Apostila

Estrutura de Dados e Modularização em C

Professor Eros Schettini Roman
IFSP-Catanduva

Tipos de Dados em C

A linguagem C possui quatro tipos básicos de dados: *int, float, char* e *double*. Na maioria dos computadores, esses quatro tipos são nativos do hardware da máquina. Uma variável *double* é um número de ponto flutuante de dupla precisão. Uma declaração de variável em C especifica dois itens. Primeiro, a quantidade de armazenamento que deve ser reservada para objetos declarados com esse tipo. Por exemplo, uma variável do tipo *int* precisa ter espaço suficiente para armazenar o maior valor inteiro possível. Segundo, ela especifica como os dados representados por *strings* de bits devem ser interpretados. Os mesmos bits numa sequência específica de armazenamento podem ser interpretados como um inteiro ou um número de ponto flutuante, resultando em dois valores numéricos totalmente diferentes.

Uma declaração de variável especifica que deve ser reservado armazenamento para objeto do tipo especificado e que o objeto nessa posição de armazenamento pode ser referenciado com o identificador de variável especificado.

Vetores

Definição de Vetor

Vetor também é conhecido como variável composta homogênea unidimensional. Composta por poder armazenar um conjunto de valores em uma mesma variável. Homogênea, pois todos os elementos são do mesmo tipo e unidimensional por precisar de apenas uma dimensão para acessar seus elementos. Isso quer dizer que se trata de um conjunto de variáveis de mesmo tipo, que possuem o mesmo identificador (nome) e são alocadas sequencialmente na memória. Como as variáveis têm o mesmo nome, o que as distingue é um índice que referencia sua localização dentro da estrutura.

Os índices utilizados na linguagem C para identificar as posições de um vetor começam sempre em 0 (zero) e vão até o tamanho do vetor menos uma unidade.

Declaração de Vetor

Os vetores em C são identificados pela existência de colchetes logo após o nome da variável no momento da declaração. Dentro dos colchetes deve-se colocar o número de posições do vetor. Exemplo:

int vet[10];

vet

3	2	9	-5	34	7	65	8	-1	58
0	1	2	3	4	5	6	7	8	9

Nesse exemplo o vetor chamado vet possui dez posições, começando pela posição 0 e indo até a posição 9 (tamanho do vetor - 1). Em cada vetor poderão ser armazenados números inteiros, conforme especificado pelo tipo *int* na declaração.

char x[5];

x

D	#	f	}	8
0	1	2	3	4

Nesse exemplo o vetor chamado x possui cinco posições, começando pela posição 0 e indo até a posição 4. Em cada posição poderão ser armazenados caracteres, conforme especificado pelo tipo char na declaração.

Atribuindo Valores ao Vetor

As atribuições em um vetor exigem que seja informada em qual de suas posições o valor ficará armazenado. Deve-se lembrar sempre que a primeira posição de um vetor em C tem o índice 0.

vet[3] = -5; atribui o valor -5 à quarta posição do vetor

x[2] = 'f'; atribui a letra 'f' à terceira posição do vetor

Preenchendo um Vetor

Preencher um vetor significa atribuir valores a todas as suas posições. Assim, deve-se implementar um mecanismo que controle o valor do índice:

```
for (i=0 ; i<10 ; i++)
        scanf (“%d”, &vet[i]);
```

Neste exemplo, a estrutura de repetição *for* foi utilizada para garantir que a variável i assuma todos os valores possíveis para o índice do vetor (de 0 a 9). Assim, para cada execução da repetição, será utilizada uma posição diferente do vetor para fazer a leitura.

Mostrando os Elementos do Vetor

Mostrar os elementos contidos em um vetor também exige a utilização do índice.

```
for (i=0 ; i<10 ; i++)
        printf (“%d”,  vet[i]);
```

Neste exemplo, a estrutura de repetição *for* foi utilizada para garantir que a variável i assuma todos os valores possíveis para o índice do vetor (de 0 a 9). Assim, para cada execução da repetição, será utilizada uma posição diferente do vetor e, dessa forma, todos os valores do vetor serão mostrados.

Exemplos

1 – Crie um programa que leia um vetor inteiro de 10 posições e mostre-os na sequência.

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
        int i, vet[10];

        // Leitura dos elementos do vetor
        printf (“Informe 10 elementos inteiros:\n”);
        for (i=0 ; i<10 ; i++)
        {
                printf(“ Digite o elemento %d: “, i+1);
                scanf (“%d”, &vet[i]);
        }
        // Impressão dos elementos do vetor
        printf(“Os elementos lidos do vetor são: “);
        for (i=0 ; i<10 ; i++)
                printf(“%d- “,vet[i]);
        system ("pause");
        return 0;
```

```
}
```

2 – Crie um programa que leia um vetor inteiro de 10 posições e mostre-os em ordem inversa.

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
	int i, vet[10];
	// Leitura dos elementos do vetor
	printf (“Informe 10 elementos inteiros:\n”);
	for (i=0 ; i<10 ; i++)
	{
		printf(“Elemento %d: “, i+1);
		scanf (“%d”, &vet[i]);
	}
	// Impressão dos elementos do vetor em ordem inversa
	printf(“Os elementos lidos do vetor na ordem inversa são: “);
	for (i=9 ; i>=0 ; i--)
		printf(“Elemento [%d]: %d “,i+1, vet[i]);
	system ("pause");
	return 0;
}
```

3 – Crie um programa que leia um vetor do tipo real de 10 posições e calcule e mostre a somatória de todos os elementos.

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
	int i;
	float soma, vet[10];
	soma=0;
	printf (“Informe 10 elementos em ponto flutuante:\n”);
	for (i=0 ; i<10 ; i++)
	{
		printf(“Elemento %d: “, i+1);
		scanf (“%f”, &vet[i]);
		soma += vet[i];
	}
	printf(“A somatória  dos elementos lidos do vetor é: %f“, soma);
	system ("pause");
```

```
	return 0;
}
```

4 – Crie um programa com uma função que receba um vetor e o seu tamanho por parâmetro e retorne a média dos elementos do vetor para o programa principal.

```
#include <stdio.h>
#include <stdlib.h>
#define N 5

float funcao( int a[], int size )
{
	int i;
	float soma = 0;

	for ( i = 0; i < size; i++ )
		soma += a[i];
	return( soma / size );
}

int main()
{
	int a[N];
	int i;

