CAPÍTULO 5

VETORES E MATRIZES

5.1 Vetores

Um vetor armazena uma determinada quantidade de dados de mesmo tipo. Vamos supor o problema de encontrar a média de idade de 4 pessoas. O programa poderia ser:

```
main()
{
  int idade0, idade1, idade2, idade3;

  printf("Digite a idade da pessoa 0: ");
  scanf("%d", &idade0);
  printf("Digite a idade da pessoa 1: ");
  scanf("%d", &idade1);
  printf("Digite a idade da pessoa 2: ");
  scanf("%d", &idade2);
  printf("Digite a idade da pessoa 3: ");
  scanf("%d", &idade3);
  printf("Idade media: %d\n", (idade0+idade1+idade2+idade3) / 4);
}
```

Suponhamos agora que desejássemos encontrar a média das idades de 500 pessoas. A tarefa passa a ser bem mais trabalhosa, sendo em diversos casos impraticável se resolver da maneira apresentada acima.

A solução para este problema é a utilização de vetores. Um vetor é uma série de variáveis de mesmo tipo referenciadas por um único nome, onde cada variável é diferenciada através de um índice, que é representado entre colchetes depois do nome da variável.

A declaração

```
int idade[4];
```

aloca memória para armazenar 4 inteiros e declara a variável idade como um vetor de 4 elementos.

O programa da média das idades poderia ser substituído por:

```
main()
{
  int idade[4], i, soma = 0;

  for(i = 0; i < 4; i++)
  {
    printf("Digite a idade da pessoa %d: ", i);
    scanf("%d", &idade[i]);
  }
  for(i = 0; i < 4; i++)
    soma += idade[i];
  printf("Idade media: %d\n", soma / 4);
}
```

Ao escrever o mesmo programa utilizando vetores, a complexidade do código fica independente do tamanho do vetor.

Para acessar um determinado elemento do vetor, deve-se chamar o elemento pelo nome da variável seguido do índice, entre colchetes. Note que o índice começa em 0, e não em 1. Ou seja, o elemento

```
idade[2]
```

não é o segundo elemento e sim o terceiro, pois a numeração começa de 0. Em nosso exemplo utilizamos uma variável inteira, i, como índice do vetor.

Suponhamos o seguinte programa:

```
main()
{
  int idade[5];

  idade[0] = 15;
  idade[1] = 16;
  idade[2] = 17;
  idade[3] = 18;
  idade[4] = 19;
  idade[5] = 20;
}
```

Nota-se que neste programa, definimos a variável *idade* como um vetor de 5 elementos (0 a 4). Definimos os elementos do vetor mas, na última definição, ultrapassamos o limite do vetor. A linguagem C não realiza verificação de limites em vetores. Quando o vetor foi definido, o compilador reservou o espaço de memória equivalente a 5 variáveis inteiras, ou seja, 10 bytes. Quando tentamos acessar um elemento que ultrapasse o limite de um vetor, estamos acessando uma região de memória que não pertence a esse vetor.

5.1.1 Inicialização

A linguagem C permite que vetores sejam inicializados. No caso, será inicializada uma variável contendo o número de dias de cada mês:

```
int numdias[12] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
```

O número de elementos do vetor pode ser omitido, ou seja

```
int numdias[] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
```

Se nenhum número for fornecido para inicializar o vetor, o compilador contará o número de itens da lista de inicialização e o fixará como dimensão do vetor.

Na falta de inicialização explícita, variáveis **extern** e variáveis **static** são inicializadas com valor zero; variáveis automáticas têm valor indefinido (isto é, lixo).

Se há menos inicializadores que a dimensão especificada, os outros serão zero. Se há mais inicializadores que o especificado, o compilador acusa um erro.

Em C não há como se especificar a repetição de um inicializador, nem de se inicializar um elemento no meio de um vetor sem inicializar todos os elementos anteriores ao mesmo tempo.

