

LÓGICA PARA REDES DE COMPUTADORES

**CAPÍTULO 4 - COMO CONSTRUIR E
MANIPULAR ESTRUTURAS DE
DADOS, HOMOGÊNEAS E
HETEROLOGÊNEAS, COM A
LINGUAGEM DE PROGRAMAÇÃO C?**

Ana Paula da Costa Cardoso

Introdução

Quando você calcula a maior nota ou a menor altura, não fica uma pergunta no ar? Eu calculei a maior, a menor, a média, a soma, etc., mas de quais números? Não é viável criar muitas variáveis para armazenar todos estes números. Imagine que você tenha desenvolvido um sistema para uma pizzaria, no qual o usuário pode lançar os pedidos e depois fechar a conta do cliente. Vamos imaginar a seguinte situação: o sistema fecha a venda e você a apresenta ao cliente. Mas, o cliente deseja saber quais foram os itens que ele consumiu. Não é verdade?

Você já deve ter se perguntado: como faço para armazenar temporariamente meus dados? Existem estruturas que permitem que eu consiga manipular meus dados, depois que o usuário digitou? Como faço para ler uma sequência de nomes de clientes e depois imprimi-los em ordem alfabética? A resposta para estas perguntas é: sim, existe.

Nas linguagens de programação, existem estruturas que podem ser consideradas como um conjunto de variáveis. Esse conjunto de variáveis se organiza na forma de vetores ou matrizes. Isso mesmo! São muito parecidos com os vetores e matrizes da matemática. Agora, podemos ler uma sequência de dados, armazená-los temporariamente, manipulá-los e, depois, podemos imprimi-los para o usuário.

As estruturas de vetores e matrizes são de armazenamento temporário, como quaisquer outras variáveis declaradas na linguagem C. Não são estruturas de armazenamento permanente, ou seja, se sair do programa, perdem-se os dados.

A criação e manipulação de vetores e matrizes são conceitos e habilidades extremamente importantes, quando se trata de programação. Neste capítulo, iremos estudar o que são os vetores e as matrizes e como manipulá-los, ou seja, como armazenar, alterar, apagar e imprimir dados neles.

Estude com dedicação! Boa leitura!

4.1 Estruturas de dados unidimensionais – vetores com a linguagem de programação C

A primeira estrutura que vamos estudar é a do vetor, ou também conhecida como *array* unidimensional. Um vetor é um conjunto de variáveis, de mesmo tipo e com o mesmo nome (DEITEL; DEITEL, 2011). Falando assim, soa meio estranho, né? Então, imagine um vetor como se fosse um conjunto de vários vagões ligados entre si e todos sendo puxados pela mesma máquina, sendo que cada vagão é uma variável. Pois bem, este é o princípio de um vetor.

Vamos ver mais a fundo? Acompanhe a seguir.

4.1.1 Introdução

tender o conceito de vetor, ou *array unidimensional*, como um conjunto de

de mesmo tipo e mesmo nome, por meio de um exemplo. Considere quatro ia ao lado da outra, enumeradas. Podemos enumerá-las de diversas formas, por exemplo, começando com dez e indo de um em um, então teríamos a caixa 10, caixa 11, caixa 12 até caixa 14, correto?

Ou podemos também, enumerá-las a partir do zero, então teríamos as caixas de 0 a 3, certo?

Agora, vamos dar um nome para todas as caixas, por exemplo, *idade*. Espera um pouco, se todas têm o mesmo nome, como vou diferenciá-las? Simples: pela numeração. Observe a figura a seguir. São quatro caixas enumeradas de zero a três. O nome das caixas é *idade*, então, para diferenciar cada caixa e, assim, alterar as idades dentro delas, basta fazer uma referência à numeração da caixa. A caixa *idade* número 1, possui a idade de 5 anos. E a caixa *idade* de número 0, possui qual idade dentro dela? Isso mesmo, ela guarda a idade de 12 anos. Agora ficou mais claro, né?



Figura 1 - Caixas simulando um vetor.

Fonte: Elaborada pela autora, 2019.

Alguns pontos devem ser destacados neste exemplo. Vamos ver quais são clicando na interação a seguir.

Todas as caixas são contíguas, ou seja, estão todas juntas, adjacentes umas às outras.

A numeração das caixas começou pelo valor zero, e cresce de forma sequencial, sem saltos ou pulos. Sendo assim, não teremos uma numeração com números faltando, ou fora da ordem ou decrescendo.

A numeração começou na esquerda e seguiu para a direita.

Agora que ficou mais claro o entendimento, vamos pegar este exemplo e trazê-lo para a

programação.

O vetor é um conjunto de variáveis, e como uma variável é uma posição de memória, então, o vetor é um conjunto de posições de memória que podem armazenar dados do mesmo tipo. Se temos um vetor do tipo inteiro, só poderemos armazenar valores do tipo inteiro.

A numeração faz referência a uma posição da memória, veja a figura a seguir. É um exemplo de um vetor, ou *array*, de 12 posições, ou seja, de 12 variáveis, com o nome de *c*. Como são 12, este *array* é enumerado de 0 a 11, assim, temos as posições que vão de 0 a 11. Cada variável é chamada também de elemento, então, temos 12 elementos. Observe o elemento *c[4]*, *c* é o nome do *array* e 4 é a posição que ele se encontra no vetor, ou seja, ele é o quinto elemento e seu valor é de 1543.

O número de posição do elemento dentro do array c	c [0]	O nome do array é c
c [1]	-45	
c [2]	6	
c [3]	0	
c [4]	72	
c [5]	1543	← Valor
c [6]	-89	
c [7]	0	
c [8]	62	
c [9]	-3	
c [10]	1	
c [11]	6453	
	78	

Figura 2 - Vetor de 12 posições.

Fonte: DEITEL; DEITEL, 2006, p. 252.

De forma resumida, podemos destacar alguns pontos. Veja clicando nos itens a seguir.

- Um vetor é um conjunto de variáveis, posições de memória contíguas.
- Possui um nome e armazena sempre o mesmo tipo de dados.
- Cada elemento, ou variável, possui um número que indica a sua posição na sequência de todas os elementos. Por exemplo, elemento 0, é a primeira variável.

- A numeração deve ser sequencial, com passo de 1, ou seja, a numeração irá de 1 em 1, começando por 0.
-

VOCÊ QUER VER?

O filme *Steve Jobs* (SORKIN, 2015), protagonizado por Michael Fassbender, retrata os momentos importantes da vida profissional e pessoal do fundador da Apple e criador do Macintosh e do iMac. Para quem está conhecendo o mundo da computação, este filme dá um contexto muito interessante sobre o cenário da indústria de informática.

Agora iremos estudar como a linguagem C estrutura um vetor.

4.1.2 Estrutura

Como sabemos que um vetor é um conjunto de variáveis, o princípio de declaração de vetores pela linguagem C, segue a mesma linha.

Para declarar um vetor em C:

tipo_dados nome_vetor [tamanho];

Veja a especificação de cada termo clicando nas abas a seguir.

tipo_dados

É o tipo de dados que serão armazenados no vetor, por exemplo, tipo int, float, char etc.

nome_vetor

É o nome dado ao vetor. Segue as mesmas regras de criação de nomes de variáveis simples.

tamanho

É a quantidade de elementos, ou seja, de variáveis que compõem o vetor.

Vamos ver alguns exemplos na próxima figura. Na linha 4, foram declarados dois vetores. O vetor *idade*, com 4 elementos enumerados de 0 a 3, e o vetor *c* com 12 elementos enumerados de 0 a 11, ambos do tipo inteiro. Na linha 5, temos um vetor *salário*, do tipo *float*, com 5 elementos. E, por último, um vetor do tipo caractere, com 10 elementos. Aqui cabe uma explicação. No vetor *nome*, não é possível armazenar 10 nomes, e sim armazenar um nome com até 10 caracteres. Para cada elemento, podemos guardar um caractere.

```
1 #include<stdio.h>
2 main()
3 {
4     int idade[4], c[12];
5     float salario[15];
6     char nome[10];
7 }
```

Figura 3 - Declaração de vetores no C.
Fonte: Elaborada pela autora, 2019.

Para acessar cada elemento, utilizamos o nome e a posição entre colchetes. Analise o exemplo a seguir, na próxima figura. Temos um vetor de 4 posições, com nome de *idade*, que pode armazenar dados do tipo inteiro, linha 4.

```
1 #include<stdio.h>
2 main()
3 {
4     int idade[4];
5     idade[3] = 4;
6     idade[1] = 15;
7     idade[0] = 6;
8     idade[2] = 10;
9 }
```

Figura 4 - Atribuindo valores a um vetor.

Fonte: Elaborada pela autora, 2019.

Na linha 5, figura anterior, o elemento de posição 3, ou seja, o quarto elemento, recebe o valor de 4. Na linha 6, o segundo elemento, posição 1, recebe o valor de 15. O preenchimento do vetor não precisa ser na ordem sequencial, pode ser alternado, como fizemos na figura anterior. Então, nosso vetor ficou assim:

6	15	10	4
idade[0]	idade[1]	idade[2]	idade[3]

Figura 5 - Vetor carregado.

Fonte: Elaborada pela autora, 2019.

Vamos analisar outro exemplo, na figura a seguir. Você consegue ver quais são os valores que serão impressos na tela para o usuário? Vamos entender este exemplo. Existe um vetor chamado *x*, com 4 elementos e do tipo inteiro. Na posição 0, primeiro elemento, foi guardado o valor de 12. Na posição 1, segundo elemento foi guardado o valor de *x[0]* mais 2. *x[0]* é o primeiro elemento e possui o valor de 12, então, $12+2=14$. O terceiro elemento, posição 2, recebe o valor de *x[1]*, que vale 14, mais 2, portanto, recebe o valor de 16. O último elemento, *x[3]*, recebe o valor de *x[2]*, que vale 16, mais 2. Então, *x[3]* recebe o valor de 18.

```
1 #include<stdio.h>
2 main()
3 {
4     int x[4];
5     x[0] = 12;
6     x[1] = x[0] + 2;
7     x[2] = x[1] + 2;
8     x[3] = x[2] + 2;
9     printf("Vetor x = [%d , %d , %d , %d ]", x[0], x[1], x[2], x[3]);
10 }
```

Figura 6 - Programa para carregar um vetor carregado e depois imprimi-lo.

Fonte: Elaborada pela autora, 2019.

Observe como fica a tela de execução do exemplo dado na figura anterior:

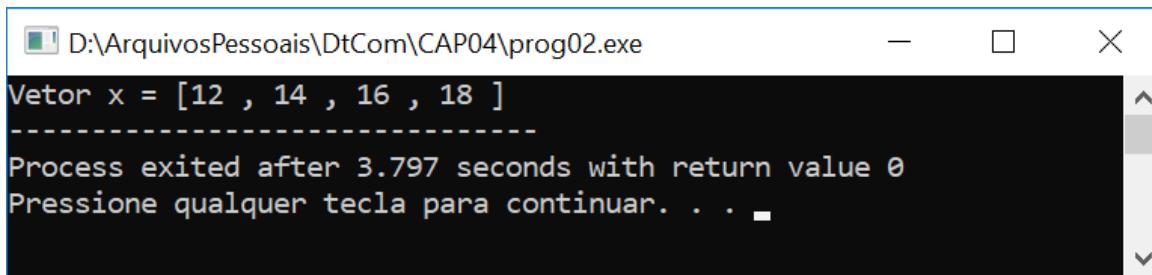


Figura 7 - Tela de execução do programa para carregar um vetor e imprimi-lo.

Fonte: Elaborada pela autora, 2019.