	for ( i = 0; i < N; i++ )
		a[i] = i;
	printf("Resultado = %5.2f\n", funcao(a, N));
	system ("pause");
	return 0;
}
```

5 - Realizou-se uma pesquisa para determinar o índice de mortalidade infantil em um certo período. Construa um programa que leia o número de crianças nascidas no período e, depois, em um número indeterminado de vezes, o sexo de uma criança morta ("m": masculino; "f": feminino) e o número de meses de vida da criança. Como finalizador, teremos a letra "x" no lugar do sexo da criança. Calcule e imprima:

- A porcentagem de crianças mortas no período
- A porcentagem de crianças do sexo masculino mortas no período

- A porcentagem de crianças que viveram 2 anos ou menos no período

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

int main()
{
	int qtdNasc, mesVida, op = 1;
	int qtdCriancaMorta = 0, criancaMortaMasc = 0, criancaMortaIM2Anos =0;
	char sexo[2];

	printf("Qual a quantidade de criancas nascidas?\n");
	scanf("%d", &qtdNasc);
	printf("\n---------------------------------\n");
	printf("Criancas mortas no periodo:");
	printf("\n---------------------------------\n");

	while (op && qtdNasc > qtdCriancaMorta)
	{
		printf("\n-- Crianca %d\n\n", qtdCriancaMorta + 1);
		printf("Qual o sexo da crianca?\n");
		printf("[m] Masculino\n");
		printf("[f] Feminino\n");
		printf("[x] Sair\n");
		scanf("%s", &sexo);
		if (strcmp(sexo, "m") == 0)
			criancaMortaMasc++;
		else if (!(strcmp(sexo, "f") == 0))
			op = 0;
		if (op)
		{
			qtdCriancaMorta++;
			printf("\nCom quantos meses de vida a crianca morreu?\n");
			scanf("%d", &mesVida);
			if (mesVida <= 24) criancaMortaIM2Anos++;
		}
		if (qtdNasc == qtdCriancaMorta)
		{
			printf("\nFim do cadastramento:\n");
			printf("O numero de nascimentos e' igual ao numero de
			criancas mortas.\n");
		}
	}
```

```
        printf("\n---------------------------------\n");
        printf("Estatisticas:");
        printf("\n---------------------------------\n\n");
        printf("Criancas mortas no periodo: %0.2f%%\n", (float) qtdCriancaMorta
        * 100 / qtdNasc);
        printf("Criancas do sexo masculino mortas: %0.2f%%\n", (float)
        criancaMortaMasc * 100 / qtdNasc);
        printf("Criancas que viveram 2 anos ou menos: %0.2f%%\n", (float)
        criancaMortaIM2Anos * 100 / qtdNasc);
        system ("pause");
        return 0;
}
```

Exercícios

1. Faça um programa que carregue um vetor com 10 elementos inteiros, calcule e mostre:

   a. A quantidade de números pares;

   b. Quais os números pares;

   c. A quantidade de números ímpares;

   d. Quais os números ímpares.

2. Escreva um programa C que utilize um vetor para armazenar a nota de 10 alunos de uma disciplina, calcule e imprima a média, e determine o número de alunos que tiveram nota superior à média.

3. Escreva um programa C que utilize um vetor para armazenar a idade de um conjunto de N indivíduos, número este fornecido pelo usuário. Calcule a média de idade deste grupo de indivíduos, e determine o número de pessoas que tiveram idade inferior à média.

4. Escreva um programa C que utilize um vetor para armazenar um conjunto de N elementos inteiros, número este fornecido pelo usuário. Verifique e imprima quantos desses elementos são positivos e quantos são negativos.

5. Ler dois vetores A e B com 10 elementos. Construir um vetor C, onde cada elemento de C é a soma do elemento de A com o seu correspondente em B, na mesma posição.

6. Fazer um programa na linguagem C para ler dois vetores X e Y com 5 elementos. Construir um vetor Z, sendo este a junção dos dois outros vetores. Desta forma Z deverá ter o dobro de elementos.

7. Ler N elementos de um vetor A. Construir um vetor B de mesmo tipo em que todo elemento de B deverá ser o quadrado do elemento de A correspondente.

8. Faça um programa em C que carregue um vetor com N elementos inteiros, onde N é um número fornecido pelo usuário, calcule e mostre:

   a. Os números múltiplos de 2;

   b. Os números múltiplos de 3;

   c. Os números múltiplos de 2 e 3.

9. Faça um programa em C que carregue um vetor com dez números reais, calcule e mostre a quantidade de números negativos e a soma dos números positivos desse vetor.

10. Faça um programa em C que carregue um vetor com 10 posições, calcule e mostre:

   a. O maior elemento do vetor e em que posição esse elemento se encontra;

   b. O menor elemento do vetor e em que posição esse elemento se encontra.

11. Faça um programa em C que leia um vetor de 12 posições de números inteiros e multiplique todos os elementos pelo maior valor do vetor. Mostre o vetor após os cálculos.

12. Faça um programa em C que receba o total das vendas de cada vendedor e armazene-as em um vetor. Receba também o percentual de comissão de cada vendedor e armazene-os em outro vetor. Receba os nomes desses vendedores e armazene-os em um terceiro vetor. Existem apenas dez vendedores. Calcule e mostre:

   a. Um relatório com os nomes dos vendedores e os valores a receber;

   b. O total das vendas de todos os vendedores;

   c. O maior valor a receber e quem o receberá;

   d. O menor valor a receber e quem o receberá.

13. Faça um programa em C para controlar o estoque de mercadorias de uma empresa. Inicialmente o programa deverá ler três vetores com dez posições cada, onde o primeiro corresponde ao código do produto, o segundo corresponde ao nome do produto e o terceiro corresponde ao total desse produto em estoque. Após, o programa deverá ler um conjunto

indeterminado de dados contendo o código de um cliente, o código do produto que este deseja comprar juntamente com a quantidade. Código do cliente igual a zero indica fim do programa. O programa deverá verificar:

a. Se o código do produto solicitado existe. Se existir, tentar atender ao pedido; caso contrário, exibir a mensagem "Código inexistente";

b. Cada pedido feito por um cliente só pode ser atendido integralmente. Caso isso não seja possível, escrever a mensagem "Não temos estoque suficiente desta mercadoria". Se puder atendê-lo, escrever a mensagem "Pedido atendido. Obrigado e volte sempre";

c. Efetuar a atualização do estoque somente se o pedido for atendido integralmente;

d. No final do programa, escrever os códigos e nomes dos produtos com seus respectivos estoques já atualizados.

14. Faça um programa em C para guardar a agenda telefônica de um determinado usuário. Inicialmente o programa deverá ler dois vetores com dez posições cada, onde o primeiro corresponde ao nome da pessoa e o segundo corresponde ao telefone da pessoa. Após, o programa deverá ordenar as informações em ordem alfabética dos nomes das pessoas. O programa deverá permite que o usuário escolha entre a busca individual de telefones, informando o nome da pessoa, um relatório com todos os nomes e respectivos telefones ou sair do programa.

Matriz

Definição de Matriz

Matriz também é conhecida como variável composta homogênea bi ou multidimensional. Composta por poder armazenar um conjunto de valores em uma mesma variável. Homogênea, pois todos os elementos são do mesmo tipo e bi ou multidimensional por precisar de apenas duas ou, respectivamente, várias dimensões para acessar seus elementos. Uma variável do tipo matriz precisa de um índice para cada uma de suas dimensões. Isso quer dizer que se trata de um conjunto de variáveis de mesmo tipo, que possuem o mesmo identificador (nome) e são alocadas sequencialmente na memória. Como as variáveis têm o mesmo nome, o

que as distingue são os índices que referenciam sua localização dentro da estrutura.

A linguagem C permite a declaração de matrizes de várias dimensões. O padrão ANSI (*American National Standard Institute*) prevê até 12 dimensões. Entretanto, o limite de dimensões fica por conta da quantidade de recursos disponíveis ao compilador. Apesar disso, as matrizes mais utilizadas possuem duas dimensões. Para cada dimensão deve ser utilizado um índice.

