Algoritmos 1

Conceitos Intermediários

Prof. André Luiz Marasca

1 O que é um algoritmo?

Algoritmo é uma sequência finita de **instruções** bem definidas e não ambíguas, que necessitam de um intervalo de tempo finito e uma quantidade de esforço finita.

2 Vetores

Na matemática, um vetor é uma matriz de uma só dimensão, podendo ser 1 x N ou ainda N x 1. Na computação utilizamos vetores para armazenar valores de tipos homogêneos, ou seja, armazenar valores de mesmo tipo.

Imagine um caso onde é necessário realizar o cálculo da média e desvio padrão de um conjunto de números. O cálculo da média é simples, não é necessário armazenar os valores. Contudo, o cálculo do desvio padrão precisa do valor da média, dessa forma, seria necessário que os valores lidos estivessem armazenados.

É de se imaginar que com o que sabemos até então não é o suficiente para isso, por exemplo, queremos um programa genérico, que realiza a média e desvio padrão para qualquer número de amostras que tivermos. Esta é uma tarefa praticamente impossível com o conhecimento limitado apresentado até então. A solução desse problema é: uso de vetores.

Um vetor nada mais é do que um conjunto de variáveis que são declaradas utilizando o mesmo nome, porém, elas são indexadas. Para declaração de um vetor, utiliza-se o seguinte comando: **tipo** nome_do_vetor[**tamanho_do_vetor**];

Abaixo são mostrados alguns exemplos de declaração de vetores:

```
char vetor_string[50];
int vetor_inteiro[10];
unsigned int vetor_inteiro_sem_sinal[16];
float vetor_float[15];
double vetor_double[20];
long long int vetor_inteiro_muito_longo[12];
```

Cada vetor foi declarado com um tamanho diferente, primeiro vem o **identificador do tipo de dado** desejado para o vetor, em seguida o **nome**, que segue as mesmas regras para declaração de nomes das variáveis comuns, por último, entre colchetes, vem um número identificando o **tamanho do vetor**. Um vetor de tamanho **N** identifica que serão colocadas N variáveis lado a lado, a primeira variável tem **índice 0** e a última variável tem **índice N – 1**. Para declarar um vetor e já atribuir um valor para ele podemos usar o comando:

```
int v[10] = {83,91,21,92,66,18,35,59,96,82};
```

Índice	0	1	2	3	4	5	6	7	8	9
Acesso	v[0]	v[1]	v[2]	v[3]	v[4]	v[5]	v[6]	v[7]	v[8]	v[9]
Valor	83	91	21	92	66	18	35	59	96	82

A notação acima, utilizando colchetes para atribuição de valores em um vetor, só pode ser utilizada na declaração do vetor, NÃO PODE SER UTILIZADA EM QUAISQUER OUTROS LUGARES. Para atribuir valores para um vetor em outros lugares do código, deve-se acessar diretamente o índice do vetor que de deseja alterar:

```
vetor_string[0] = 'a';
vetor_inteiro[1] = -40;
vetor_inteiro_sem_sinal[15] = 40;
vetor_float[14] = 1.618;
vetor_double[19] = 3.141592654;
vetor_inteiro_muito_longo[11] = -9223372036854775807;
```

Caso seja necessário realizar a leitura de um valor do teclado, pode-se utilizar o scanf normalmente:

```
scanf("%c", &vetor_string[2]);
scanf("%d", &vetor_inteiro[2]);
scanf("%u", &vetor_inteiro_sem_sinal[2]);
scanf("%f", &vetor_float[2]);
scanf("%lf", &vetor_double[2]);
scanf("%lld", &vetor_inteiro_muito_longo[2]);
```

Similarmente, caso seja necessário mostrar algum valor do vetor na tela usando printf, **ou usá-lo em quaisquer outras funções desejadas**, pode-se acessa-lo normalmente:

```
printf("%c\n", vetor_string[2]);
printf("%d\n", vetor_inteiro[2]);
printf("%u\n", vetor_inteiro_sem_sinal[2]);
printf("%f\n", vetor_float[2]);
printf("%lf\n", vetor_double[2]);
printf("%lld\n", vetor_inteiro_muito_longo[2]);
```