Observe que o vetor é manipulado da mesma forma que uma variável, apenas temos que informar a posição do elemento.

Que tal um exemplo, no qual o usuário digita valores para carregar o vetor? Analise a próxima figura. É um vetor chamado *i*, do tipo inteiro, com 4 elementos. O usuário carregou o vetor, no exemplo, com os números -5, 6, 8 e 2. A forma como o *printf* e o *scanf* trata de vetores é indiferente. Ou seja, dentro de um programa escrito em C, você vai trabalhar com vetores da mesma forma que, com variáveis simples, o único ponto é

informar o nome do vetor e a sua posição.

```
1 #include<stdio.h>
2 main()
3 {
4     int i[4];
5     printf("Digite 4 numeros\n");
6     scanf("%d %d %d %d", &i[0], &i[1], &i[2], &i[3]);
7     printf("Vetor x = [%d , %d , %d , %d ]", i[0], i[1], i[2], i[3]);
8 }
```

D:\ArquivosPessoais\DtCom\CAP04\prog03.exe

```
Digite 4 numeros
-5
6
8
2
Vetor x = [-5 , 6 , 8 , 2 ]
-----
Process exited after 12.49 seconds with return value 0
Pressione qualquer tecla para continuar. . .
```

Figura 8 - Programa onde o usuário carrega um vetor.

Fonte: Elaborada pela autora, 2019.

Como a posição do elemento dentro do vetor é um número inteiro, então podemos pegar uma variável e utilizá-la para informar qual a posição desejada. Analise a próxima figura.

É um vetor de 3 posições, com o nome de *vet* e do tipo inteiro. O usuário que digita os valores para preencher o vetor, linhas 5 e 6. Depois o usuário escolhe qual elemento deseja ver, linhas 7 e 8. A posição do elemento é armazenado na variável *pos*. Na linha 9, o programa manda imprimir na tela o valor do vetor na posição que está armazenada na variável *pos*. Por exemplo, se o usuário desejar ver qual o elemento está na posição 2, a variável *pos* receberá o valor de 2, e o *printf* imprimirá na tela, para o usuário, o valor de *vet[2]*. Neste exemplo, não foi testado se o valor que o usuário digitará para as posições, estarão entre 0 e 2, mas o correto é que se faça isto.

```
1 #include<stdio.h>
2 main()
3 {
4     int vet[3], pos;
5     printf("Digite 3 numeros\n");
6     scanf("%d %d %d", &vet[0], &vet[1], &vet[2]);
7     printf("Digite a posicao que deseja ver do vetor ");
8     scanf("%d", &pos);
9     printf("vetor[%d] = %d", pos, vet[pos]);
10 }
```

```
D:\ArquivosPessoais\DtCom\CAP04\prog04.exe
Digite 3 numeros
10
20
30
Digite a posicao que deseja ver do vetor 2
vetor[2] = 30
-----
Process exited after 21.1 seconds with return value 0
Pressione qualquer tecla para continuar. . .
```

Figura 9 - Programa no qual o usuário escolhe qual elemento será impresso.

Fonte: Elaborada pela autora, 2019.

E para inicializar um vetor na hora da declaração dele? É bem simples inicializar um vetor em C. Depois da declaração, coloque um sinal de igual seguido por uma lista de inicializadores, separados por vírgula, entre chaves:

```
int vet[4]={1, 3, 5, 9}
```

Ou pode-se inicializar com apenas um valor:

```
int vet[3] = {0}
```

Neste caso, este vetor foi inicializado com o valor de zero em todos os seus elementos. Observe o exemplo da figura a seguir. Foi criado uma constante chamada TAMANHO, com o valor de 5. O vetor *vet*, do tipo inteiro foi inicializado com os valores de 1, 2, 3, 4 e 5.

```
1 #include<stdio.h>
2 #define TAMANHO 5
3 main()
4 {
5     int vet[TAMANHO]={1,2,3,4,5};
6     printf("vetor = [%d, %d, %d, %d, %d]",vet[0],vet[1],vet[2],vet[3],vet[4]);
7 }
```

D:\ArquivosPessoais\DtCom\CAP04\prog05.exe

vetor = [1, 2, 3, 4, 5]

Process exited after 5.232 seconds with return value 0

Pressione qualquer tecla para continuar. . .

Figura 10 - Programa no qual o vetor é inicializado na declaração.

Fonte: Elaborada pela autora, 2019.

Escreva todos estes exemplos no editor de texto e rode os programas. Faça diversos testes. Não prossiga se este tópico não ficou claro, o entendimento até aqui é necessário para compreender o próximo tópico, que é a manipulação de vetores.
Continue acompanhando com atenção!

4.2 Estruturas de dados unidimensionais – vetores com a linguagem de programação C

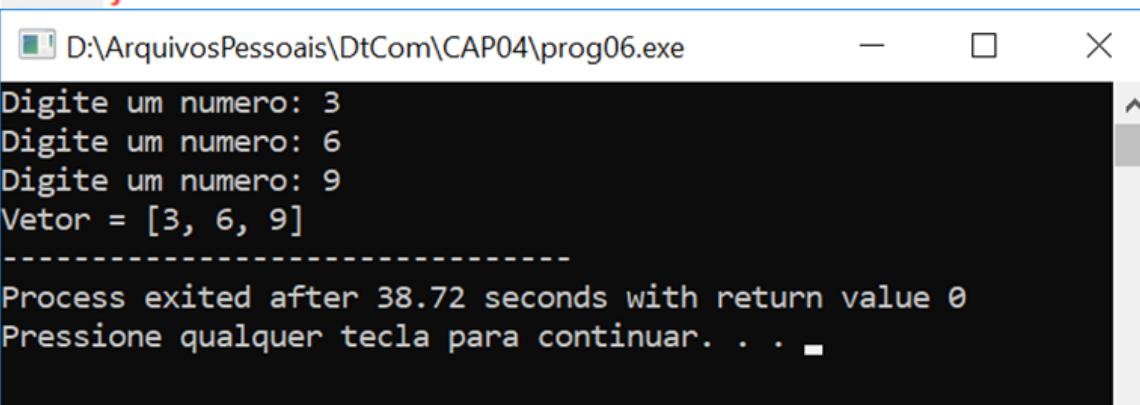
Já sabemos como são as estruturas dos vetores. Vimos também como declará-los, como armazenar valores neles, como imprimir para o usuário seus valores e como alterar seus dados. Assim, já sabemos como manipulá-los, mas não estamos tirando proveito do melhor destas estruturas. Por enquanto, estamos na mesma situação de utilizar variáveis simples.

Os vetores trabalham muito bem com as estruturas de repetição, na verdade são perfeitas pra elas. Neste item, vamos aprender a trabalhar com os vetores e as estruturas de repetição, fazendo com que os vetores sejam manipulados de forma mais otimizada.

4.2.1 Manipulação

Ficou claro como os vetores funcionam, certo? Mas, agora vamos otimizar a manipulação deles. Vamos analisar o próximo exemplo, figura a seguir. Observe as linhas de 6 a 11. Todos os *printf* são iguais e os *scanf* só alteram a posição do vetor, que começa com 0 e termina com 2.

```
1 #include<stdio.h>
2 #define TAMANHO 3
3 main()
4 {
5     int vet[TAMANHO];
6     printf("Digite um numero: ");
7     scanf("%d", &vet[0]);
8     printf("Digite um numero: ");
9     scanf("%d", &vet[1]);
10    printf("Digite um numero: ");
11    scanf("%d", &vet[2]);
12    printf("Vetor = [%d, %d, %d]", vet[0], vet[1], vet[2]);
13 }
```



```
D:\ArquivosPessoais\DtCom\CAP04\prog06.exe
Digite um numero: 3
Digite um numero: 6
Digite um numero: 9
Vetor = [3, 6, 9]
-----
Process exited after 38.72 seconds with return value 0
Pressione qualquer tecla para continuar. . .
```

Figura 11 - Programa para carregar vetor.

Fonte: Elaborada pela autora, 2019.

Agora, analise o próximo exemplo, na figura a seguir. Este programa faz a mesma coisa que o anterior, lê três números digitados pelo usuário e os armazena em um vetor. Vamos ver qual a diferença entre eles.

A mudança está nas linhas dos `scanf`, agora todas as linhas são iguais a `scanf("%d", &vet[i]);` e a variável `i` é quem está mudando de valor a cada leitura do `scanf`. Na linha 8, `i` possui o valor de 0, então o `scanf` passa a assumir a forma de `scanf("%d", &vet[0]);`. Na linha 11, como `i` já recebeu o valor de 1, o `scanf` assume `scanf("%d", &vet[1]);` e na linha 14, o `scanf` passa a ser `scanf("%d", &vet[2]);`. Percebeu porque os dois exemplos, o da figura anterior e a próxima, fazem a mesma coisa? Mas, qual a vantagem de uma ou da outra? Vamos analisar.

```
1 #include<stdio.h>
2 #define TAMANHO 3
3 main()
4 {
5     int vet[TAMANHO], i;
6     i = 0;
7     printf("Digite um numero: ");
8     scanf("%d", &vet[i]);
9     i = 1;
10    printf("Digite um numero: ");
11    scanf("%d", &vet[i]);
12    i = 2;
13    printf("Digite um numero: ");
14    scanf("%d", &vet[i]);
15    printf("Vetor = [%d, %d, %d]", vet[0], vet[1], vet[2]);
16 }
```

```
D:\ArquivosPessoais\DtCom\CAP04\prog06.exe
Digite um numero: 2
Digite um numero: 4
Digite um numero: 6
Vetor = [2, 4, 6]
-----
Process exited after 15.11 seconds with return value 0
Pressione qualquer tecla para continuar. . .
```

Figura 12 - Programa para carregar vetor.

Fonte: Elaborada pela autora, 2019.

Neste último exemplo, figura anterior, todas os pares de linhas *printf*/*scanf*, pares de linhas 7/8, 10/11 e 13/14, são iguais. O que muda é o valor da variável *i*, que vai de 0 a 2, de um em um. Então, se conseguirmos fazer a variável *i* alterar de 0 a 2, não precisaremos escrever todas as linhas, basta repeti-las. Qual o comando que você conhece que gera valores em uma variável de 0 a 2, de um em um? Isso mesmo! O comando de repetição.

VOCÊ SABIA?

Você sabia que dá para carregar um vetor automaticamente com valores aleatórios? As vezes você precisará de um vetor carregado com valores entre uma faixa específica e uma forma de resolver este problema é utilizar funções randômicas (BRAGIL, 2003). O C trabalha com estas funções *rand*, *random*, *srandom* entre outras, para gerar números aleatórios. Para saber mais sobre como utilizar estas funções, acesse

<<https://www.vivaolinux.com.br/dica/Gerando-numeros-aleatorios-em-C> (<https://www.vivaolinux.com.br/dica/Gerando-numeros-aleatorios-em-C>)>.

Veja como ficou o programa agora, na próxima figura.

Essa é a grande otimização no uso de vetores. Podemos carregar quantos valores forem necessários e depois manipulá-los. Neste exemplo, estamos carregando 3 números, mas poderíamos carregar 10, 100 ou qualquer outro valor, o programa seria o mesmo, a única mudança seria no valor da constante TAMANHO.

The figure shows a terminal window with the following content:

```
D:\ArquivosPessoais\DtCom\CAP04\prog07.exe
Digite um numero: 4
Digite um numero: 5
Digite um numero: 6
Vetor = 4, 5, 6,
-----
Process exited after 9.141 seconds with return value 0
Pressione qualquer tecla para continuar. . .
```

The terminal window title is "D:\ArquivosPessoais\DtCom\CAP04\prog07.exe". The program prompts the user to enter three numbers, which are then displayed as elements of an array named "Vetor". The process exits after 9.141 seconds.