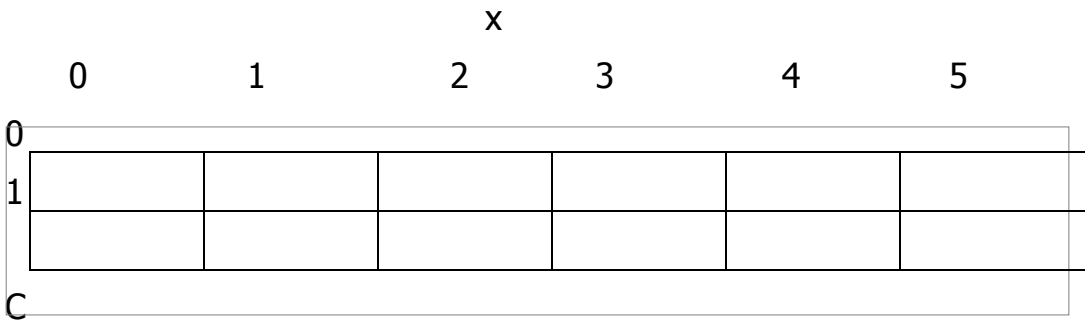
Declaração de Matriz

Tipo_dos_dados nome_variável [dimensão1] [dimensão2] [...] [dimensãoN];

Exemplo de Matriz

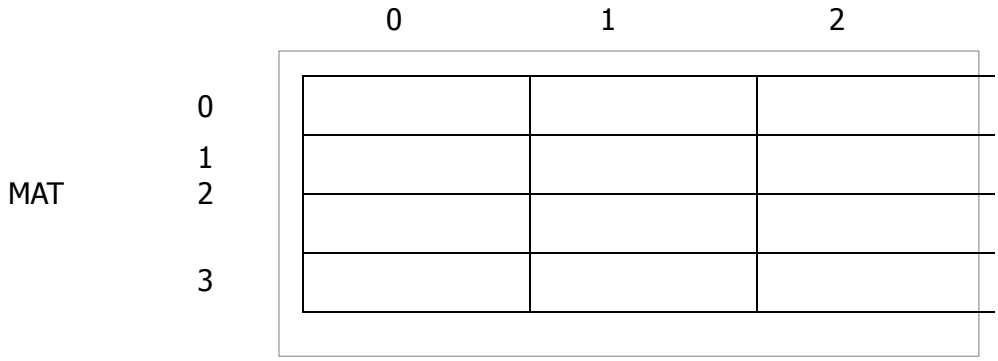
float x[2][6];

Da mesma maneira como ocorre com os vetores, os índices começam sempre em 0. Sendo assim, com a declaração anterior, criou-se uma variável chamada x contendo duas linhas (0 e 1) com seis colunas cada (0 a 5), capazes de armazenar números reais, como pode ser observado a seguir:



char MAT[4][3];

A declaração anterior criou uma variável chamada MAT contendo 4 linhas (0 a 3) com 3 colunas cada (0 a 2), capazes de armazenar símbolos, como pode ser observado a seguir:



Preenchendo uma Matriz

Preencher uma matriz significa percorrer todos os seus elementos, atribuindo-lhes um valor. Este valor pode ser recebido do usuário, por meio do teclado, ou pode ser gerado pelo programa.

No exemplo a seguir, todos os elementos da matriz bidimensional são percorridos, atribuindo-lhes valores digitados pelo usuário.

```
for (i=0 ; i<4 ; i++)
{
    for (j=0 ; j<3 ; j++)
    {
            scanf ("%s", &MAT[i][j]);
    }
}
```

Como a matriz possui 4 linhas e 3 colunas, o *for* externo deve variar de 0 a 3 (percorrendo, assim, as 4 linhas da matriz) e o *for* interno deve variar de 0 a 2 (percorrendo, assim, as 3 colunas da matriz).

Mostrando os Elementos de uma Matriz

Pode-se também percorrer todos os elementos de uma matriz acessando o seu conteúdo. Para mostrar os valores armazenados dentro de uma matriz, supondo que ela tenha sido declarada como float x[10][6], podemos executar os comandos a seguir:

```
for (i=0 ; i<10 ; i++)
{
    for (j=0 ; j<6 ; j++)
    {
            printf ("%f-", x[i][j]);
    }
```

```
    }
```

Exemplos

1 – Faça um programa que leia uma matriz 3X4 (3 linhas e 4 colunas) e, em seguida, imprima os valores em forma de matriz.

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
	int i, j;
	int mat[3][4];
	printf (“Informe os elementos inteiros da matriz:\n”);

	for (i=0 ; i<3 ; i++)
	{
		for (j=0 ; j<4 ; j++)
		{
			printf(“Elemento %d %d: “, i, j);
			scanf (“%d”, &mat[i][j]);
		}
	}
	printf(“Os elementos lidos da matriz são: “);
	for (i=0 ; i<3 ; i++)
	{
		for (j=0 ; j<4 ; j++)
		{
			printf(“%d “, mat[i][j]);
		}
		printf(“\n“);
	}
	System ("pause");
	return 0;
}
```

2 - Crie um programa que leia uma matriz do tipo real 4X2 e calcule e mostre a somatória de todos os elementos.

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
	int i, j;
	float x[4][2], soma;
	soma = 0;
```

```
        printf (“Informe os elementos em ponto flutuante da matriz:\n”);
        for (i=0 ; i<4 ; i++)
        {
                for (j=0 ; j<2 ; j++)
                {
                        printf(“Elemento %d %d: “, i, j);
                        scanf (“%f”, &x[i][j]);
                        soma += x[i][j];
                }
        }
        printf(“A somatória  dos elementos lidos da matriz é: %f“, soma);
        system ("pause");
        return 0;
}
```

3 – Elabore um programa que preencha uma matriz 3X3, calcule e mostre:

- O maior elemento da matriz e sua respectiva posição, ou seja, linha e coluna;
- O menor elemento da matriz e sua respectiva posição, ou seja, linha e coluna;

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
        int i, j, maior, menor, poslinmaior, poscolmaior, poslinmen, poscolmen;
        int mat[3][3];
        printf (“Informe os elementos inteiros da matriz:\n”);
        for (i=0 ; i<3 ; i++)
        {
                for (j=0 ; j<3 ; j++)
                {
                        printf(“Elemento %d %d: “, i, j);
                        scanf (“%d”, &mat[i][j]);
                }
        }
        maior = mat[0][0];
        poslinmaior = 0;
        poscolmaior = 0;
        menor = mat[0][0];
        poslinmen = 0;
        poscolmen = 0;
```

```
	for (i=0 ; i<3 ; i++)
	{
		for (j=0 ; j<3 ; j++)
		{
			if (mat[i][j]>maior)
			{
				maior = mat[i][j];
				poslinmaior = i;
				 poscolmaior = j;
			}else
				if (mat[i][j]<menor)
				{
					menor = mat[i][j];
					poslinmen = i;
					 poscolmen = j;
				}
		}
	}
	printf("Maior elemento da matriz: %d. ",maior);
	printf("Sua posição: %d %d.\n", poslinmaior, poscolmaior);
	printf("Menor elemento da matriz: %d. ",menor);
	printf("Sua posição: %d %d.\n", poslinmen, poscolmen);
	system ("pause");
	return 0;
}
```

4 – Crie um programa que preencha uma primeira matriz de ordem 4X3 e uma segunda matriz 3X2. O programa deverá calcular e mostrar a matriz resultante do produto matricial das duas matrizes anteriores, armazenando-a em uma terceira matriz de ordem 4X2.

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
	int i, j, k, mult,  soma;
	int A[4][3], B[3][2], C[4][2];
	printf ("Informe os elementos da primeira matriz:\n");
	for (i=0 ; i<4 ; i++)
	{
		for (j=0 ; j<3 ; j++)
		{
			printf("Elemento %d %d: ", i+1, j+1);
			scanf ("%d", &A[i][j]);
```

```
            }
        }
        printf (“Informe os elementos da segunda matriz:\n”);
        for (i=0 ; i<3 ; i++)
        {
            for (j=0 ; j<2 ; j++)
            {
                printf(“Elemento %d %d: “, i+1, j+1);
                scanf (“%d”, &B[i][j]);
            }
        }
        for (i=0 ; i<4 ; i++)
        {
            for (j=0 ; j<2 ; j++)
            {
                soma = 0;
                for (k=0 ; k<3 ; k++)
                {
                    mult = A[i][k] * B[k][j];
                    soma = soma + mult;
                }
                C[i][j] = soma;
            }

        }
        printf(“Os elementos da matriz resultante são: \n“);
        for (i=0 ; i<4 ; i++)
        {
            for (j=0 ; j<2 ; j++)
            {
                printf(“%d “, C[i][j]);
            }
            printf(“\n“);
        }
        system ("pause");
        return 0;
}
```

