```

O <nome-de-função> obedece às mesmas regras de nomenclatura das variáveis, a <sequência-de-declarações-de-parâmetros> é uma sequência de cláusulas VAR, opcional, com a seguinte sintaxe:

```
[VAR] <sequência-de-parâmetros>: <tipo-de-dado>
```

As variáveis que compõem os parâmetros a serem recebidos devem ser separadas por ponto e vírgula, a presença (opcional) da palavra-chave VAR indica passagem de parâmetros por referência; caso contrário, a

passagem será por valor. Por sua vez, <sequência-de-parâmetros> é uma sequência de nomes de parâmetros separados por vírgulas. O valor retornado pela função será do tipo especificado na sua declaração (logo após os dois pontos). Em alguma parte da função (de modo geral, no seu final), esse valor deve ser retornado por meio do comando RETORNE. De modo análogo ao programa principal, a seção de declarações internas começa com a palavra-chave VAR, e continua com a seguinte sintaxe (MORAIS; AZEVEDO, 2017b):

```
<lista-de-variáveis> : <tipo-de-dado>
```

Vamos a um exemplo: que tal o mesmo que ilustramos na Figura 13?

```
FUNCAO FUNCAO1 (X: INTEIRO, Y: CARACTERE, Z: LOGICO): INTEIRO
INICIO
SE Z ENTÃO
        RETORNE MAIUSC (COPIA(Y; 1; X))
SENÃO
        RETORNE MINUSC (COPIA(Y; 1; X))
FIMSE
FIMFUNCAO
```

No programa principal, teríamos as seguintes adequações para comportar a função FUNCAO1:

```
ALGORITMO "PROGRAMA1"
VAR PALAV: CARACTERE
TAMANHO, OPC: INTEIRO
FUNCAO FUNCAO1 (X: INTEIRO, Y: CARACTERE, Z: LOGICO): INTEIRO
INICIO
SE Z ENTÃO
        RETORNE MAIUSC(COPIA(Y; 1; X))
SENÃO
        RETORNE MINUSC(COPIA(Y; 1; X))
FIMSE
FIMFUNCAO
INICIO
ESCREVA "Digite uma palavra, a quantidade de caracteres da esquerda para a direita A extrair,
e o número 1 quiser ver em maiúsculo, ou 2 se minúsculo: "
LEIA PALAV, TAMANHO, OPC
SE OPC = 1 ENTÃO
        ESCREVA FUNÇÃO1(TAMANHO, PALAV, VERDADEIRO)
SENÃO
        ESCREVA FUNÇÃO1(TAMANHO, PALAV, FALSO)
FIMSE
FIMALGORITMO
```

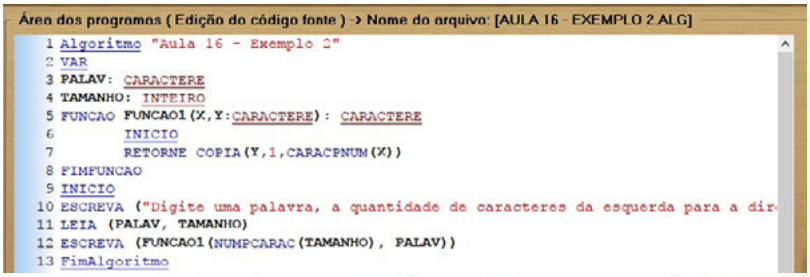
Vamos adaptar esse algoritmo para a sintaxe do VisuAlg.

IMPORTANTE:

Para testar esse algoritmo no VisuAlg, teremos que modificar um pouco a lógica, pois o VisuAlg não aceita parâmetros de diferentes tipos em uma função. Então, para fazê-lo funcionar, o algoritmo da função FUNCAO1 irá receber um número no formato alfanumérico, uma palavra (assim como já estava antes), não mais um terceiro parâmetro.

Adaptando o código em VisuAlg, teremos o seguinte:

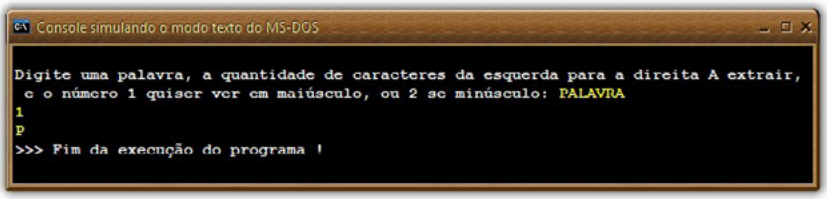
Figura 14 – Exemplo adaptado para as limitações do VisuAlg



Fonte: Reprodução do software VisuAlg.

Este será o resultado:

Figura 15 – Resultado da execução do exemplo anterior



Fonte: Reprodução do software VisuAlg.

RESUMINDO:

E então? Gostou do que lhe mostramos? Aprendeu mesmo tudinho? Agora, só para termos certeza de que você realmente entendeu o tema de estudo deste Capítulo, vamos resumir tudo o que vimos. Você deve ter aprendido que pode utilizar várias funções, inúmeras vezes, para solucionar os mais diversos tipos de problemas. Pensando nisso, ao aparecer um novo problema para o qual não haja uma função específica como solução, a maioria das linguagens de programação oferece o recurso de criação de funções externas à linguagem, também chamadas de "funções internas ao programa", o termo "interna" ou "externa", portanto, depende do referencial.

REFERÊNCIAS

BROOKSHEAR, G. J. **Ciência da Computação**: Uma Visão Abrangente. Porto Alegre: Bookman, 2013.

MORAIS, I. S.; AZEVEDO, M. **Lógica de programação**: arquivos de dados. Recife: Unissau, 2017a.

MORAIS, I. S.; AZEVEDO, M. **Lógica de programação:** funções. Recife: Unissau, 2017b.

MORAIS, I. S.; AZEVEDO, M. **Lógica de programação**: matrizes. Recife: Unissau, 2017c.

MORAIS, I. S.; AZEVEDO, M. **Lógica de programação**: procedimentos. Recife: Unissau, 2017d.

SEBESTA, R. W. **Conceitos de linguagens de programação.** São Paulo: Bookman, 2011.