Figura 13 - Programa para carregar vetor.

Fonte: Elaborada pela autora, 2019.

O próximo programa mostra como carregar dois vetores, somar os dois vetores e armazenar em um terceiro vetor e, depois, comparar os dois vetores procurando os elementos iguais nas mesmas posições. Próxima figura. Vamos analisá-lo linha por linha. Acompanhe com o programa, a tela de execução e a análise a seguir.

- Linha 2: define o valor da constante TAMANHO como 5, ela será

usada para definir o tamanho dos vetores A, B e C.

- Linha 5: declara os vetores A, B e C como inteiros, com 5 elementos cada e inicializa-os com 0.
- Linhas de 7 a 49: comando de repetição *do/while*. Teste de saída (*op != 4*), o que significa que, quando o usuário digitar o valor 4, sairá do laço.
- Linhas de 9 a 14: imprime as opções para o usuário.
- Linhas de 15 a 23: opção 1, carregar os dois vetores. Cada vetor é carregado individualmente, sendo primeiro o vetor A e depois o vetor B. No *for* para carregar os dois vetores, linhas 18 e 21, a condição é *i <= TAMANHO -1*. Isso ocorre porque se o vetor possui 5 elementos, então suas posições irão de 0 a 4, assim, a condição está testando *i <= 4*. A digitação dos números de cada vetor foi feita dando-se um espaço entre eles, em vez de um *enter*. Veja a seguir:

```

15
16 if(op == 1)
17 {
18     printf("Vetor A. Digite %d numeros: ", TAMANHO);
19     for(i = 0; i<=TAMANHO-1 ; i++)
20         scanf("%d", &A[i]);
21     printf("Vetor B. Digite %d numeros: ", TAMANHO);
22     for(i = 0; i<=TAMANHO-1 ; i++)
23         scanf("%d", &B[i]);
}
1 - Carregar vetores
2 - Somar vetores: A + B
3 - Comparar vetores
4 - Sair
Opcao: 1
Vetor A. Digite 5 numeros: 2 5 7 10 -3
Vetor B. Digite 5 numeros: 2 5 3 10 12

```

Figura 14 - Linhas 15 a 23.

Fonte: Elaborada pela autora, 2019.

- Linhas de 24 a 32: opção 2, somar o vetor A com o vetor B e

armazenar no vetor C. Observe que, para todos os vetores, os índices, ou seja, as posições possuem a mesma variável. Isto significa que o primeiro elemento de A será somado com o primeiro elemento de B e será armazenado na primeira posição de C. Para somar os dois vetores e depois armazenar no vetor C, linha 29, utiliza o mesmo princípio de variáveis simples. Por exemplo, se quiséssemos somar o vetor A com o vetor B e guardar o resultado no vetor A, a linha 29 ficaria assim: A[i] = A[i] + B[i];. Isso é o mesmo princípio dos acumuladores e somadores. Veja a seguir:

```
24 if(op == 2)
25 {
26     printf("C = ");
27     for(i = 0; i<=TAMANHO-1 ; i++)
28     {
29         C[i] = A[i] + B[i];
30         printf("%d ", C[i]);
31     }
32 }
```

```
1 - Carregar vetores
2 - Somar vetores: A + B
3 - Comparar vetores
4 - Sair
Opcão: 2
C = 4 10 10 20 9
```

Figura 15 - Linhas de 24 a 32.

Fonte: Elaborada pela autora, 2019.

- Linhas de 33 a 48: opção 3, comparar os dois vetores procurando por elementos iguais. As linhas 36, 37 e 38, são para imprimir o vetor A na tela e as linhas de 39 a 41, para imprimir o vetor B. Na linha 42, o comando *for* irá fazer com que se passe por todos os elementos de cada vetor e na linha 44, se faz o teste

da igualdade. Se o elemento, na posição 2, do vetor A for igual ao elemento, na posição 2, do vetor B, então incrementa a variável *cont*. Note que a variável *cont* foi zerada na linha 35. Isso torna-se necessário para não acumular os valores em *cont*. Por exemplo, passou a primeira vez, e os dois vetores tinham 3 elementos iguais nas mesmas posições. Se for passar a segunda vez e não zerar *cont*, o programa indicará que existem 6 elementos iguais, sendo que, na verdade, são apenas 3. Por isso, é necessário zerar *cont* antes da verificação. Veja a seguir:

The screenshot shows a terminal window with the following content:

```
33 if(op == 3)
34 {
35     cont = 0;
36     printf("A = ");
37     for(i = 0; i<=TAMANHO-1 ; i++)
38         printf("%d ", A[i]);
39     printf("\nB = ");
40     for(i = 0; i<=TAMANHO-1 ; i++)
41         printf("%d ", B[i]);
42     for(i = 0; i<=TAMANHO-1 ; i++)
43     {
44         if(A[i] == B[i])
45             cont++;
46     }
47     printf("\n%d elementos iguais\n", cont);
48 }
```

1 - Carregar vetores
2 - Somar vetores: A + B
3 - Comparar vetores
4 - Sair
Opção: 3
A = 2 5 7 10 -3
B = 2 5 3 10 12
3 elementos iguais

Figura 16 - Linhas 33 a 48.

Fonte: Elaborado pela autora, 2019.

- Os vetores foram zerados na hora da criação para evitar um erro, caso o usuário escolhesse a opção de somar dois vetores antes de carregá-los.

```

1 #include<stdio.h>
2 #define TAMANHO 5
3 main()
4 {
5     int A[TAMANHO]={0}, B[TAMANHO]={0}, C[TAMANHO]={0}, i;
6     int cont=0, op;
7     do
8     {
9         printf("\n1 - Carregar vetores\n");
10        printf("2 - Somar vetores: A + B\n");
11        printf("3 - Comparar vetores\n");
12        printf("4 - Sair\n");
13        printf("Opção: ");
14        scanf("%d", &op);
15        if(op == 1)
16        {
17            printf("Vetor A. Digite %d numeros: ", TAMANHO);
18            for(i = 0; i<TAMANHO-1 ; i++)
19            {
20                scanf("%d", &A[i]);
21            }
22            printf("Vetor B. Digite %d numeros: ", TAMANHO);
23            for(i = 0; i<TAMANHO-1 ; i++)
24            {
25                scanf("%d", &B[i]);
26            }
27        }
28        if(op == 2)
29        {
30            printf("C = ");
31            for(i = 0; i<TAMANHO-1 ; i++)
32            {
33                C[i] = A[i] + B[i];
34                printf("%d ", C[i]);
35            }
36        }
37        if(op == 3)
38        {
39            cont = 0;
40            printf("A = ");
41            for(i = 0; i<TAMANHO-1 ; i++)
42            {
43                printf("%d ", A[i]);
44            }
45            printf("\nB = ");
46            for(i = 0; i<TAMANHO-1 ; i++)
47            {
48                printf("%d ", B[i]);
49            }
50        }
51    }while(op != 4);
52 }

```

```

D:\ArquivosPessoais\DtCom\CAP04\prog08.exe
1 - Carregar vetores
2 - Somar vetores: A + B
3 - Comparar vetores
4 - Sair
Opção: 1
Vetor A. Digite 5 numeros: 2 5 7 10 -3
Vetor B. Digite 5 numeros: 2 5 3 10 12

1 - Carregar vetores
2 - Somar vetores: A + B
3 - Comparar vetores
4 - Sair
Opção: 2
C = 4 10 10 20 9
1 - Carregar vetores
2 - Somar vetores: A + B
3 - Comparar vetores
4 - Sair
Opção: 3
A = 2 5 7 10 -3
B = 2 5 3 10 12
3 elementos iguais

1 - Carregar vetores
2 - Somar vetores: A + B
3 - Comparar vetores
4 - Sair
Opção: 4

-----
Process exited after 88.44 seconds with return value 0
Pressione qualquer tecla para continuar. . .

```

Figura 17 - Linhas 1 a 50.

Fonte: Elaborado pela autora, 2019.

Estes exemplos mostraram como fazer algumas manipulações com os vetores. Se você

sabe somar dois vetores, saberá somar duas notas, duas alturas, dois pesos etc.

CASO

Uma empresa de projetos e instalação de redes de computadores deseja contratar dois funcionários para o departamento de projetos e infraestrutura de redes. Como a procura foi muito grande, eles tiveram a inscrição de 25 candidatos, e optaram por fazer uma prova de seleção. A prova consiste em 10 questões de múltiplas escolhas, com cinco opções de respostas (1, 2, 3, 4, 5), e cada questão vale um ponto. Esta prova irá selecionar os candidatos que obtiverem notas iguais ou superiores a 6. Somente os candidatos que passarem pela prova de seleção irão para a etapa de entrevista. A empresa deseja um sistema para facilitar a correção das provas e a identificação dos candidatos que obtiveram notas iguais ou superiores a 6. Você foi convidado para desenvolver este sistema. O sistema terá uma tela principal com 4 opções: 1-cadastrar o gabarito, 2-lançar as respostas dos candidatos, 3-relatórios e 4-sair. O gabarito contém as respostas corretas. Por exemplo, na questão de número 1, a resposta correta é o número 4. Dica: veja o exemplo anterior, no qual comparamos dois vetores. Lá não guardávamos a quantidade de números iguais. Para este exemplo, guarde a quantidade de números iguais dentro de um vetor. Depois de resolver este problema, você pode conferir sua resposta, no código abaixo, mas tente fazer antes de olhar a resposta. Bom estudo!

```
1 #include<stdio.h>
2 #include<stdlib.h>
3 #define TAMANHO 10
4 #define QDECANDIDATOS 25
5 main()
6 {
7     int i, j, op, op2, resposta[TAMANHO]={0};
8     int gabarito[TAMANHO]={0}, candidato[QDECANDIDATOS]={0};
9     do
10    {
11        printf("\n1 - Carregar gabarito\n");
12        printf("2 - Lancar notas\n");
13        printf("3 - Relatorios\n");
14        printf("4 - Sair\n");
15        printf("Opcão: ");
16        scanf("%d", &op);
17        if(op == 1)
18        {
19            printf("GABARITO\n");
20            for(i = 0; i<=TAMANHO-1 ; i++)
21            {
22                printf("Resposta da questao %d: ", i+1);
23                scanf("%d", &gabarito[i]);
24            }
25            printf("Gabarito = ");
26            for(i = 0; i<=TAMANHO-1 ; i++)
27            {
28                printf("%d ", gabarito[i]);
29            }
30            system("pause");
31        }
32        if(op == 2)
33        {
34            for(i = 0; i<=QDECANDIDATOS-1 ; i++)
35            {
36                candidato[i] = 0;
37            }
38            j = 0;
39            do
40            {
41                printf("Candidato %d\n", j+1);
42                for(i = 0; i<=TAMANHO-1 ; i++)
43                {
44                    printf("Questao %d ", i + 1);
45                    scanf("%d", &resposta[i]);
46                    if(resposta[i] == gabarito[i])
47                        candidato[j]++;
48                }
49            }while(j < QDECANDIDATOS);
50        }
51        if(op == 3)
52        {
53            printf("Candidatos Aprovados\n");
54            for(i = 0; i<=QDECANDIDATOS-1 ; i++)
55            {
56                if(candidato[i]>=6)
57                {
58                    printf("Candidato %d, nota = %d \n",i+1, candidato[i]);
59                }
60            }
61        }
62    }while(op != 4);
63 }
```

Agora, vamos entender como a linguagem C trabalha com caracteres.