5 – Faça um programa que use uma função para receber uma matriz lida, e que retorne a média dos elementos dessa matriz para o programa principal.

```
#include <stdio.h>
```

```
#include <stdlib.h>

float media( int a[][], int lin, int col )
{
	int i, j;
	float soma;

	soma = 0;

	for ( i = 0; i < lin; i++ )
		for ( j = 0; j < col; j++ )
			soma += a[i][j];

	return( soma / (lin * col) );
}

int main()
{
	int A[3][3];
	int i, j;

	for (i = 0; i < 3; i++ )
		for ( j = 0; j < 3; j++ )
		{
			A[i][j] = i + j;
		}

	printf("Media = %5.2f\n\n", media(A,3,3));
	system ("pause");
	return 0;
}
```

Exercícios

1. Faça um programa em C que carregue uma matriz 4x3 com números inteiros, calcule e mostre a quantidade de elementos pares entre esses elementos.

2. Faça um programa que carregue uma matriz 3x4 com números inteiros, calcule e mostre a quantidade de elementos entre 15 e 20.

3. Faça um programa que preencha uma matriz 4X3 com números inteiros, calcule e mostre:

    a. A quantidade de elementos entre 10 e 20 em cada linha;

b. A média dos elementos pares da matriz

4. Faça um programa que carregue uma matriz 5x3 com números inteiros, calcule e mostre a somatória dos elementos de cada linha com vetor.

5. Crie um programa que preencha uma matriz 4X4 com números inteiros e some cada uma das colunas, armazenando o resultado da soma em um vetor. A seguir, o programa deverá multiplicar cada elemento da matriz pela soma da coluna da matriz e mostrar a matriz resultante.

6. A matriz a seguir contém exemplos de vários itens que estão estocados em diversos armazéns de uma companhia. É fornecido, também, um vetor com o custo de cada um dos produtos armazenados.

	Produto 1 (unid.)	Produto 2 (unid.)	Produto 3 (unid.)
Armazém 1	12	37	37
Armazém 2	14	42	31
Armazém 3	20	22	25

Custo (R$)	26	40	30

Fazer um programa que:

a. Leia o estoque inicial e o custo;

b. Determine e imprima quantos itens estão armazenados em cada armazém;

c. O custo total de:

- Cada produto em cada armazém;
- Estoque em cada armazém.

Modularização (Sub-Rotinas)

O conceito de modularização é usado em programação para dividir o programa em partes menores, que chamamos de subprogramas ou sub-rotinas. Dividindo um programa em módulos, podemos chamar a sub-rotina quantas vezes quisermos, ao invés de reescrevermos esse mesmo código inúmeras vezes. Assim, basta chamá-lo por meio de seu nome passando ou não alguma informação.

Outra vantagem de modularizar é aumentar a produtividade, evitando reescrever código e reduzindo o universo de correção de possíveis erros.

Modularização é implementada na maioria das linguagens por meio de funções e procedimentos. Na linguagem C só existe função. Aqui neste tópico só utilizaremos programas, ao invés de fluxogramas e algoritmos.

Os programas em geral são executados linearmente, uma linha após a outra, até o fim. Contudo, quando são utilizadas sub-rotinas, podemos desviar a execução dos programas por meio de uma chamada à uma função.

Um programa escrito na linguagem C tem, no mínimo, uma função chamada main, por onde toda execução começa. Existem também muitas outras funções pré-definidas na linguagem C, como por exemplo clrscr (limpa a tela), getch e gets (entrada de dados). Estas funções são adicionadas aos programas pela diretiva #include, no momento da "linkedição", processo de construção efetiva do código executável.

Além disso, o usuário também pode criar quantas funções quiser, dependendo do problema que estiver sendo resolvido. As funções às vezes precisam receber valor(es) externo(s), chamados parâmetros, e também podem devolver algum valor produzido para quem a chamou, denominado retorno.

Os parâmetros são passados por uma lista de variáveis colocadas dentro dos parênteses, logo após o nome da função. Caso haja retorno, a última linha da função deverá incluir o comando return, seguido do valor ou variável que será devolvido a quem chamou a função. O tipo do valor retornado deverá ser exatamente igual ao do tipo informado antes do nome da função. Caso não haja retorno, deverá ser digitada a palavra void (vazio ou sem retorno).

Função sem Passagem de Parâmetros e sem Retorno

O tipo mais simples de função é aquele que não recebe nenhuma informação no momento de sua chamada e que também não repassa nenhum valor para quem a chamou. Exemplo para somar dois valores:

```
#include <stdio.h>
#include <stdlib.h>
void soma()
{
        int a, b, s;
        printf("Informe o primeiro valor:");
        scanf("%d",&a);
        printf("Informe o segundo valor:");
        scanf("%d",&b);
        s = a + b;
        printf("Soma = %d\n", s);
}
int main()
```

```
{
        soma();
        system ("pause");
        return 0;
}
```

Função com Passagem de Parâmetros e sem Retorno

O segundo tipo de função é representado por aquelas que recebem valores no momento em que são chamadas (parâmetros), mas que, no final, não devolvem valor para quem as chamou (retorno). Exemplo para cálculo da multiplicação:

```
#include <stdio.h>
#include <stdlib.h>
void multiplica(float x, float y)
{
        float prod;
        prod = x * y;
        printf("Produto = %f", prod);
}
int main()
{
        float a, b;
        printf("Informe o primeiro valor:");
        scanf("%f",&a);
        printf("Informe o segundo valor:");
        scanf("%f",&b);
        multiplica (a, b);
        system ("pause");
        return 0;
}
```

Função sem Passagem de Parâmetros e com Retorno

O 3º tipo de função é representado por aquelas que não recebem valores no momento em que são chamadas (parâmetros), mas que, no final, devolvem um valor para quem as chamou (retorno). Exemplo para calcular a média de dois valores:

```
#include <stdio.h>
#include <stdlib.h>
float calcula_media()
{
        float media, n1, n2;
        printf("Informe o primeiro valor:");
```

```
        scanf("%f",&n1);
        printf("Informe o segundo valor:");
        scanf("%f",&n2);
        media = (n1 + n2)/2;
        return media;
}
int main()
{
        float resp;
        resp = calcula_media();
        printf("A média é: %f", resp);
        system ("pause");
        return 0;
}
```