As variáveis de um vetor são sempre chamadas de **índice do vetor**, pode-se utiliza-las da mesma forma que uma variável comum, com uma vantagem, é possível realizar uma operação matemática para definir qual índice deseja-se acessar.

```
int i, fibonacci[10];
fibonacci[0] = 0;
fibonacci[1] = 1;
for (i = 2; i < 10; i++)
    fibonacci[i] = fibonacci[i - 1] + fibonacci[i - 2];
```

Para declarar um vetor, não necessariamente precisamos conhecer o tamanho dele, pode-se simplesmente atribuir os valores desejados ao vetor usando chaves. O compilador faz o trabalho de decidir o tamanho correto.

```
int A[] = {1, 7, 9, 11};
int B[] = {6, 14, 25, 32, 42};
```

Para descobrir o tamanho do vetor basta usar o comando **sizeof**, no entanto, o operador **sizeof** irá retornar o tamanho em bytes do vetor, para saber o número de posições basta dividir pelo número de bytes que o tipo do vetor possui (linha 4). Outra forma para descobrir a quantidade de bytes que o tipo do vetor possui é chamar o **sizeof** para algum índice do vetor (linha 3) por exemplo o índice 0:

```
1   int A[] = {1, 7, 9, 11};
2   int B[] = {6, 14, 25, 32, 42};
3   int M = sizeof(A) / sizeof(A[0]);
4   int N = sizeof(B) / sizeof(int);
```

2.1 Exemplos de códigos que utilizam vetores

2.1.1 Inverter Vetor

Faça um programa que leia um conjunto de valores do teclado, salva num vetor e em seguida mostra os números lidos de traz para frente.

Entrada: Será informado um valor **N** indicando o tamanho do vetor e o número de valores inteiros que devem ser lidos em sequência.

Saída: Imprima os **N** números lidos do último para o primeiro.

Exemplo de Entrada	**Exemplo de Saída**
10 1 2 3 4 5 6 7 8 9 10	10 9 8 7 6 5 4 3 2 1

```c
#include <stdio.h>
int main (void)
{
    int i, N;
    scanf("%d", &N);
    int v[N];
    for (i = 0; i < N; i++)
        scanf("%d", &v[i]);
    for (i = N - 1; i >= 0; i--)
        printf("%d ", v[i]);
    printf("\n");
    return 0;
}
```

Índice	0	1	2	3	4	5	6	7	8	9
Acesso	v[0]	v[1]	v[2]	v[3]	v[4]	v[5]	v[6]	v[7]	v[8]	v[9]
Valor	10	9	8	7	6	5	4	3	2	1

2.1.2 Histograma

Faça um programa que calcula o histograma dos valores fornecidos de entrada. Um Histograma nada mais é do que contar a quantidade de números repetidos, por exemplo a quantidade de 1's lidos.

Entrada: A entrada contém inicialmente um número N, indicando quantos números X serão lidos a seguir, onde X está no intervalo 0 <= X <= 10.

Saída: O programa deve mostrar o histograma dos números 0 até 10.

Exemplo de Entrada	**Exemplo de Saída**
10 0 1 2 3 1 1 1 2 4 10	1 4 2 1 1 0 0 0 0 0 1

```c
#include <stdio.h>
int main (void)
{
    int i, N, x;
    int v[11];
    for (i = 0; i <= 10; i++)
        v[i] = 0;
    scanf("%d", &N);
    for (i = 0; i < N; i++)
    {
        scanf("%d", &x);
        v[x]++;
    }
    for (i = 0; i <= 10; i++)
        printf("%d ", v[i]);
    printf("\n");
}
```