4.2.2 Manipulação de strings em C

String, em C, é um vetor de caracteres. Para trabalhar com caracteres, precisamos utilizar uma biblioteca específica para isso, a *string.h*.

Primeiro, vamos diferenciar caracteres de *strings*. Caractere é apenas um elemento, devendo estar sempre entre aspas simples. Já uma *string*, que é um vetor de caracteres, deve sempre estar entre aspas duplas.

VOCÊ QUER LER?

Trabalhar com cadeia de caracteres na linguagem C é muito divertido. Mas para você fazer muitas coisas é necessário conhecer as funções que o C oferece, além das que a gente conheceu, existem outras e, para aplicações variadas (C PROGRESSIVO, s/d). Conheça mais sobre como manipular as *strings*, acessando o *site*

<<https://www.cprogressivo.net/2013/03/Aprenda-a-usar-todas-as-funcoes-da-biblioteca-> (<https://www.cprogressivo.net/2013/03/Aprenda-a-usar-todas-as-funcoes-da-biblioteca-string-h-em-C.html>)*string* (<https://www.cprogressivo.net/2013/03/Aprenda-a-usar-todas-as-funcoes-da-biblioteca-string-h-em-C.html>)-h-em-C.html
(<https://www.cprogressivo.net/2013/03/Aprenda-a-usar-todas-as-funcoes-da-biblioteca-string-h-em-C.html>)>.

Uma característica interessante de *strings* no C, é que as *strings* devem possuir um terminador \0, indicando o fim da sequência de caracteres (ASCENCIO; CAMPOS, 2007).

Para inicializar uma *string*, seguimos o mesmo princípio dos outros tipos de dados. A seguir, foi criado uma *string*, cujo nome é *nome* e que foi inicializada com os valores de “joao”. Neste caso, o próprio C coloca no final o terminador \0. Veja na próxima figura.

```
1 #include<stdio.h>
2 main()
3 {
4     char nome[5]={"joao"};
5 }
```

j	o	a	o	\0
---	---	---	---	----

Figura 18 - Declarar e inicializar uma string.

Fonte: Elaborada pela autora, 2019.

Agora, vamos estudar algumas funções que manipulam *strings*. A seguir temos algumas funções, suas descrições e uns exemplos.

Começamos por **scanf**:

- lê palavras do teclado;
- não serve para ler frases;
- identificador %s e a variável não é precedida pelo &;
- observe o exemplo, o nome digitado é composto, mas o *scanf* só leu a primeira palavra, figura a seguir.

```
1 #include<stdio.h>
2 main()
3 {
4     char nome[10];
5     printf("Digite seu nome: ");
6     scanf("%s", nome);
7     printf("Olá %s ", nome);
8 }
```

```
D:\ArquivosPessoais\DtCom\CAP04\prog09.exe
Digite seu nome: Cristiano Ronaldo
Olá Cristiano
-----
Process exited after 37.57 seconds with return value 0
Pressione qualquer tecla para continuar. . .
```

Figura 19 - Programa para ler um nome usando o scanf.

Fonte: Elaborada pela autora, 2019.

Agora, as características do **gets**:

- Lê uma frase do teclado até Agora, as características do a tecla *enter* ser apertada;
- sintaxe: *gets(variavel)*.

```
1 #include<stdio.h>
2 main()
3 {
4     char nome[50];
5     printf("Digite seu nome: ");
6     gets(nome);
7     printf("Ola %s ", nome);
8 }
```

D:\ArquivosPessoais\DtCom\CAP04\prog09.exe

```
Digite seu nome: Mario Sergio Silva
Ola Mario Sergio Silva
-----
Process exited after 14.94 seconds with return value 0
Pressione qualquer tecla para continuar. . .
```

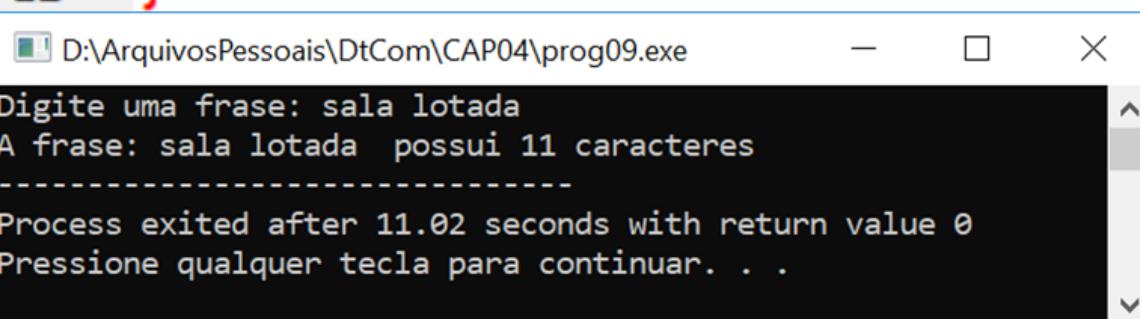
Figura 20 - Programa para ler um nome usando o `gets`.

Fonte: Elaborada pela autora, 2019.

A seguir, o **`strlen`**:

- biblioteca `string.h`;
- retorna um inteiro que é o número de caracteres da `string`, antes do terminador `\0`;
- sintaxe: `strlen(string)`.

```
1 #include<stdio.h>
2 #include<string.h>
3 main()
4 {
5     char frase[50];
6     int i;
7     printf("Digite uma frase: ");
8     gets(frase);
9     i = strlen(frase);
10    printf("A frase: %s ", frase);
11    printf(" possui %d caracteres ", i);
12 }
```



```
D:\ArquivosPessoais\DtCom\CAP04\prog09.exe
Digite uma frase: sala lotada
A frase: sala lotada possui 11 caracteres
-----
Process exited after 11.02 seconds with return value 0
Pressione qualquer tecla para continuar. . .
```

Figura 21 - Programa para contar o tamanho da string.

Fonte: Elaborada pela autora, 2019.

Agora, vamos saber sobre o **strcpy**:

- biblioteca *string.h*;
- copia o valor de uma variável para outra, ambas do tipo *string*, ou copia uma *string* para uma variável;
- sintaxe: *strcpy*(variável_destino, variável_origem).

```
1 #include<stdio.h>
2 #include<string.h>
3 main()
4 {
5     char frase[200], copia[200] ;
6     printf("Digite uma frase: ");
7     gets(frase);
8     strcpy(copia, frase);
9     printf("A frase: %s ", frase);
10    printf("\nA copia: %s ", copia);
11 }
```

```
D:\ArquivosPessoais\DtCom\CAP04\prog10.exe
Digite uma frase: não estou com fome
A frase: não estou com fome
A copia: não estou com fome
-----
Process exited after 21.88 seconds with return value 0
Pressione qualquer tecla para continuar. . .
```

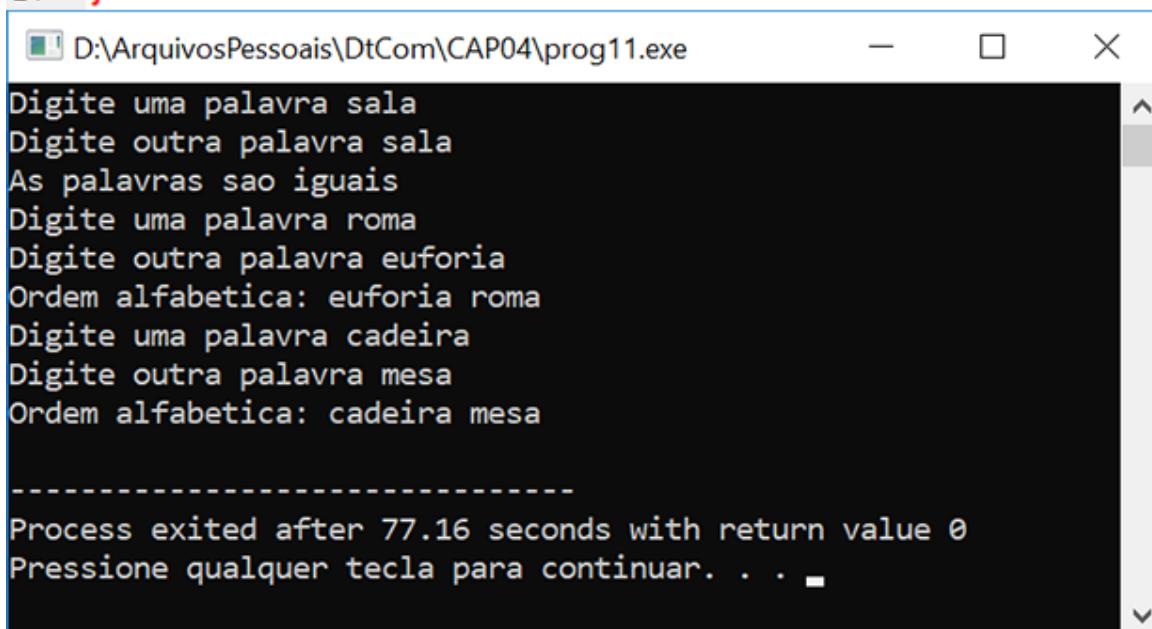
Figura 22 - Programa para copiar uma string.

Fonte: Elaborada pela autora, 2019.

Seguem as características do ***strcmp***.

- Biblioteca *string.h*.
- Compara duas *strings*, retornando um inteiro:
 - zero: as duas *strings* são iguais;
 - negativo: a *string* 1 é menor (alfabeticamente) que a *string* 2;
 - positivo: a *string* 1 é maior (alfabeticamente) que a *string* 2.
- Considera letras maiúsculas diferentes de minúsculas.
- Sintaxe: variável = *strcmp(str1, str2)*.

```
1 #include<stdio.h>
2 #include<string.h>
3 main()
4 {
5     char palavra[200], palavra2[200] ;
6     int res;
7     for(int i=0; i<3; i++)
8     {
9         printf("Digite uma palavra ");
10        gets(palavra);
11        printf("Digite outra palavra ");
12        gets(palavra2);
13        res = strcmp(palavra, palavra2);
14        if(res == 0)
15            printf("As palavras sao iguais\n");
16        else
17        {
18            if(res<0)
19                printf("Ordem alfabetica: %s %s\n", palavra, palavra2);
20            else
21                printf("Ordem alfabetica: %s %s\n", palavra2, palavra);
22        }
23    }
24 }
```



```
D:\ArquivosPessoais\DtCom\CAP04\prog11.exe
Digite uma palavra sala
Digite outra palavra sala
As palavras sao iguais
Digite uma palavra roma
Digite outra palavra euforia
Ordem alfabetica: euforia roma
Digite uma palavra cadeira
Digite outra palavra mesa
Ordem alfabetica: cadeira mesa

-----
Process exited after 77.16 seconds with return value 0
Pressione qualquer tecla para continuar. . . -
```