Função com Passagem de Parâmetros e com Retorno

Por fim, temos as funções que recebem valores no momento em que são chamadas (parâmetros) e que, no final, devolvem um valor para quem as chamou (retorno). Exemplo de divisão de dois valores:

```
#include <stdio.h>
#include <stdlib.h>
float divisao(float dividendo, float divisor);

int main()
{
        float a, b, resp;
        printf("Informe o valor do dividendo:");
        scanf("%f",&a);
        printf("Informe o valor do divisor:");
        scanf("%f",&b);
        resp = divisao (a, b);
        printf("O resultado da divisão é: %f", resp);
        system ("pause");
        return 0;
}

float divisao(float dividendo, float divisor)
{
        float quoc;
        quoc = dividendo / divisor;
        return quoc;

}
```

Passagem de Parâmetros Por Valor

Passagem de parâmetros por valor significa que a função trabalhará com cópia dos valores passados no momento da chamada. Ou seja, as alterações feitas nos parâmetros formais, que estão dentro função, não se refletem nos parâmetros atuais, que são os valores externos à função. Na chamada da função, o valor do parâmetro atual é copiado no parâmetro formal. Assim, quando a passagem é por valor, isto significa que o parâmetro é de entrada.

Todos os exemplos que fizemos anteriormente já estavam fazendo uso dessa técnica de passagem, sem mesmo sabermos conceitualmente o seu significado. Exemplo de passagem por valor em uma função que realiza a soma do dobro de dois valores:

```
#include <stdio.h>
#include <stdlib.h>

int soma_dobro (int a , int b)
{
	int soma;
	a = 2*a;
	b = 2*b;
	soma = a + b;
	return soma;
}

int main()
{
	int x, y, resp;
	printf("Digite o primeiro numero");
	scanf("%d",&x);
	printf("Digite o segundo numero");
	scanf("%d",&y);
	resp = soma_dobro(x, y);
	printf("A soma do dobro de %d e %d eh: %d", x, y, resp);
	system ("pause");
	return 0;
}
```

Passagem de Parâmetros por Referência

Passagem de parâmetros por referência significa que os parâmetros passados para a função correspondem a **endereços de memória** ocupados por variáveis. Portanto, sempre que necessário for acessar determinado valor, isso será feito por **referência**, ou seja, apontando seu endereço.

Neste tipo de passagem, toda alteração feita num parâmetro formal (interno) corresponde a mesma alteração feita no seu parâmetro atual (externo) associado. Isto é conseguido pela passagem do endereço do parâmetro atual no momento da chamada. Quando a passagem é feita por referência significa que o parâmetro é de entrada-saída.

Portanto, quando criamos uma função, temos que observar como classificar os parâmetros entre os tipos mencionados (valor ou referência) para evitar os efeitos colaterais decorrentes do modo de passagem utilizado.

Na linguagem C, na passagem por referência, os parâmetros atuais são precedidos pelo símbolo "&" e os parâmetros formais são precedidos pelo símbolo "*".

```
#include <stdio.h>
#include <stdlib.h>

int soma_dobro(int *a , int *b)
{
        int soma;
        *a = 2*(*a);
        *b = 2*(*b);
        soma = *a + *b;
        return soma;
}

int main()
{
        int x, y, resp;
        printf("Digite o primeiro numero");
        scanf("%d",&x);
        printf("Digite o segundo numero");
        scanf("%d",&y);
        resp = soma_dobro(&x, &y);
        printf("A soma do dobro de %d e %d eh: %d", x, y, resp);
        system ("pause");
        return 0;
}
```

Exercícios de Funções

1. Escreva uma função que receba como parâmetro o raio de uma esfera, calcule e mostre no programa principal o seu volume: $v = 4/3*\prod R^3$.

2. Escreva uma função que recebe um inteiro positivo *m* e devolve 1 se *m* é par e 0 (zero) em caso contrário.
3. Crie uma função em linguagem C que receba 2 números e retorne o maior valor.
4. Crie uma função em linguagem C que receba 3 números e retorne o menor valor.
5. Faça uma função que recebe a idade de uma pessoa em anos, meses e dias e retorna essa idade expressa em dias.
6. Faça uma função que recebe, por parâmetro, a altura (alt) e o sexo de uma pessoa e retorna o seu peso ideal. Para homens, calcular o peso ideal usando a fórmula peso ideal = 72.7 x alt - 58 e ,para mulheres, peso ideal = 62.1 x alt - 44.7.
7. Faça uma função que recebe 3 valores inteiros por parâmetro e retorna-os ordenados em ordem crescente.
8. Escreva uma função que receba dois inteiros não-negativos *n* e retorne a soma dos números inteiros existentes entre eles.
9. Escreva uma função que recebe como parâmetros três números inteiros *a, b* e *c,* sendo *a* maior que 1. A função deverá somar todos os inteiros entre b e c que sejam divisíveis por a (inclusive b e c) e retornar o resultado para o programa principal.

Estruturas (Structs - Registros)

Como já dito no tópico referente aos tipos básicos, a linguagem C possui quatro tipos básicos de dados: *int*, *float*, *double* e *char*. Estes tipos representam um único item de informação e, baseado neles, podemos definir tipos complexos que possibilitam agrupar um conjunto de variáveis de tipos diferentes sob um único nome (variáveis compostas heterogêneas).

Estruturas (*structs*) são tipos de variáveis que agrupam dados geralmente desiguais, enquanto vetores e matrizes são tipos de variáveis que agrupam dados similares. Os itens de dados de uma estrutura são

chamados de membros, enquanto os itens de um vetor ou matriz são chamados de elementos.

Criando Novos Tipos de Dados com *struct*

Por meio da palavra-chave *struct,* definimos um novo tipo de dado. Definir um novo tipo de dado significa informar ao compilador o seu nome, o seu tamanho em bytes e o formato em que ele deve ser armazenado e recuperado da memória.

Após ter sido definido, o novo tipo existe e pode ser utilizado para criar variáveis de modo similar a qualquer tipo simples.

O exemplo a seguir cria um tipo de dado que pode armazenar as informações de um aluno.

```
#include <stdio.h>
#include <stdlib.h>

struct aluno  // Definição da estrutura
{
        int nmat;              // Número de Matrícula do aluno
        float nota[3];         // Nota do aluno
        float media;           // Média do aluno
};

int main()
{
        struct aluno ana;              // Declara variável do tipo aluno

        ana.nmat = 456;
        ana.nota[0] = 7.5;
        ana.nota[1] = 5.2;
        ana.nota[2] = 8.4;
        ana.media = (ana.nota[0]+ana.nota[1]+ana.nota[2])/3.0;
        printf("\nMatrícula: %d", ana.nmat);
        printf("\nMédia: %f", ana.media);
        system ("pause");
        return 0;
}
```

Definir uma estrutura não cria nenhuma variável, somente informa ao compilador as características de um novo tipo de dado. Não há nenhuma reserva de memória.

A palavra *struct* indica que um novo tipo de dado está sendo definido e a palavra aluno será o seu nome.

No nosso exemplo, definimos o tipo aluno antes de *main*(), o que permite um acesso global a todas as funções definidas no programa. Poderíamos colocar essa definição dentro de uma função, restringindo o acesso às instruções do mesmo bloco e escritas dentro dela.

Exemplos

1 – Criar uma estrutura para o nome de uma pessoa dividida em três partes: primeiro nome, nome do meio e último nome.