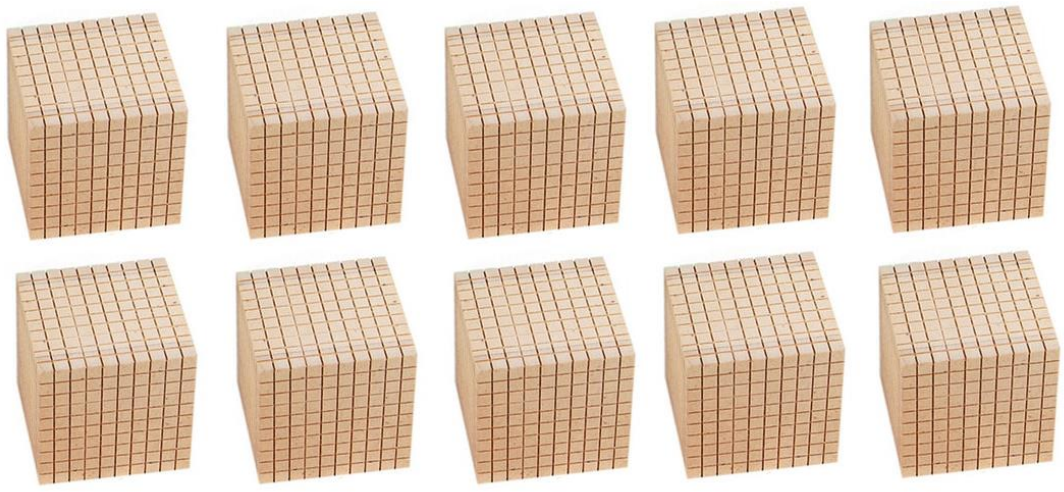
Índice	0	1	2	3	4	5	6	7	8	9	10
Acesso	v[0]	v[1]	v[2]	v[3]	v[4]	v[5]	v[6]	v[7]	v[8]	v[9]	v[10]
Valor	1	4	2	1	1	0	0	0	0	0	1

3 Matrizes

Assim como na matemática, na programação podemos trabalhar com matrizes, que nada mais são do que variáveis agrupadas em duas ou mais dimensões, um bom exemplo para isso é o material dourado.

Por exemplo, uma variável pode ser vista como um bloquinho de 1 unidade. Já um vetor de tamanho 10, pode ser visto como uma barrinha de 10 unidades. Uma matriz de duas dimensões 10 x 10 pode ser visto como uma placa de 100 unidades. Já uma matriz de três dimensões, 10 x 10 x 10 pode ser vista como um cubo de 1000 unidades.

Em teoria não é possível visualizar uma matriz de quatro ou mais dimensões, porém, como muito bem explicado no filme *Interstellar*, se o espaço tridimensional é limitado, podemos sim visualizar a quarta dimensão, por exemplo, uma matriz 10 x 10 x 10 x 10:



Voltando para o caso mais utilizado, matrizes de duas dimensões, na matemática são utilizadas as nomenclaturas “linhas” e “colunas”, onde as linhas são representadas por *i* e as colunas são representadas por *j*. Na linguagem C, a primeira posição de um vetor é a posição 0, logo a primeira posição de uma matriz é a posição i = 0, e j = 0.

Para declarar uma matriz, primeiro deve-se definir o número de linhas, em seguida deve-se definir o número de colunas.

```
char M1[100][10];
int M2[10][10];
unsigned int M3[60][20];
float M4[20][400];
double M5[200][1000];
long long int M6[3][3];
```

Para atribuir um valor a alguma posição especifica da matriz, pode-se utilizar a mesma notação de declaração:

```
M1[0][0] = 'a';
M2[7][1] = -40;
M3[45][19] = 40;
M4[16][399] = 1.618;
M5[150][999] = 3.141592654;
M6[2][1] = -9223372036854775807;
```