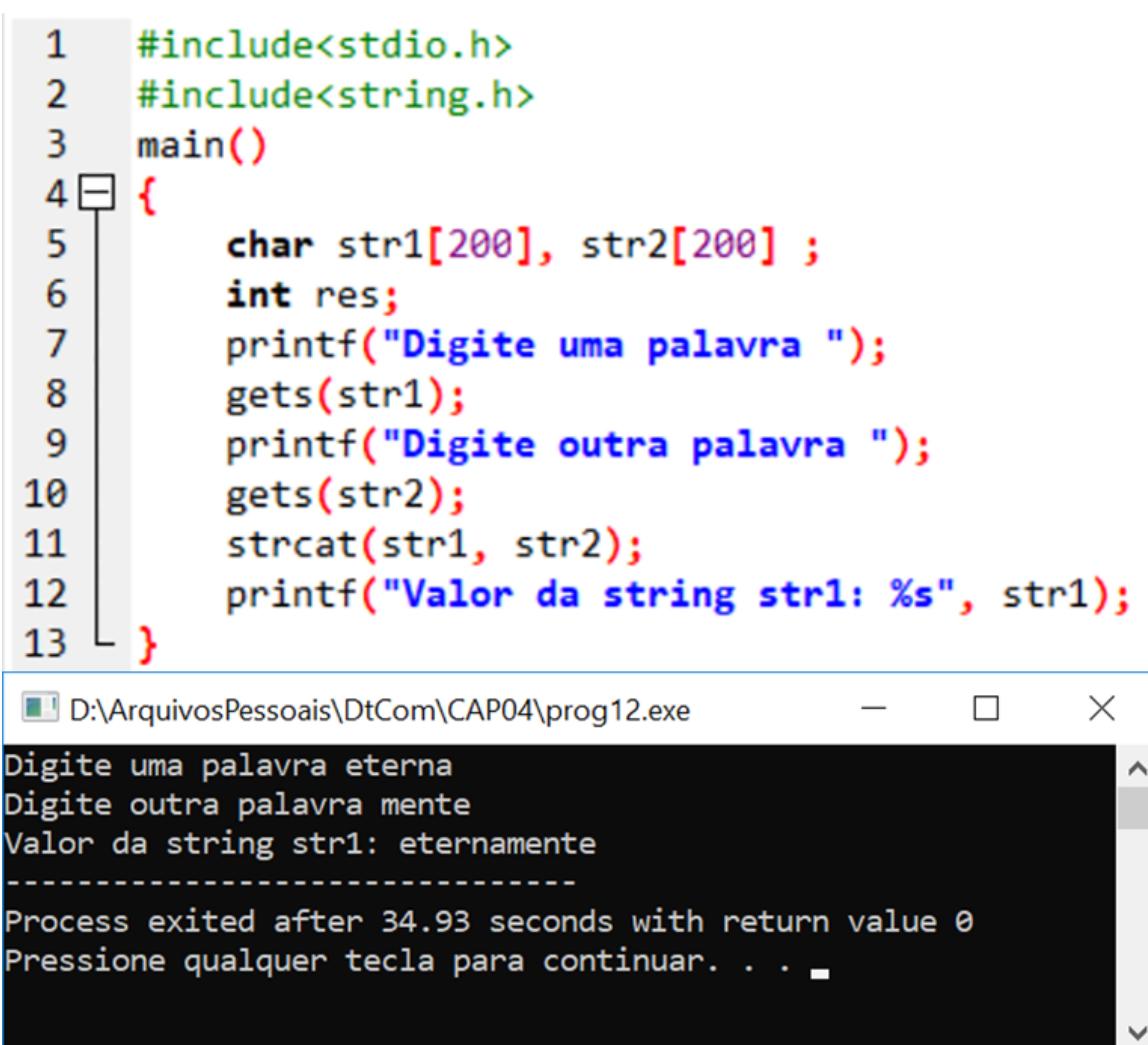
Figura 23 - Programa para colocar em ordem alfabética duas palavras.

Fonte: Elaborada pela autora, 2019.

A seguir, as características do **strcat**.

- Biblioteca *string.h*.

- Concatena duas *strings*.
- Sintaxe: *strcat(str1, str2)*.
- Copia a *string* str2 para a str1, sem apagar o valor de str1, começando pelo 0/0.



The screenshot shows a terminal window with the following content:

```
1 #include<stdio.h>
2 #include<string.h>
3 main()
4 {
5     char str1[200], str2[200] ;
6     int res;
7     printf("Digite uma palavra ");
8     gets(str1);
9     printf("Digite outra palavra ");
10    gets(str2);
11    strcat(str1, str2);
12    printf("Valor da string str1: %s", str1);
13 }
```

D:\ArquivosPessoais\DtCom\CAP04\prog12.exe

```
Digite uma palavra eterna
Digite outra palavra mente
Valor da string str1: eternamente
-----
Process exited after 34.93 seconds with return value 0
Pressione qualquer tecla para continuar. . .
```

Figura 24 - Programa para concatenar duas palavras.

Fonte: Elaborada pela autora, 2019.

Com isso, encerramos o assunto vetores e nosso próximo tópico é sobre matrizes.
Continue acompanhando!

4.3 Estruturas de dados multidimensionais – Matrizes com a Linguagem de Programação C

• os vetores são estruturas de armazenamento de uma dimensão, ou apenas

ou uma coluna. Existe a estrutura de armazenamento de duas dimensões, linhas, as chamadas matrizes ou *arrays* bidimensionais.

Para acessar um elemento de um vetor, é necessário informar o nome do vetor e a posição do elemento e, para as matrizes, a regra é a mesma. O que muda é a posição, nas matrizes, a posição é o encontro da linha com a coluna. Então, por exemplo, temos uma matriz A cujo elemento, na linha 3 e coluna 2, é 5, sendo identificado por A[3,2]. Lembrando que nas matrizes, também as posições de linhas e colunas começam com 0. Neste tópico, vamos entender como as matrizes se estruturam e como manipulá-las. Bons estudos!

4.3.1 Introdução

As matrizes são *arrays* de duas dimensões, compostas de linhas e colunas que armazenam o mesmo tipo de dados. Por isso são conhecidas como estruturas homogêneas de armazenamento. As matrizes diferem-se dos vetores apenas pela dimensão.

Veja a figura a seguir. É uma matriz com 3 linhas e 4 colunas, portanto, esta matriz consegue armazenar 12 elementos. Cada elemento é identificado pela sua posição. A posição na matriz é o número da linha e o número da coluna. Por exemplo, o elemento em destaque na figura, é o elemento na posição 0x2, linha 0 e coluna 2, ou seja, primeira linha e terceira coluna, pois a numeração começa com 0.

		colunas			
		0	1	2	3
linha 0	0	light blue	light blue	red	light blue
	1	yellow	yellow	yellow	yellow
	2	pink	pink	pink	pink

Figura 25 - Matriz 3x4.

Fonte: Elaborada pela autora, 2019.

Clique nos itens para ver como podemos resumir isso.

- Uma matriz é um conjunto de variáveis, ou seja, posições de memória adjacentes.
- Possui um nome e armazena, sempre o mesmo tipo de dados, como números ou vetores.
- Cada elemento, ou variável, possui uma posição que identifica a linha e coluna a que pertence.
- A numeração deve ser sequencial, com passo de 1, ou seja, a numeração irá de 1 em 1, começando por 0, tanto para as linhas, quanto para as colunas.

VOCÊ O CONHECE?

Você já ouviu falar de Mark Zuckerberg? Genial e polêmico, ele é o fundador do Facebook, a maior rede social do planeta. Desde a criação do Facebook até o ano de 2018, ele acumula vários processos jurídicos (LOPES, 2018). Saiba mais sobre a carreira deste jovem talento na área de TI, na reportagem:

<<https://revistagalileu.globo.com/Tecnologia/noticia/2018/05/mark-zuckerberg-12-fatos-sobre-o-fundador-do-facebook.html>
(<https://revistagalileu.globo.com/Tecnologia/noticia/2018/05/mark-zuckerberg-12-fatos-sobre-o-fundador-do-facebook.html>)>.

A seguir, vamos entender como a linguagem C estruturas as matrizes.

4.3.2 Estrutura

No C, para declarar uma matriz usa-se a seguinte sintaxe:

tipo_dado nome_matriz [numero_linhas] [numero_colunas]

Clique nas abas para conhecer cada um dos termos.

.

tipo_dado

É o tipo de dados que serão armazenados na matriz, por exemplo, *int*, *float*, *char*, etc.

.

nome_matriz

É o nome da matriz. Segue as mesmas regras de criação de variáveis simples.

número_linhas

É a quantidade de linhas que a matriz terá.

numero_colunas

É a quantidade de colunas que a matriz terá.

VOCÊ SABIA?

Você sabia que a linguagem C permite a criação de matrizes sem a especificação da quantidade de linhas ou de colunas? Isto é bastante útil quando você precisar ler uma sequência de dados sem saber o tamanho (CASAVELLA, s/d). Por exemplo, um programa irá colher as opiniões de vários clientes de um *shopping*. Neste caso, não se sabe de quantos clientes. Leia mais sobre como declarar matrizes e vetores sem especificar a quantidade, em

<http://linguagemc.com.br/> (<http://linguagemc.com.br/arrays-com-variadas-dimensoes-em-c/>)
<http://linguagemc.com.br/arrays-com-variadas-dimensoes-em-c/>-com-variadas-dimensoes-em-c/
(<http://linguagemc.com.br/arrays-com-variadas-dimensoes-em-c/>)>.

Veja o exemplo a seguir, na próxima figura. Na linha 4, foi declarada uma matriz *mat* com 2 linhas e 3 colunas, ou seja, com $2 \times 3 = 6$ elementos, do tipo inteira. As linhas possuem numeração de 0 a 1, e as colunas de 0 a 2. Na linha 6, são armazenados os três números digitados pelo usuário, na primeira linha da matriz, na linha 0. Sendo, o primeiro número digitado, armazenado na primeira coluna, coluna 0, o segundo número, na segunda coluna, coluna 1, e o último número digitado, na terceira coluna, coluna 2.

Nas linhas de 7 a 9, do programa, são atribuídos os valores de 1, para a linha 1, segunda linha, da matriz. A matriz é impressa nas linhas 10 e 11 do programa.

The screenshot shows a terminal window titled "D:\ArquivosPessoais\DtCom\CAP04\prog13.exe". The program code is as follows:

```
1 #include<stdio.h>
2 main()
3 {
4     int mat[2][3];
5     printf("Digite 3 numeros ");
6     scanf("%d %d %d", &mat[0][0], &mat[0][1], &mat[0][2]);
7     mat[1][0]=1;
8     mat[1][1]=1;
9     mat[1][2]=1;
10    printf("Primeira linha da matriz: %d %d %d\n", mat[0][0], mat[0][1], mat[0][2]);
11    printf("Segunda linha da matriz: %d %d %d\n", mat[1][0], mat[1][1], mat[1][2]);
12 }
```

The terminal output is:

```
Digit 3 numeros 10 15 20
Primeira linha da matriz: 10 15 20
Segunda linha da matriz: 1 1 1

-----
Process exited after 11.49 seconds with return value 0
Pressione qualquer tecla para continuar. . .
```

Figura 26 - Programa para carregar uma Matriz 2x3.

Fonte: Elaborada pela autora, 2019.

Para inicializar uma matriz na declaração, coloca-se o sinal de igual e abre-se as chaves. Dentro destas chaves, para cada linha abre-se outras chaves e coloca os valores correspondentes de cada coluna. Parece confuso? Veja o exemplo a seguir, próxima figura. Na linha 4 do programa, a matriz é criada, com 2 linhas e 3 colunas. E, já é inicializada com os valores de 1,2 e 3 na primeira linha e com os valores de 4, 5 e 6, na segunda linha.

The screenshot shows a terminal window with the following content:

```
1 #include<stdio.h>
2 main()
3 {
4     int mat[2][3] = {{1,2,3}, {4,5,6}};
5     printf("Primeira linha da matriz: %d %d %d\n", mat[0][0], mat[0][1], mat[0][2]);
6     printf("Segunda linha da matriz: %d %d %d\n", mat[1][0], mat[1][1], mat[1][2]);
7 }
```

D:\ArquivosPessoais\DtCom\CAP04\prog14.exe

```
Primeira linha da matriz: 1 2 3
Segunda linha da matriz: 4 5 6
-----
Process exited after 4.024 seconds with return value 0
Pressione qualquer tecla para continuar. . .
```

Figura 27 - Programa inicializa uma Matriz 2x3 na declaração.