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
        struct
        {
            char first [15];
            char midinit;
            char last [20];
        }  sname,  ename;

        int i;

        printf("Digite seu primeiro nome (menor do que 15 caracteres: ");
// A leitura de uma string é feita diretamente sem a necessidade de &
        scanf("%s", ename.first);

        printf("Digite seu sobrenome do meio (com apenas uma letra):");
/* Para leitura de um caracter, há a necessidade de utilizar & e a
        formatação utiliza %s também */
        scanf("%s", &ename.midinit);

// Leitura do último sobrenome, igual à 1ª leitura
        printf("Digite seu ultimo sobrenome: ");
        scanf("%s", ename.last);

//Impresão do nome completo
        for ( i = 0; (i < 15) && (ename.first[i] != '\0' ); i++ )
```

```
        {
                printf("%c", ename.first[i]);
        }
        printf("%c", ' ' );

        printf("%c", ename.midinit);

        printf("%c", ' ' );
        for ( i = 0; (i < 20) && (ename.last[i] != '\0' ); i++ )
        {
                printf("%c", ename.last[i]);
        }
        system ("pause");
        return 0;
}
```

2 – Veja uma outra maneira de utilizar estrutura para nome de uma pessoa, semelhante ao exemplo anterior.

```
#include <stdio.h>
#include <stdlib.h>

typedef struct
{
       char first[10];
       char middle[10];
       char last [10];
} NAMETYPE;

int writename(NAMETYPE *name)
{
   int count, i;
   count = 0;

        for ( i = 0; (i < 10) && (name->first[i] != '\0' ); i++ )
        {
                printf("%c", name->first[i]);
                count++;
        }
        printf("%c", ' ' );
        count++;
        for ( i = 0; (i < 10) && (name->middle[i] != '\0' ); i++ )
        {
```

```
            printf("%c", name->middle[i]);
            count++;
        }
        printf("%c", ' ' );
        count++;
        for ( i = 0; (i < 10) && (name->last[i] != '\0' ); i++ )
        {
            printf("%c", name->last[i]);
            count++;
        }
        return(count-2);
}

int main()
{
   NAMETYPE nome;

   printf("Digite o primeiro nome:");
   scanf ("%s", &nome->first);
   printf("Digite o nome do meio:");
   scanf ("%s", &nome->middle);
   printf("Digite o ultimo nome:");
   scanf ("%s", &nome->last);
   printf("\nQuantidade de caracteres no nome = %d\n", writename(&nome));
   system ("pause");
   return 0;
}
```

3 – Exemplo de um vetor de *struct* com vetor como membro interno.

```
#include <stdio.h>
#include <stdlib.h>
typedef struct        // Definição da estrutura
{
        int nmat;              // Número de Matrícula
        float nota[3];
        float media;
}aluno;
```

```
int main()
{
    aluno a[10];              // Declara variável do tipo aluno
    int i,j;
    float soma;
    for (i=0; i<5; i++)
    {
        printf("Informe o numero de matricula do %dº aluno: ", i+1);
        scanf("%d",&a[i].nmat);
        soma=0;
        for (j=0; j<3; j++)
        {
            printf("Informe a nota %d do aluno %d: ", j+1, i+1);
            scanf("%f",&a[i].nota[j]);
            soma += a[i].nota[j];
        }
        a[i].media = (soma/3.0);
    }
    for (i=0; i<5; i++)
    {
       printf("\nMatricula: %d", a[i].nmat);
       for (j=0; j<3; j++)
           printf("\n     nota %d do aluno %d: %f", j+1, i+1, a[i].nota[j]);
       printf("\n     Media: %f", a[i].media);
    }
printf("\n);
system ("pause");
return 0;
}
```

4 - Criar uma estrutura chamada automóvel com os seguintes membros: motor (inteiro), chassi (inteiro), cor (*string*), modelo (*string*), ano (inteiro), preco (*float*). O programa deve fazer a leitura dos membros da estrutura e criar:

- Uma função que calcule quantos anos tem o carro
- Uma função que reajuste o preço do carro em 10%
- Uma função com um laço que aumente a rotação do motor em 1000 RPM
- Uma função que mostre o valor dos membros da estrutura.

```
#include <stdio.h>
#include <stdlib.h>
typedef struct {
	int iMotor ;
	int iChassi;
	char cCor[10];
	char cModelo[30];
	int iAno ;
	float fPreco;
}  oVeiculo;

void Consultar(oVeiculo sVeiculo)
{
     printf("\n");
     printf("********** Dados do Automovel ********** \n");
     printf("\n");
     printf("Rotacoes do Motor do Carro.: ");
     printf("%d\n",sVeiculo.iMotor);
     printf("Chassi do Carro: ");
     printf("%d\n",sVeiculo.iChassi);
     printf("Cor do Carro...: ");
     printf("%s\n",sVeiculo.cCor);
     printf("Modelo do Carro: ");
     printf("%s\n",sVeiculo.cModelo);
     printf("Ano do Carro...: ");
     printf("%d\n",sVeiculo.iAno);
     printf("Preço do Carro.: ");
     printf("%f\n",sVeiculo.fPreco);
     system("PAUSE");
}
int Rotacao(int iMotor)
{
   int i, iResult;
   iResult = iMotor;
   for (i = 0; i < 1000; i++)
   {
        iResult = iResult +1;
   }
   printf("\n");
   printf("A rotacao e %d",iResult)  ;
   printf("\n");
   system("PAUSE");
   return iResult;
}
```

```
void CalcAnos(int iData)
{
     printf("\n");
     printf("O Carro Possui %d anos",(2022 - iData))  ;
     printf("\n");
     system("PAUSE");
}
float CalcPrecos(float fPreco)
{
      printf("\n");
      printf("O Valor do carro e %f reais",(fPreco + ((fPreco * 10)/100)))  ;
      printf("\n");
      system("PAUSE");
      return fPreco + ((fPreco * 10)/100);
}
int main ()
{
     oVeiculo tVeiculo;
     int iOpcao;

    printf("********** Dados do Automovel ********** \n");
    printf("\n");
    printf("Informe a quantidade de rotacoes do motor do Carro.: ");
    scanf("%d",&tVeiculo.iMotor);
    printf("Informe o Chassi do Carro: ");
    scanf("%d",&tVeiculo.iChassi);
    printf("Informe a Cor do Carro...: ");
    scanf("%s",&tVeiculo.cCor);
    printf("Informe o Modelo do Carro: ");
    scanf("%s",&tVeiculo.cModelo);
    printf("Informe o Ano do Carro...: ");
    scanf("%d",&tVeiculo.iAno);
    printf("Informe o Preço do Carro.: ");
    scanf("%f",&tVeiculo.fPreco);
    printf("\n");
    do
    {
            system("cls");
            Consultar(tVeiculo);
            printf("\[1] – Quantidade de Anos do carro [2] – Reajusta Preco em
10% [3] - Aumento de Rotacao em 1000 rpm[4] - Consulta\n");
            printf("\n");
            printf("Opcao.....................: ");
            scanf("%d",&iOpcao);
```

```
            if (iOpcao == 1)
            {
              CalcAnos(tVeiculo.iAno);
            }
            else if (iOpcao == 2)
            {
                tVeiculo.fPreco = CalcPrecos(tVeiculo.fPreco);
            }
            else if (iOpcao == 3)
            {
                tVeiculo.iMotor = Rotacao(tVeiculo.iMotor);
            }
            else if (iOpcao == 4)
            {
                Consultar(tVeiculo);
                system("PAUSE");
            }
            if (iOpcao == 0)
            {
                  break;
            }
    }while(iOpcao <= 4);
system ("pause");
return 0;
}
```

5 – Exemplo de estruturas aninhadas