Ao utilizar as funções scanf e printf apenas deve-se informar a posição desejada.

```
scanf("%c", &M1[0][0]);
scanf("%d", &M2[7][1]);
scanf("%u", &M3[45][19]);
scanf("%f", &M4[16][399]);
scanf("%lf", &M5[150][999]);
scanf("%lld", &M6[2][1]);

printf("%c\n", M1[0][0]);
printf("%d\n", M2[7][1]);
printf("%u\n", M3[45][19]);
printf("%f\n", M4[16][399]);
printf("%lf\n", M5[150][999]);
printf("%lld\n", M6[2][1]);
```

Caso seja necessário ler uma matriz completa, deve-se apenas utilizar uma estrutura dentro de outra estrutura de repetição, uma para linhas e outra para linhas.

```
int i, j;
int A[10][10];
for(i = 0; i < 10; i++)
{
    for (j = 0; j < 10; j++)
    {
        scanf("%d", &A[i][j]);
    }
}
```

Para imprimir o conteúdo de uma matriz pode-se fazer o mesmo, lembrando que j é o contador das colunas, então o FOR interno irá mostrar todas colunas da linha i, em seguida, podemos imprimir um '\n'.

```
for(i = 0; i < 10; i++)
{
    for (j = 0; j < 10; j++)
    {
        printf(" %d", A[i][j]);
    }
    printf("\n");
}
```

Para exemplificar a indexação de uma matriz, considere o simples exemplo abaixo:

```
int i, j, n = 1;
int A[10][10];
for (i = 0; i < 10; i++)
{
    for (j = 0; j < 10; j++)
    {
        A[i][j] = n;
        n++;
    }
}
```

Note que o padrão em que ‘n’ aumenta, isso porque para cada i, o j vai de 0 até 9.

i \ j	0	1	2	3	4	5	6	7	8	9
0	1	2	3	4	5	6	7	8	9	10
1	11	12	13	14	15	16	17	18	19	20
2	21	22	23	24	25	26	27	28	29	30
3	31	32	33	34	35	36	37	38	39	40
4	41	42	43	44	45	46	47	48	49	50
5	51	52	53	54	55	56	57	58	59	60
6	61	62	63	64	65	66	67	68	69	70
7	71	72	73	74	75	76	77	78	79	80
8	81	82	83	84	85	86	87	88	89	90
9	91	92	93	94	95	96	97	98	99	100

Se por outro lado, o FOR mais externo fosse para o j, então para cada j o valor de i iria variar de 0 até 9. Portanto o padrão de atribuição de ‘n’ seria totalmente diferente.

```
int i, j, n = 1;
int A[10][10];
for (j = 0; j < 10; j++)
{
    for (i = 0; i < 10; i++)
    {
        A[i][j] = n;
        n++;
    }
}
```

i \ j	0	1	2	3	4	5	6	7	8	9
0	1	11	21	31	41	51	61	71	81	91
1	2	12	22	32	42	52	62	72	82	92
2	3	13	23	33	43	53	63	73	83	93
3	4	14	24	34	44	54	64	74	84	94
4	5	15	25	35	45	55	65	75	85	95
5	6	16	26	36	46	56	66	76	86	96
6	7	17	27	37	47	57	67	77	87	97
7	8	18	28	38	48	58	68	78	88	98
8	9	19	29	39	49	59	69	79	89	99
9	10	20	30	40	50	60	70	80	90	100

3.1 Operações sobre os índices

Na matemática é comum utilizarmos a parte acima da diagonal principal, ou abaixo da diagonal principal.... Sabendo que os índices das matrizes em C são definidos da forma:

i \ j	0	1	2	3
0	A00	A01	A02	A03
1	A10	A11	A12	A13
2	A20	A21	A22	A23
3	A30	A31	A32	A33

Diagonal principal	i == j
Elementos acima da principal	j > i
Elementos abaixo da principal	i > j
Diagonal secundaria	i + j == N – 1
Elementos acima da secundaria	i + j < N – 1
Elementos abaixo da secundaria	i + j > N – 1