Fonte: Elaborada pela autora, 2019.

O próximo exemplo, na figura a seguir, temos duas matrizes, matA e matB, ambas do mesmo tamanho, 2x3, 2 linhas por 3 colunas, inicializadas com os mesmos valores. Nas linhas de 6 a 8, do programa, somamos a primeira linha de matA com a primeira linha de matB, posição por posição, ou seja, o elemento da posição 0x0 de matB será somado ao elemento da posição 0x0 de matA, e o resultado será armazenado na posição 0x0 de matB. Estamos usando o princípio de somados.

The screenshot shows a terminal window with the following content:

```
1 #include<stdio.h>
2 main()
3 {
4     int matA[2][3] = {{1,2,3}, {4,5,6}};
5     int matB[2][3] = {{1,2,3}, {4,5,6}};
6     matB[0][0] = matB[0][0] + matA[0][0];
7     matB[0][1] = matB[0][1] + matA[0][1];
8     matB[0][2] = matB[0][2] + matA[0][2];
9     printf("Primeira linha da matriz B: %d %d %d\n",matB[0][0], matB[0][1], matB[0][2]);
10    printf("Segunda linha da matriz B: %d %d %d\n",matB[1][0], matB[1][1], matB[1][2]);
11 }
```

D:\ArquivosPessoais\DtCom\CAP04\prog15.exe

```
Primeira linha da matriz B: 2 4 6
Segunda linha da matriz B: 4 5 6

-----
Process exited after 4.172 seconds with return value 0
Pressione qualquer tecla para continuar. . .
```

Figura 28 - Programa para somar a primeira linha de B com a primeira linha de A.

Fonte: Elaborada pela autora, 2019.

Você percebeu que trabalhar com matrizes possui o mesmo princípio de trabalhar com vetores, com a exceção da identificação das posições. Nas matrizes, temos que identificar qual a linha e qual a coluna do elemento. Agora, vamos aprender como manipular as matrizes de forma otimizada, porque ficar escrevendo uma fórmula de manipulação para cada posição da matriz, não facilita nosso trabalho.

4.4 Estruturas de dados multidimensionais – Matrizes com a Linguagem de Programação C

As matrizes são estruturas de armazenamento temporário, bidimensional e de dados de mesmo tipo. Para declarar uma matriz na linguagem C, basta colocar o tipo, seguido do nome e a quantidade de linhas e de colunas. Para acessar um elemento, use o nome da matriz e o número da linha e da coluna deste elemento. A quantidade de elemento de uma matriz será dada pela quantidade de linhas vezes a quantidade de colunas, por exemplo, uma matriz A[3,4] possui 12 elementos.

Já sabemos o que são as matrizes, como são estruturadas no C, como declará-las, como inicializá-las na declaração, como acessar um elemento e como alterar seus valores.

Neste tópico, aprenderemos como manipular de forma otimizada e prática, estas poderosas estruturas: as matrizes.

Vamos lá!

4.4.1 Manipulação

Em vetores, vimos que para percorrê-los bastava utilizar um comando de repetição. Será que nas matrizes, como são dois índices, linha e coluna, precisaremos dois comandos de repetição? Sim, precisaremos de dois comandos, um para as linhas e o outro para as colunas.

Analise o exemplo a seguir, próxima figura. É criada uma matriz A com duas linhas e três colunas. As variáveis l e c, começam com os valores de 0, linhas 5 e 6 do programa. Enquanto a variável l vale 0, a variável c vai de 0 a 2. Depois a variável l recebe o valor de 1, e a variável c, recomeça com valor de 0 e vai até o valor de 2, novamente. Isso é a mesma coisa de um comando de repetição dentro do outro.

```
1 #include<stdio.h>
2 main()
3 {
4     int A[2][3], l, c;
5     l = 0;
6     c = 0;
7     A[l][c] = 10;
8     c = 1;
9     A[l][c] = 20;
10    c = 2;
11    A[l][c] = 30;
12    l = 1;
13    c = 0;
14    A[l][c] = 40;
15    c = 1;
16    A[l][c] = 50;
17    c = 2;
18    A[l][c] = 60;
19    printf("Primeira linha da matriz: %d %d %d\n",A[0][0], A[0][1], A[0][2]);
20    printf("Segunda linha da matriz: %d %d %d\n",A[1][0], A[1][1], A[1][2]);
21 }
```

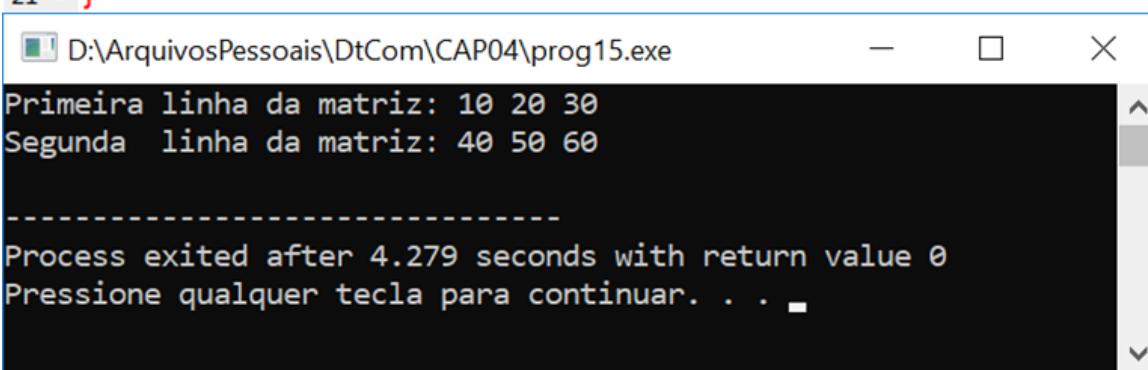


Figura 29 - Programa para carregar uma matriz.

Fonte: Elaborada pela autora, 2019.

Veja como ficou o programa para carregar a mesma matriz A[2x3], mas agora utilizando dois comandos de repetição, aninhados (um dentro do outro), próxima figura. Clique na interação a seguir para ler algumas observações.

Agora, veja a imagem a seguir:

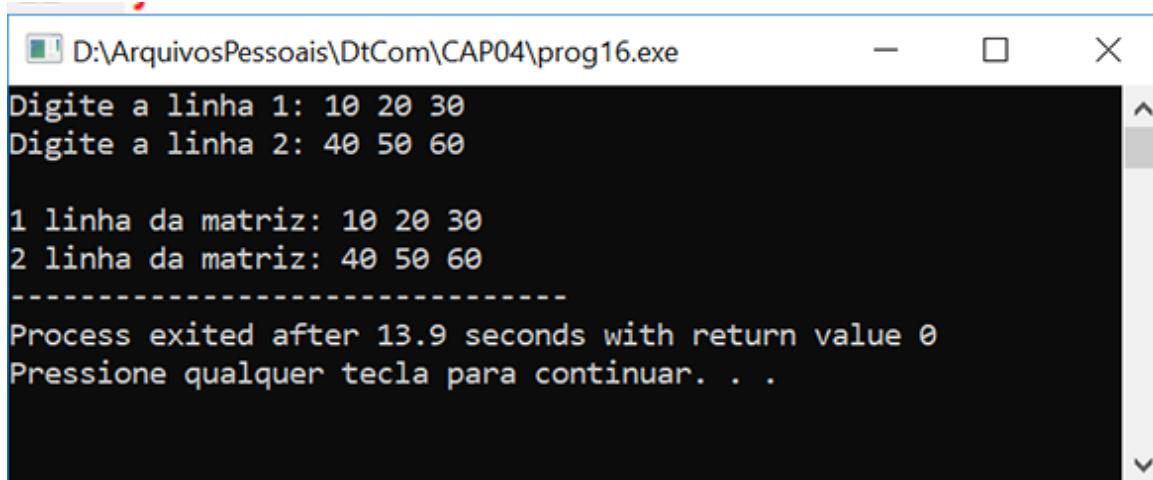
Para percorrer uma matriz são necessários dois comandos de repetição aninhados.

Se o comando de repetição interno representar as colunas, a matriz será preenchida por linhas, ou seja, preencherá uma linha inteira para, depois, ir para a próxima linha.

Se o comando de repetição interno representar as linhas, a matriz será preenchida por colunas, ou seja, preencherá uma coluna inteira para depois ir para a próxima coluna.

Neste exemplo, a matriz foi carregada por linha, primeiro a linha 0, e depois a linha 1.

```
1 #include<stdio.h>
2 main()
3 {
4     int A[2][3], l, c;
5     for(l = 0; l <= 1; l++)
6     {
7         printf("Digite a linha %d: ", l+1 );
8         for(c = 0; c <= 2; c++)
9         {
10            scanf("%d", &A[l][c]);
11        }
12    }
13    for(l = 0; l <= 1; l++)
14    {
15        printf("\n%d linha da matriz: ", l+1);
16        for(c = 0; c <= 2; c++)
17        {
18            printf("%d ", A[l][c] );
19        }
20    }
21 }
```



Digite a linha 1: 10 20 30
Digite a linha 2: 40 50 60

1 linha da matriz: 10 20 30
2 linha da matriz: 40 50 60

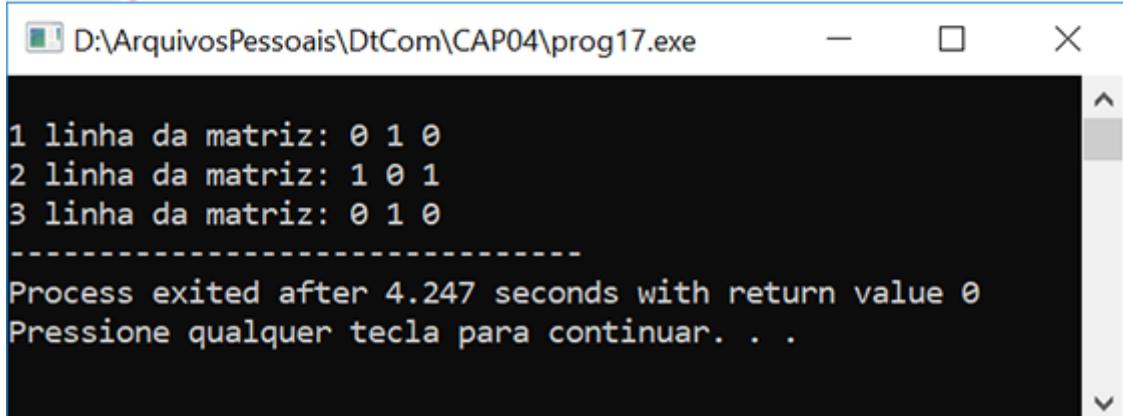
Process exited after 13.9 seconds with return value 0
Pressione qualquer tecla para continuar. . .

Figura 30 - Programa para carregar uma matriz com comandos de repetição for.

Fonte: Elaborada pela autora, 2019.

Neste outro exemplo, o programa cria uma matriz A[3x3] com a sua diagonal principal valendo 1 e o restante valendo 0. Lembrando que a diagonal principal de uma matriz é a posição onde o número da linha é igual ao número da coluna. Vide próxima figura.

```
1 #include<stdio.h>
2 main()
3 {
4     int A[3][3], l, c;
5     for(l = 0; l <= 2; l++)
6     {
7         for(c = 0; c <= 2; c++)
8         {
9             if((c+l)%2 == 0)
10                 A[l][c] = 0;
11             else
12                 A[l][c] = 1;
13         }
14     }
15     for(l = 0; l <= 2; l++)
16     {
17         printf("\n%d linha da matriz: ", l+1);
18         for(c = 0; c <= 2; c++)
19         {
20             printf("%d ", A[l][c]);
21         }
22     }
23 }
```



```
D:\ArquivosPessoais\DtCom\CAP04\prog17.exe
1 linha da matriz: 0 1 0
2 linha da matriz: 1 0 1
3 linha da matriz: 0 1 0
-----
Process exited after 4.247 seconds with return value 0
Pressione qualquer tecla para continuar. . .
```

Figura 31 - Programa que preenche a diagonal principal 1.