```
#include <stdio.h>
#include <stdlib.h>

typedef struct {
        char first[15];
        char middle[15];
        char last [15];
} nametype;

typedef struct {
    char straddr [40];
    char city [15];
    char state [2];
    char zip [10];
} addrtype;

typedef struct {
```

```
    nametype name;
    addrtype address;
} nmadtype;

void writename(nmadtype name_address)
{
        int  i;
        printf("Nome Completo: ");
        for (i = 0; (i < 15) && (name_address.name.first[i] != '\0' ); i++) {
                printf("%c", name_address.name.first[i]);
        }
        printf(" ");
        for (i = 0; (i < 15) && (name_address.name.middle[i] != '\0' ); i++) {
                printf("%c", name_address.name.middle[i]);
        }
        printf(" ");
        for (i = 0; (i < 15) && (name_address.name.last[i] != '\0' ); i++) {
                printf("%c", name_address.name.last[i]);
        }
       printf("\nRua: ");
        for (i = 0; (i < 40) && (name_address.address.straddr[i] != '\0' ); i++) {
                printf("%c", name_address.address.straddr[i]);
        }
       printf("\nCidade: ");
        for (i = 0; (i < 15) && (name_address.address.city[i] != '\0' ); i++) {
                printf("%c", name_address.address.city[i]);
        }
       printf("\nEstado: ");
        for (i = 0; (i < 2) && (name_address.address.state[i] != '\0' ); i++) {
                printf("%c", name_address.address.state[i]);
        }
        printf("\nCEP: ");
        for (i = 0; (i < 10) && (name_address.address.zip[i] != '\0' ); i++) {
                printf("%c", name_address.address.zip[i]);
        }
        printf("\n");
}

int main()
{
        nmadtype nmad1, nmad2;

        printf("Digite o primeiro nome:");
        scanf ("%s", nmad1.name.first);
```

```
        printf("Digite o nome do meio:");
        scanf ("%s", nmad1.name.middle);
        printf("Digite o ultimo nome:");
        scanf ("%s", nmad1.name.last);

        printf("Digite o endereço:");
        scanf ("%s", nmad1.address.straddr);
        printf("Digite a cidade:");
        scanf ("%s", nmad1.address.city);
        printf("Digite o estado:");
        scanf ("%s", nmad1.address.state);
        printf("Digite o CEP:");
        scanf ("%s", nmad1.address.zip);

        writename(nmad1);
        system ("pause");
        return 0;
}
```

6 - Exemplo de estruturas aninhadas

```
#include <stdio.h>
#include <stdlib.h>

typedef struct {
   char first [15];
   char middle[15];
   char last [20];
}nametype;

typedef struct {
   char straddr [40];
   char city [10];
   char state [2];
   char zip [5];
}addrtype;

typedef struct {
    nametype name;
    addrtype address;
}nmadtype;

typedef struct {
   int month;
   int day;
```

```
    int year;
}date;

typedef struct {
    char deptno [2];
    char jobtitle [20];
}position;

typedef struct {
     nmadtype nameaddr;
     position job;
     float salary;
     date datehired;
}employee;

typedef struct{
    nmadtype nmad;
    float gpindex;   //Índice de pontos na graduação
    int credits;
    date dateadm;
}student;


int main()
{
        employee e;
        student s;
         int  i;

          printf("Digite o primeiro nome do empregado: ");
          scanf ("%s", e.nameaddr.name.first);
          printf("Digite o nome do meio: ");
          scanf ("%s", e.nameaddr.name.middle);
          printf("Digite o ultimo nome: ");
          scanf ("%s", e.nameaddr.name.last);
          printf("Digite a rua: ");
          scanf ("%s", e.nameaddr.address.straddr);
          printf("Digite a cidade ");
          scanf ("%s", e.nameaddr.address.city);
          printf("Digite o estado ");
          scanf ("%s", e.nameaddr.address.state);
          printf("Digite o CEP");
          scanf ("%s", e.nameaddr.address.zip);
          printf("Digite o numero do depto do empregado: ");
```

```
        scanf ("%s", e.job.deptno);
        printf("Digite o cargo do empregado: ");
        scanf ("%s", e.job.jobtitle);
        printf("Digite o salario: ");
        scanf ("%f", &e.salary);
        printf("Digite o ano de contratacao: ");
        scanf ("%d", &e.datehired.year);
        printf("Digite o mes de contratacao(01-12): ");
        scanf ("%d", &e.datehired.month);
        printf("Digite o dia de contratacao(01-31): ");
        scanf ("%d", &e.datehired.day);

        printf("Digite o primeiro nome do estudante: ");
        scanf ("%s", s.nmad.name.first);
        printf("Digite o nome do meio: ");
        scanf ("%s", s.nmad.name.middle);
        printf("Digite o ultimo nome: ");
        scanf ("%s", s.nmad.name.last);
        printf("Digite a rua: ");
        scanf ("%s", s.nmad.address.straddr);
        printf("Digite a cidade ");
        scanf ("%s", s.nmad.address.city);
        printf("Digite o estado ");
        scanf ("%s", s.nmad.address.state);
        printf("Digite o CEP");
        scanf ("%s", s.nmad.address.zip);
        printf("Digite o indice de graduacao");
        scanf ("%f", &s.gpindex);

            if (*e.nameaddr.name.middle  ==  *s.nmad.name.middle) &&
             (*e.nameaddr.name.last  ==  *s.nmad.name.last))
             if  (s.gpindex  >  3.0)
                   e.salary *=  1.10;

      printf(       if  ((*e.nameaddr.name.first  ==  *s.nmad.name.first) &&
"Nome completo do empregado: ");
        for (i = 0; (i < 15) && (e.nameaddr.name.first[i] != '\0' ); i++) {
                printf("%c", e.nameaddr.name.first[i]);
        }
        printf(" ");
        for (i = 0; (i < 15) && (e.nameaddr.name.middle[i] != '\0' ); i++) {
                printf("%c", e.nameaddr.name.midle[i]);
        }
        printf(" ");
```

```
        for (i = 0; (i < 20) && (e.nameaddr.name.last[i] != '\0' ); i++) {
                printf("%c", e.nameaddr.name.last[i]);
        }
       printf("\nNumero do depto do empregado: ");
        for (i = 0; (i < 2) && (e.job.deptno[i] != '\0' ); i++) {
                printf("%c", e.job.deptno[i]);
        }
       printf("\nCargo do empregado: ");
        for (i = 0; (i < 20) && (e.job.jobtitle[i] != '\0' ); i++) {
                printf("%c", e.job.jobtitle[i]);
        }
        printf("\nSalario: ");
        printf("%6.2f", e.salary);
        printf("\nData de contratacao:
%d/%d/%d",e.datehired.day,e.datehired.month,e.datehired.year);
        printf("\n");
        system ("pause");
        return 0;
}
```

7 - Exemplo de estruturas aninhadas