5.1.2 Vetores como argumentos de funções

Vetores podem ser passados também como argumentos de funções. Vamos rescrever o programa que calcula a média das idades para um programa contendo uma função que receba um vetor e retorne a sua média.

```
int media(int valor[], int nElementos)
{
  int i, soma = 0;

  for(i = 0; i < nElementos; i++)
    soma += valor[i];
  return soma / nElementos;
}

main()
{
  int idade[5], i;

  for(i = 0; i < 5; i++)
  {
    printf("Digite a idade da pessoa %d: ", i);
    scanf("%d", &idade[i]);
  }
  printf("Idade media: %d\n", media(idade, 5));
}
```

Note que na declaração da função, o argumento que representa o vetor é declarado com colchetes. Além dele, passamos como argumento da função também o tamanho do vetor. Sem ele, a função não tem como saber o tamanho do vetor que foi passado a ela como argumento.

Na função main(), chamamos a função media() passando dois atributos: idade e 5. Para acessarmos um elemento do vetor, escrevemos após o seu nome o índice do elemento desejado entre colchetes (por exemplo, idade[0]). A chamada ao nome de um vetor sem que seja fornecido o índice de um determinado elemento representa o primeiro endereço de memória acessado por esse vetor. Ou seja, ao passarmos um vetor como argumento da função, o compilador não cria uma cópia do vetor para ser utilizada na função. É o próprio vetor quem é passado como argumento e pode ser alterado na função.

5.2 Vetores de caracteres

O vetor de caracteres, também conhecidos como *string*, é uma das formas de dados mais importantes da linguagem C. É usado para manipular texto, como palavras, nomes e frases. Em algumas linguagens, como Pascal e Basic, string é um tipo primitivo. Em C, é tratada como um vetor de elementos do tipo **char**. Cada elemento pode ser acessado individualmente, como em qualquer vetor, através do uso de colchetes.

Sempre que o compilador encontra qualquer coisa entre aspas, ele reconhece que se trata de uma string constante, isto é, os caracteres entre aspas mais o caractere nulo ('\0'). Para declarar uma variável do tipo string, deve-se declará-la como um vetor de caracteres, ou seja:

```
char Nome[50];
```

Esta declaração cria a variável Nome como string, ou vetor de caracteres, permitindo um comprimento de até 50 caracteres. Vale lembrar que o último caractere de uma string deve ser o caractere nulo, ou seja, temos 49 caracteres para trabalhar livremente. A declaração:

```
char Nome[5] = "Pedro";
```

é inválida, por não ter sido reservado espaço suficiente para as 6 variáveis ('P', 'e', 'd', 'r', 'o' e '\0').

Para manipulação de strings, são fornecidas diversas funções na biblioteca C (arquivo string.h). Algumas dessas funções são descritas a seguir. A sintaxe apresentada para cada função não é exatamente a sintaxe fornecida pela biblioteca, mas representa basicamente o que a função executa e já foi explicado até o presente momento.

5.2.1 A função strlen()

A função strlen() retorna o número de caracteres da string, sem contar o caractere nulo.

Sintaxe:

```
int strlen(char[] s);
```

Exemplo:

```
main()
{
    char Nome[6] = "Navio";

    printf("%d\n%d\n", strlen("Barco a vela"), strlen(Nome));
}

SAÍDA
12
5
```

Note que os espaços fazem parte da string, e são simplesmente caracteres, assim como letras e algarismos.

5.2.2 A função strcmp()

A função strcmp() compara duas strings. Retorna

- um valor menor que zero se a primeira string for menor que a segunda;
- zero se as strings forem iguais;
- um valor maior que zero se a primeira string for maior que a segunda.

Entende-se pôr comparação entre strings, sua posição em ordem alfabética. A ordem alfabética é baseada na tabela ASCII (Anexo A). Portanto, cuidado ao comparar maiúsculas com minúsculas, pois na tabela ASCII as letras maiúsculas possuem um valor menor que as letras minúsculas, ou seja, o caractere 'Z' vem antes do caractere 'a'.