Fonte: Elaborada pela autora, 2019.

Vamos construir um programa para ler alguns dados de 5 pessoas. O programa deverá

ler a idade, o peso e a altura, e deverá informar: a média, a maior e a menor idade, peso e altura. Clique nos itens para ver as configurações de matrizes que usaremos para construirmos este programa.

Cada linha representará uma informação, A primeira será a das alturas, a segunda linha será dos pesos e a terceira linha armazenará as idades.

A matriz será do tipo *float* por causa da altura e peso.

As colunas de 0 a 4 armazenarão os dados de cada pessoa. Coluna 0 armazena os dados da pessoa 1, coluna 1, dados da pessoa 2, até coluna 4 dados da pessoa 5.

A coluna 5, sexta coluna, armazenará a média dos dados.

A coluna 6, sétima coluna, armazenará o maior valor.

A coluna 7, oitava coluna, armazenará o menor valor.

Então, a nossa matriz será do tipo *float*, terá 3 linhas e 8 colunas, e a representação dos dados na matriz é ilustrada pela figura a seguir.

		colunas			
		0	1	2	3
linha 0	0	light blue	light blue	red	light blue
	1	yellow	yellow	yellow	yellow
	2	pink	pink	pink	pink

Figura 32 - Organização da matriz para o nosso exemplo.

Fonte: Elaborada pela autora, 2019.

O programa terá a seguinte tela principal, na próxima figura. A matriz dados foi criada e inicializada com 0, linha 5. Na linha 9, utilizamos o system("cls") para limpar a tela, que funciona apenas para Windows. Se for rodar no Linux ou Unix, utilize system("clear"). O comando do/while começa na linha 7 e termina na linha 136.

```
1 #include<stdio.h>
2 #include<stdlib.h>
3 main()
4 {
5     float dados[3][8]={0}, maior, menor, soma;
6     int i, j, op;
7     do
8     {
9         system("cls");
10        printf("1- Digitar alturas\n");
11        printf("2- Digitar pesos\n");
12        printf("3- Digitar idades\n");
13        printf("4- Relatorios\n");
14        printf("5- Zerar matriz\n");
15        printf("6- Sair\n");
16        printf("Opcao: ");
17        scanf("%d", &op);
136    }while(op!=6);
137 }
```

D:\ArquivosPessoais\CTI\CTILOG\CTILOG.c

```
1- Digitar alturas
2- Digitar pesos
3- Digitar idades
4- Relatorios
5- Zerar matriz
6- Sair
Opcao:
```

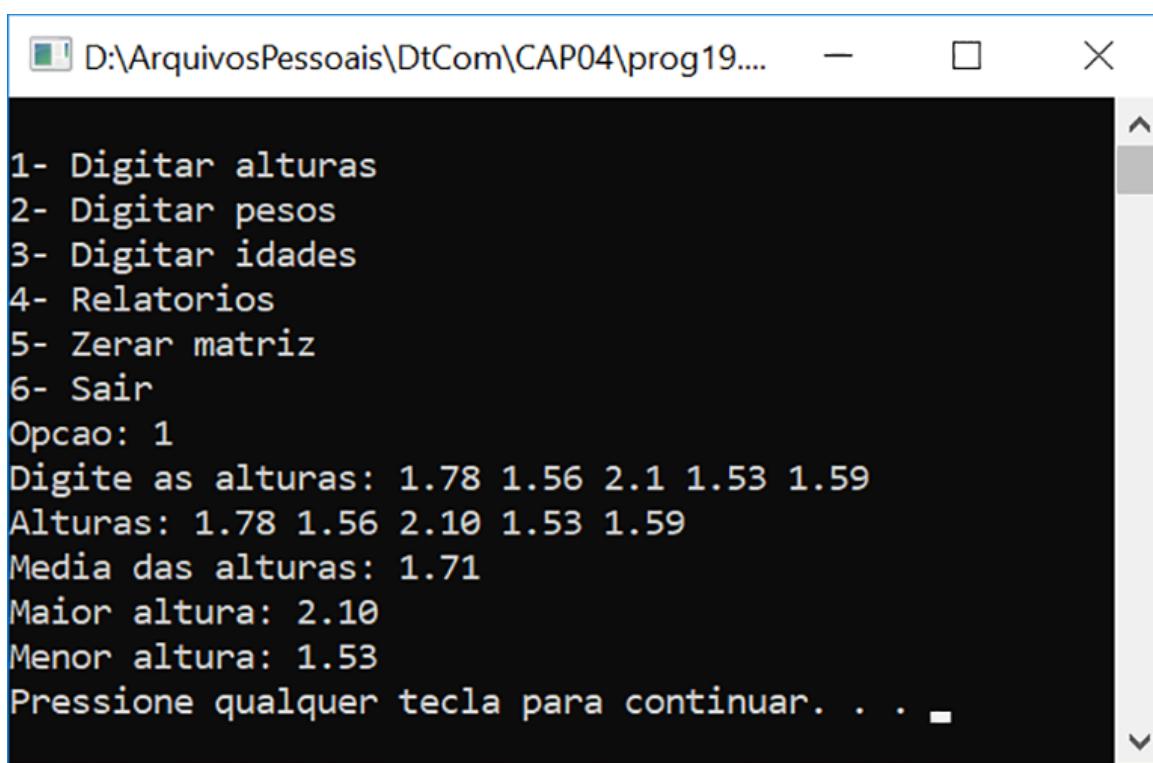
Figura 33 - Tela principal para o nosso exemplo.

Fonte: Elaborada pela autora, 2019.

A primeira opção é para a leitura das alturas. Na medida em que vamos lendo as

alturas, já aproveitamos para calcular a maior, menor e a média. Como sabemos que as alturas serão armazenadas na linha 0, não precisaremos de um outro comando de repetição. Veja como ficou a opção 1. Logo após a leitura de dados, foi exibida todas as alturas, a maior, menor e a média, 34 a 41. A variáveis maior, soma e menor, são inicializadas dentro da opção. Isto ocorre para que as variáveis não guardem valores de outras leituras. Por exemplo, o usuário digitou 5 alturas, viu a média, a maior e a menor altura. Depois resolveu digitar outras alturas. Se as variáveis não forem novamente inicializadas, os novos cálculos estarão errados, pois guardaram os valores anteriores. Veja a figura a seguir.

```
18 if(op==1)
19 {
20     printf("Digite as alturas: ");
21     maior = 0;
22     soma = 0;
23     menor = 1000;
24     for(i = 0; i <= 4; i++)
25     {
26         scanf("%f", &dados[0][i]);
27         soma = soma + dados[0][i];
28         if(dados[0][i] > maior)
29             maior = dados[0][i];
30         if(dados[0][i] < menor)
31             menor = dados[0][i];
32     }
33     dados[0][5] = soma/5;
34     dados[0][6] = maior;
35     dados[0][7] = menor;
36     printf("Alturas: ");
37     for(i = 0; i <= 4; i++)
38     {
39         printf("%.2f ", dados[0][i] );
40     }
41     printf("\nMedia das alturas: %.2f\n", dados[0][5] );
42     printf("Maior altura: %.2f\n", dados[0][6] );
43     printf("Menor altura: %.2f\n", dados[0][7] );
44     system("pause");
45 }
```



D:\ArquivosPessoais\DtCom\CAP04\prog19....

```
1- Digitar alturas
2- Digitar pesos
3- Digitar idades
4- Relatorios
5- Zerar matriz
6- Sair
Opcao: 1
Digite as alturas: 1.78 1.56 2.1 1.53 1.59
Alturas: 1.78 1.56 2.10 1.53 1.59
Media das alturas: 1.71
Maior altura: 2.10
Menor altura: 1.53
Pressione qualquer tecla para continuar. . . -
```

Figura 34 - Opção 1: cadastro das alturas.

Fonte: Elaborada pela autora, 2019.

Para digitar as idades e os pesos, será o mesmo princípio. O que mudará é o número da linha. Para o peso será linha 1 e para a idade, linha 2. Veja a figura a seguir, para os pesos, opção 2.

```
46 if(op==2)
47 {
48     printf("Digite os pesos: ");
49     maior = 0;
50     soma = 0;
51     menor = 1000;
52     for(i = 0; i <= 4; i++)
53     {
54         scanf("%f", &dados[1][i]);
55         soma = soma + dados[1][i];
56         if(dados[1][i] > maior)
57             maior = dados[1][i];
58         if(dados[1][i] < menor)
59             menor = dados[1][i];
60     }
61     dados[1][5] = soma/5;
62     dados[1][6] = maior;
63     dados[1][7] = menor;
64     printf("Pesos: ");
65     for(i = 0; i <= 4; i++)
66     {
67         printf("%.2f ", dados[1][i]);
68     }
69     printf("\nMedia dos pesos: %.2f\n", dados[1][5]);
70     printf("Maior peso: %.2f\n", dados[1][6]);
71     printf("Menor peso: %.2f\n", dados[1][7]);
72     system("pause");
73 }
```

D:\ArquivosPessoais\DtCom\CAP04\prog19....

```
1- Digitar alturas
2- Digitar pesos
3- Digitar idades
4- Relatorios
5- Zerar matriz
6- Sair
Opcão: 2
Digite os pesos: 65 78.9 88 34.5 120
Pesos: 65.00 78.90 88.00 34.50 120.00
Media dos pesos: 77.28
Maior peso: 120.00
Menor peso: 34.50
Pressione qualquer tecla para continuar. . .
```

Figura 35 - Opção 2: cadastro dos pesos.

Fonte: Elaborada pela autora, 2019.

Para as idades, além da alteração do número da linha para 2, é interessante que se

imprima as idades sem as casas decimais, porque não temos o costume de falarmos nossa idade com número quebrados, com exceção de idades de crianças. Então, nas linhas 95, 98 e 99, a configuração do *float* foi para 0 casas decimais: %.0f. Analise a figura a seguir.

```
74 if(op==3)
75 {
76     printf("Digite as idades: ");
77     maior = 0;
78     soma = 0;
79     menor = 1000;
80     for(i = 0; i <= 4; i++)
81     {
82         scanf("%f", &dados[2][i]);
83         soma = soma + dados[2][i];
84         if(dados[2][i] > maior)
85             maior = dados[2][i];
86         if(dados[2][i] < menor)
87             menor = dados[2][i];
88     }
89     dados[2][5] = soma/5;
90     dados[2][6] = maior;
91     dados[2][7] = menor;
92     printf("Idades: ");
93     for(i = 0; i <= 4; i++)
94     {
95         printf("%.0f ", dados[2][i] );
96     }
97     printf("\nMedia das idades: %.2f\n", dados[2][5] );
98     printf("Maior idade: %.0f\n", dados[2][6] );
99     printf("Menor idade: %.0f\n", dados[2][7] );
100    system("pause");
101 }
```

D:\ArquivosPessoais\DtCom\CAP04\prog19....

```
1- Digitar alturas
2- Digitar pesos
3- Digitar idades
4- Relatorios
5- Zerar matriz
6- Sair
Opcao: 3
Digite as idades: 45 67 47 99 23
Idades: 45 67 47 99 23
Media das idades: 56.20
Maior idade: 99
Menor idade: 23
Pressione qualquer tecla para continuar. . .
```

Figura 36 - Opção 3: cadastro idades.