```
# include <stdio.h>
#include <stdlib.h>

typedef struct {
   char first [15];
   char middle[15];
   char last [20];
}nametype;

typedef struct {
   char straddr [40];
   char city [10];
   char state [2];
   char zip [5];
}addrtype;

typedef struct {
    nametype name;
    addrtype address;
}nmadtype;

typedef struct {
```

```
    int month;
    int day;
    int year;
}date;

typedef struct {
    char deptno [2];
    char jobtitle [20];
}position;

typedef struct {
     nmadtype nameaddr;
     position job;
     float salary;
     date datehired;
}employee;

typedef struct{
    nmadtype nmad;
    float gpindex;   //Índice de pontos na graduação
    int credits;
    date dateadm;
}student;

void read_employee(employee *emp)
{
        printf("Digite o primeiro nome do empregado: ");
        scanf ("%s", emp->nameaddr.name.first);
        printf("Digite o nome do meio: ");
        scanf ("%s", emp->nameaddr.name.middle);
        printf("Digite o ultimo nome: ");
        scanf ("%s", emp->nameaddr.name.last);
        printf("Digite o codigo do departamento do empregado: ");
        scanf ("%s", emp->job.deptno);
        printf("Digite o cargo do empregado: ");
        scanf ("%s", emp->job.jobtitle);
        printf("Digite o salario: ");
        scanf ("%f", &emp->salary);
        printf("Digite o ano de contratacao: ");
        scanf ("%d", &emp->datehired.year);
        printf("Digite o mes de contratacao: ");
        scanf ("%d", &emp->datehired.month);
        printf("Digite o dia de contratacao: ");
        scanf ("%d", &emp->datehired.day);
```

```
}

void read_student(student *est)
{
        printf("Digite o primeiro nome do estudante: ");
         scanf ("%s", est->nmad.name.first);
         printf("Digite o nome do meio: ");
         scanf ("%s", est->nmad.name.middle);
         printf("Digite o ultimo nome: ");
         scanf ("%s", est->nmad.name.last);
         printf("Digite a rua: ");
         scanf ("%s", est->nmad.address.straddr);
         printf("Digite a cidade: ");
         scanf ("%s", est->nmad.address.city);
         printf("Digite o estado: ");
         scanf ("%s", est->nmad.address.state);
         printf("Digite o CEP: ");
         scanf ("%s", est->nmad.address.zip);
         printf("Digite o indice de graduacao: ");
         scanf ("%f", &est->gpindex);
}

void write_employee(employee emp)
{
         int  i;
         printf("Nome completo do empregado: ");
         for (i = 0; (i < 15) && (emp.nameaddr.name.first[i] != '\0' ); i++) {
                  printf("%c", emp.nameaddr.name.first[i]);
         }
         printf(" ");
         for (i = 0; (i < 15) && (emp.nameaddr.name.middle[i] != '\0' ); i++) {
                  printf("%c", emp.nameaddr.name.middle[i]);
         }
         printf(" ");
         for (i = 0; (i < 20) && (emp.nameaddr.name.last[i] != '\0' ); i++) {
                  printf("%c", emp.nameaddr.name.last[i]);
         }
         printf("\nNumero do depto do empregado: ");
         for (i = 0; (i < 2) && (emp.job.deptno[i] != '\0' ); i++) {
                  printf("%c", emp.job.deptno[i]);
         }
         printf("\nCargo do empregado: ");
         for (i = 0; (i < 20) && (emp.job.jobtitle[i] != '\0' ); i++) {
                  printf("%c", emp.job.jobtitle[i]);
```

```
        }
        printf("\nSalario: ");
        printf("%6.2f", emp.salary);
        printf("\nData de contratacao: %d/%d/%d",emp.datehired.day,
emp.datehired.month, emp.datehired.year);
        printf("\n");
}

int main()
{
       employee e;
       student s;
       int  i;

       read_employee(&e);
       read_student(&s);

       if  ((*e.nameaddr.name.first  ==  *s.nmad.name.first) &&
              (*e.nameaddr.name.middle  ==  *s.nmad.name.middle) &&
              (*e.nameaddr.name.last  ==  *s.nmad.name.last))
              if  (s.gpindex  >  3.0)
                    e.salary *=  1.10;

      write_employee(e);
      system ("pause");
      return 0;
}
```

Exercícios

1. Crie uma estrutura em C com pelo menos 3 membros e com no mínimo dois tipos diferentes entre estes membros para armazenar informações de um cliente (nome, idade, sexo, salário). Você deve fazer a leitura destes valores, calcular a idade em meses do indivíduo imprimindo-a, juntamente com os valores dos membros da estrutura.

2. Crie uma estrutura em C com pelo menos 3 membros e com no mínimo dois tipos diferentes entre estes membros para armazenar informações de um time de futebol (nome, número de vitórias, número de empates, colocação). Você deve fazer a leitura destes valores, calcular o número de pontos do time (V=3pts, E=1pt) imprimindo-o, juntamente com os valores dos membros da estrutura.

3. Crie uma estrutura em C com pelo menos 3 membros e com no mínimo dois tipos diferentes entre estes membros para armazenar informações de um carro (Nº chassi, Ano, Modelo, Cor e Placa). Você deve fazer a leitura destes valores juntamente com o ano atual e calcular a idade do carro em anos, meses e dias. Você deve mostrar tais informações juntamente com os valores dos membros da estrutura.

4. A prefeitura de uma cidade fez uma pesquisa entre seus habitantes, coletando dados sobre salário, idade e número de filhos. Faça um programa, usando estrutura, que leia esses dados de um número indeterminado de pessoas, calcule e mostre:

- A média de salário da população;
- A média do número de filhos;
- O maior salário;
- O percentual de mulheres com salário superior a R$ 1.000,00.

5. Crie uma estrutura (*struct*) em C com pelo menos 4 membros e com no mínimo dois tipos diferentes entre estes membros para armazenar informações de um cliente (nome, idade, sexo, salário). O membro nome deve ser desmembrado em uma outra estrutura dividida em três partes. Você deve fazer a leitura destes valores, calcular a idade em meses do indivíduo, imprimindo-a juntamente com os valores de todos os membros da estrutura.

6. Desenvolva um programa em C com duas estruturas (*structs*): Nome e Funcionário. A estrutura Nome deve ter o nome dividido em três partes: início, meio e fim. A estrutura Funcionário deve fazer uso da estrutura anterior e ainda possuir os membros endereço, salário, tempo de serviço e CPF. Faça a leitura de valores para os membros da estrutura. Você deve usar uma porcentagem fornecida pelo usuário para calcular o aumento do salário. Após esse cálculo, se o valor reajustado do salário não ultrapassar R$ 1000,00 seu novo salário será acrescido de R$ 200, caso contrário seu bônus será de R$ 100,00. Dentro dessa estrutura você deverá fazer o cálculo de uma proporção salário/tempo de serviço. Se essa proporção for superior a 100, você deve imprimir que este funcionário é da categoria C, senão ele é da categoria A. Faça a impressão dos valores dos membros da estrutura.

7. Faça um programa em C com estrutura com as seguintes características

a. Criar uma estrutura chamada Data de Fabricação que possui três membros: dia, mês e ano.
b. Criar uma outra estrutura chamada automóvel com os seguintes membros: rotação do motor (inteiro), chassi (inteiro), cor (string), modelo (string), preço (float). Esta estrutura ainda possui outro membro (data) que é do tipo da estrutura do item a).
c. O programa deverá:
    i. fazer a leitura dos membros da estrutura
    ii. calcular quantos anos tem o carro;
    iii. reajustar o preço do carro em 10%;
    iv. criar um laço que aumente a rotação do motor em 1000 rpm;
    v. mostrar o valor dos membros da estrutura.