Sintaxe:

```
int strcmp(char[] s1, char[] s2);
```

Exemplo:

```
main()
{
    printf("%d\n", strcmp("Ariranha", "Zebra"));
    printf("%d\n", strcmp("Zebra", "Ariranha"));
    printf("%d\n", strcmp("Zebra", "Zebra"));
}
```

```
SAÍDA
(Algum valor menor que 0)
(Algum valor maior que 0)
0
```

Espaços e outros caracteres especiais também são considerados na comparação, algumas vezes não fornecendo o que se espera de uma ordenação alfabética.

5.2.3 A função strcpy()

A função strcpy() copia o conteúdo de uma string para outra. A string de destino já tem que ter espaço reservado na memória antes de ser chamada a função.

Sintaxe:

```
void strcpy(char[] destino, char[] origem);
```

Exemplo:

```
main()
{
    char Nome[10];

    strcpy(Nome, "Teste")
    printf("%s\n", Nome);
}

SAÍDA
Teste
```

5.2.4 A função strcat()

A função strcat() concatena duas strings, ou seja, anexa o conteúdo de uma na outra. Similarmente à função strcpy(), a string de destino tem que ter espaço reservado na memória.

Sintaxe:

```
void strcat(char[] destino, char[] origem);
```

Exemplo:

```
main()
{
    char Nome[12];

    strcpy(Nome, "Teste")
    strcpy(Nome, "geral")
    printf("%s\n", Nome);
}
SAÍDA
Testegeral
```

5.3 Matrizes

A linguagem C permite vetores de qualquer tipo, inclusive vetores de vetores. Por exemplo, uma matriz é um vetor em que seus elementos são vetores. Com dois pares de colchetes, obtemos uma

matriz de duas dimensões e, para cada par de colchetes adicionado, obtemos uma matriz com uma dimensão a mais.

Por exemplo, a declaração

```
int A[5][6];
```

indica que A é uma matriz 5x6 e seus elementos são inteiros.

5.3.1 Inicialização

As matrizes são inicializadas como os vetores, ou seja, os elementos são colocados entre chaves e separados por vírgulas. Como seus elementos são vetores, estes, por sua vez, também são inicializados com seus elementos entre chaves e separados por vírgulas. Por exemplo:

```
int A[5][6] = { { 1,  2,  3,  4,  5,  6},
                { 7,  8,  9, 10, 11, 12},
                {13, 14, 15, 16, 17, 18},
                {19, 20, 21, 22, 23, 24},
                {25, 26, 27, 28, 29, 30} };
```

Caso as matrizes sejam de caracteres, isto é equivalente a termos um vetor de strings. Sua inicialização pode se dar da forma

```
char Nome[5][10] = {"Joao",
                    "Jose",
                    "Maria",
                    "Geraldo",
                    "Lucia"};
```

A matriz será inicializada como

J	o	a	o	\0	\0	\0	\0	\0	\0
J	o	s	e	\0	\0	\0	\0	\0	\0
M	a	r	i	a	\0	\0	\0	\0	\0
G	e	r	a	l	d	o	\0	\0	\0
L	u	c	i	a	\0	\0	\0	\0	\0

5.3.2 Matrizes como argumento de funções

A passagem de uma matriz para uma função é similar à passagem de um vetor. O método de passagem do endereço da matriz para a função é idêntico, não importando quantas dimensões ela possua, já que sempre é passado o endereço da matriz.

Entretanto, na declaração da função, a matriz é um pouco diferente. A função deve receber o tamanho das dimensões a partir da segunda dimensão. Por exemplo:

```
void Determinante(double A[][5]);
```

Note que é fornecido a segunda dimensão da matriz. Isto é necessário para que, ao chamarmos o elemento A[i][j], a função saiba a partir de que elemento ela deve mudar de linha. Com o número de elementos de cada linha, a função pode obter qualquer elemento multiplicando o número da linha pelo tamanho de cada linha e somando ao número da coluna.

Por exemplo, o elemento A[2][3], é o elemento de índice 13, já que 2 * 5 + 3 = 13.