Fonte: Elaborada pela autora, 2019.

Para a opção de relatórios, copie de cada opção anterior a parte de mostrar os dados

para o usuário. Veja a figura a seguir.

The figure shows a terminal window with the following content:

```
1 D:\ArquivosPessoais\DtCom\CAP04\prog19.exe
1- Digitar alturas
2- Digitar pesos
3- Digitar idades
4- Relatorios
5- Zerar matriz
6- Sair
Opcao: 4
Alturas: 1.78 1.56 2.10 1.53 1.59
Media das alturas: 1.71
Maior altura: 2.10
Menor altura: 1.53
Pesos: 65.00 78.90 88.00 34.50 120.00
Media dos pesos: 77.28
Maior peso: 120.00
Menor peso: 34.50
Idades: 45 67 47 99 23
Media das idades: 56.20
Maior idade: 99
Menor idade: 23
Pressione qualquer tecla para continuar. . .
```

Figura 37 - Opção 4: relatórios.

Fonte: Elaborada pela autora, 2019.

A opção 5, zerar a matriz, percorreremos a matriz atribuindo um valor zero para cada elemento. Para percorrer a matriz inteira precisamos de dois comandos de repetição. Analise a figura a seguir.

The screenshot shows a code editor window with a terminal below it. The code in the editor is:

```
130
131 if(op==5)
132 {
133     for(i = 0; i <= 7; i++)
134         for(j = 0; j <= 2; j++)
135             dados[j][i] = 0;
```

The terminal window shows the following output:

```
D:\ArquivosPessoais\DtCom\CAP04\prog19.exe
1- Digitar alturas
2- Digitar pesos
3- Digitar idades
4- Relatorios
5- Zerar matriz
6- Sair
Opcão: 4
Alturas: 0.00 0.00 0.00 0.00 0.00
Media das alturas: 0.00
Maior altura: 0.00
Menor altura: 0.00
Pesos: 0.00 0.00 0.00 0.00 0.00
Media dos pesos: 0.00
Maior peso: 0.00
Menor peso: 0.00
Idades: 0 0 0 0 0
Media das idades: 0.00
Maior idade: 0
Menor idade: 0
Pressione qualquer tecla para continuar. . .
```

Figura 38 - Opção 5: zerar a matriz dados.

Fonte: Elaborada pela autora, 2019.

Agora que você já sabe trabalhar com matrizes, vamos continuar com as *strings* em C.

Você aprendeu que uma cadeia de caracteres é um vetor de caracteres na linguagem C. Então, nós vimos como trabalhar com palavras ou frases. Agora, vamos aprender como manipular um array bidimensional de caracteres. As funções são as mesmas, o que vai alterar é que, antes, a gente manipulava, por exemplo, um nome. Agora, vamos manipular um vetor de nomes, que para o C é uma matriz de caracteres.

Como primeiro exemplo, que tal escrevermos os dias da semana por extenso? Veja

próxima figura. Mas, antes, leia algumas observações.

- Na linha 6 foi declarada a matriz de caracteres dias, com 7 linhas por 10 colunas.
- Na linha 7, foi utilizada a função `strcpy` para copiar a *string* “domingo” para a primeira linha da matriz. Perceba que não foi necessária a inserção do número das colunas, bastou a identificação do número da linha.
- Das linhas 8 a 13, preenchimento da matriz:

d	o	m	i	n	g	o	\0
s	e	g	u	n	d	a	\0
t	e	r	c	a	\0		
q	u	a	r	t	a	\0	
q	u	i	n	t	a	\0	
s	e	x	t	a	\0		
s	a	b	a	d	o	\0	

Figura 39 - Preenchimento da matriz.

Fonte: Elaborado pela autora, 2019.

- Das linhas 14 a 20, comando `do/while` para permitir consultas do

usuário.

- Linha 19, de acordo com o número que o usuário digitou, será impresso o dia da semana. Uma correção deve ser feita para alinhar o número que o usuário digitou e a posição na matriz. Por exemplo, o usuário digitou o número 2, referente à segunda feira, mas segunda está na posição 1, por isso, no impresso ficou dias[op-1].

```
1 #include<stdio.h>
2 #include<string.h>
3 main()
4 {
5     int op, d;
6     char dias[7][8];
7     strcpy(dias[0], "domingo");
8     strcpy(dias[1], "segunda");
9     strcpy(dias[2], "terca");
10    strcpy(dias[3], "quarta");
11    strcpy(dias[4], "quinta");
12    strcpy(dias[5], "sexta");
13    strcpy(dias[6], "sabado");
14    do
15    {
16        printf("\ndigite o dia da semana ou 0 para sair: ");
17        scanf("%d", &op);
18        if(op>=1 && op<=7)
19            printf("%s", dias[op-1]);
20    }while(op!=0);
21 }
```

D:\ArquivosPessoais\DtCom\CAP04\prog20.exe

```
digite o dia da semana ou 0 para sair: 1
domingo
digite o dia da semana ou 0 para sair: 2
segunda
digite o dia da semana ou 0 para sair: 3
terca
digite o dia da semana ou 0 para sair: 4
quarta
digite o dia da semana ou 0 para sair: 5
quinta
digite o dia da semana ou 0 para sair: 6
sexta
digite o dia da semana ou 0 para sair: 7
sabado
digite o dia da semana ou 0 para sair: 8

digite o dia da semana ou 0 para sair: 0
-----
Process exited after 33.88 seconds with return value 0
Pressione qualquer tecla para continuar. . .
```

Figura 40 - Programa para escrever por extenso os dias da semana.

Fonte: Elaborada pela autora, 2019.

Por fim, depois de entender como manipular *strings* em C, temos uma boa base para

nossa compreensão de como manipular matrizes.

Assim, aprendemos sobre as duas estruturas de armazenamento temporário de dados: os vetores e as matrizes.

Os vetores são estruturas de uma dimensão e as matrizes, são multidimensionais. O vetor é um conjunto de variáveis, de mesmo tipo, que possui apenas uma dimensão. Os elementos são identificados pelo nome do vetor e a sua posição.

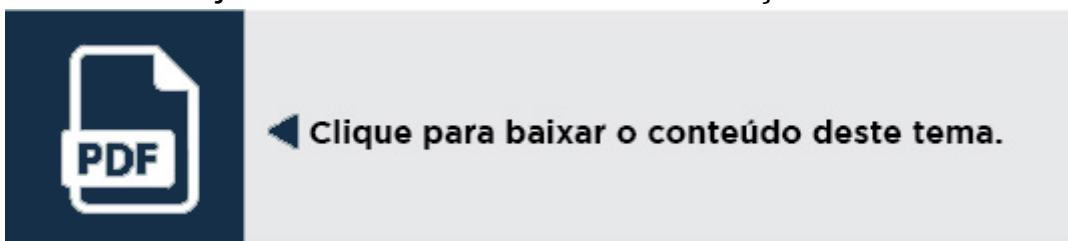
Já as matrizes, também chamadas de *arrays* bidimensionais, são consideradas como um conjunto de vetores, no qual, cada elemento é identificado pelo nome da matriz e a sua posição, linha e coluna.

Síntese

Chegamos ao final do capítulo. Aqui, estudamos as duas estruturas de armazenamento temporário: vetores e matrizes. Entendemos que os vetores são um conjunto de variáveis de mesmo tipo e com um único nome. Seus elementos possuem uma posição, um número, que o identifica, e a numeração das posições começa com 0. E as matrizes são *arrays* bidimensionais, de mesmo tipo, cujo elementos são identificados pelo nome da matriz e pelos números da linha e da coluna.

Neste capítulo, você teve a oportunidade de:

- conhecer as estruturas dos vetores em linguagem C;
- entender como carregar, alterar e apagar os valores dos vetores na linguagem de programação C;
- estudar as matrizes, também chamadas de *arrays* bidimensionais;
- manipular as matrizes utilizando a linguagem de programação C;
- compreender como a linguagem C trabalha com as cadeias de caracteres;
- aplicar os conceitos de vetores e as matrizes em exemplos práticos do dia a dia;
- perceber que os sistemas são mais completos e práticos se utilizarem os vetores e as matrizes, pois permitem que os valores sejam reutilizados durante a execução.



Bibliografia

- ASCENCIO, A. F. G.; CAMPOS, E. A. V. **Fundamentos da Programação de Computadores**. 2. ed. São Paulo: Pearson Prentice Hall, 2007.
- BRAGIL, R. **Gerando números aleatórios em C**. Porta Viva o Linux, publicado em 19 ago. 2003. Disponível em: <<https://www.vivaolinux.com.br/dica/Gerando-numeros-aleatorios-em-C>> (<https://www.vivaolinux.com.br/dica/Gerando-numeros-aleatorios-em-C>). Acesso em 2/2/2019.
- CASAVELLA, E. **Arrays com várias dimensões em C**. Portal Intellectuale Tecnologia e Treinamento, s/d. Disponível em: <<http://linguagemc.com.br/> (<http://linguagemc.com.br/arrays-com-varias-dimensoes-em-c/arrays> (<http://linguagemc.com.br/arrays-com-varias-dimensoes-em-c/-com-varias-dimensoes-em-c/>)> (<http://linguagemc.com.br/arrays-com-varias-dimensoes-em-c/>). Acesso em 2/2/2019.
- C PROGRESSIVO. **Strings e Caracteres em C - Tutorial**. Portal cprogressivo.net, s/d. Disponível em: <<https://www.cprogressivo.net/p/> (<https://www.cprogressivo.net/p/strings-e-caracteres-em-c.html>)> (<https://www.cprogressivo.net/p/strings-e-caracteres-em-c.html> (<https://www.cprogressivo.net/p/strings-e-caracteres-em-c.html>). Acesso em 2/2/2019.
- DEITEL, P.; DEITEL, H. **C++ Como Programar**. 5. ed. São Paulo: Pearson Prentice Hall, 2006.
- DEITEL, P.; DEITEL, H. **C Como Programar**. 6. ed. São Paulo: Pearson Prentice Hall, 2011.
- LOPES, L. **Mark Zuckerberg**: 12 fatos sobre o fundador do Facebook. Portal revista Galileu, publicado em 14 mai. 2018. Disponível em: <<https://revistagalileu.globo.com/Tecnologia/noticia/2018/05/mark-zuckerberg-12-fatos-sobre-o-fundador-do-facebook.html>> (<https://revistagalileu.globo.com/Tecnologia/noticia/2018/05/mark-zuckerberg-12-fatos-sobre-o-fundador-do-facebook.html>). Acesso em 2/2/2019.
- MICROSOFT. **Matrizes multidimensionais (C)**. Portal Microsoft Docs, publicado em 03/11/2016. Disponível em: <<https://docs.microsoft.com/pt-br/cpp/c-language/multidimensional-arrays-c?view=vs-2017>> (<https://docs.microsoft.com/pt-br/cpp/c-language/multidimensional-arrays-c?view=vs-2017>)> (<https://docs.microsoft.com/pt-br/cpp/c-language/multidimensional-arrays-c?view=vs-2017>)> (<https://docs.microsoft.com/pt-br/cpp/c-language/multidimensional-arrays-c?view=vs-2017>). Acesso em 2/2/2019.
- SORKIN, A. **Steve Jobs**. Direção: Danny Boyle. Produção: Danny Boyle, Guymon Casady, Christian Colson, Mark Gordon e Scott Rudin. Cor, 122min. Estados Unidos, 2015.