Vetores

Nos capítulos anteriores estudamos as opções disponíveis na linguagem C para representar:

- Números inteiros em diversos intervalos.
- Números fracionários com várias alternativas de precisão e magnitude.
- Letras, dígitos, símbolos e outros caracteres.

Todas estas opções estão definidas previamente na própria linguagem C e, portanto, são chamadas de **tipos de dados da linguagem C**.

Agora, conheceremos os mecanismos para definir novos tipos de dados, conforme a necessidade do seu programa. Assim, eles são chamados de **tipos de dados do programador**.

Vetores

Um vetor é uma seqüência de vários valores do mesmo tipo, armazenados *seqüencialmente* na memória, e fazendo uso de um *mesmo nome de variável* para acessar esses valores. Um vetor também pode ser entendido logicamente como uma lista de elementos de um mesmo tipo.

Cada elemento desta seqüência pode ser acessado individualmente através de um índice dado por um número inteiro. Os elementos são indexados de 0 até *n-1*, onde *n* é a quantidade de elementos do vetor. O valor de *n* também é chamado de dimensão ou tamanho do vetor. O vetor tem tamanho fixo durante a execução do programa, definido na declaração. Durante a execução não é possível aumentar ou diminuir o tamanho do vetor. Note que a numeração começa em zero, e não em um. **Essa é uma fonte comum de erros.**

A Figura 1 ilustra um vetor com 10 elementos, denominados v_0, v_1, … v_9, todos eles de tipo `int`.

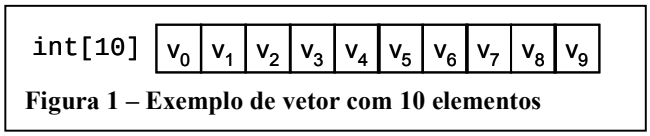

Figura 1 – Exemplo de vetor com 10 elementos

É importante saber que os elementos do vetor são armazenados sequencialmente na memória do computador. Assim, na figura, se cada valor de tipo `int` ocupar 4 bytes de memória, teremos 40 bytes consecutivos reservados na memória do computador para armazenar todos os valores do vetor. No entanto, por ora, não faremos uso explícito dessa informação, uma vez que o compilador se encarregará de endereçar cada elemento do vetor automaticamente, conforme as necessiades do programador, como veremos.

Declaração de vetores

A declaração de vetores obedece à mesma sintaxe da declaração de variáveis. A diferença está no valor entre colchetes, que determina quantos elementos ele armazenará, ou seja, em outras palavras, determina o seu tamanho ou dimensão.

```
tipo variável[tamanho];
```

Por exemplo, para declarar um vetor com 10 números inteiros:

```
int vetor[10];
```

O tamanho precisa ser necessariamente um número inteiro e constante. Ele não pode ser resultado de uma expressão:

```
int tamanho = 10;
int vetor[tamanho*2];  // ERRADO!
```

Acesso ao conteúdo de vetores

Já aprendemos que uma invocação de nome de variável é capaz de acessar um dado na memória. O tipo do valor retornado é igual ao tipo utilizado na declaração da variável. O nome da variável também é chamado de uma expressão de referência de memória, ou **simplesmente** de referência de memória.

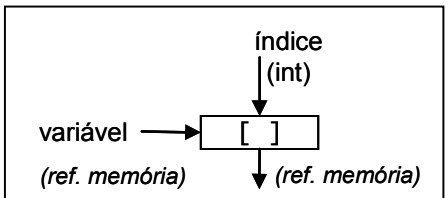

Figura 2 – Operador índice

Com os vetores, temos uma nova expressão de referência de memória: o **operador de índice** **`[ ]`**. Ele utiliza uma referência de memória (normalmente uma variável do tipo vetor) e um número inteiro (o índice). Ele retorna uma referência para o elemento correspondente ao índice. O tipo do valor retornado é o mesmo tipo da declaração do vetor.

Por exemplo, para atribuir o valor 3 na primeira posição do vetor, escrevemos:

```
vetor[0] = 3;
```

Note que o índice zero indica a primeira posição no vetor. A expressão `vetor[0]` referencia a posição de memória correspondente ao elemento de índice zero no vetor. Para somar os primeiros três elementos e armazenar o valor calculado no quarto elemento, escrevemos:

```
vetor[3] = vetor[0] + vetor[1] + vetor[2];
```

Em expressões, uma referência indexada a um vetor pode ser usada da mesma forma e nas mesmas posições em que usaríamos variáveis convencionais de mesmo tipo. Tudo se passa como se tivéssemos várias variáveis declaradas simultaneamente, todas de mesmo tipo, e com "nomes" `vetor[0]`, `vetor[1]`, e assim por diante.

É muito comum utilizar a estrutura de repetição **`for`** para percorrer todos os elementos de um vetor. Por exemplo, para imprimir todos os elementos de um vetor de 100 elementos:

```
int indice;
int vetor[100];
...
for (indice = 0; indice < 100; indice++) {
     printf("%d", vetor[indice]);
}
```

Lembre-se que para um vetor de tamanho 100, o primeiro elemento tem índice 0 e o último elemento tem índice 99. Por este motivo, o **`for`** repete para valores de `indice` variando de 0 até 99. A condição utilizada para terminar o **`for`** é `indice < 100`. Isto sugere que o **`for`** está realizando 100 repetições, que é justamente o tamanho do vetor. A condição `indice`

<= 99 também funcionará corretamente, mas não deixa tão claro que o vetor possui 100 elementos.

Exemplo

Um programa que lê dez números e os imprime em ordem inversa. Para isso, é necessário armazenar os 10 números para poder imprimi-los de trás para frente. Seria possível utilizar 10 variáveis distintas, mas a solução com vetor é bem mais elegante. Mais ainda, se fossem 10.000 números, e não 10, ficaria impraticável usar variáveis distintas.

Código fonte:

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

int main(int argc, char *argv[]) {

     int valores[10];
     int indice;

     printf("Escreva 10 números inteiros: ");
     for (indice = 0; indice < 10; indice++) {
          scanf("%d", &valores[indice] );
     }

     printf("Valores em ordem reversa:\n");
     for (indice = 9; indice >= 0; indice--) {
          printf("%d ", valores[indice]);
     }

     return 0;
}
```

Consulte: Vetores\Reverso01\Reverso01.vcproj

Descrição passo a passo:

```
int valores[10];
int indice;
```

A primeira variável, valores, é um vetor de números inteiros, com 10 elementos. Eles são numerados seqüencialmente de 0 até 9. Este vetor armazenará os valores digitados pelo usuário. A segunda declaração cria uma variável contadora para o **for** de leitura e o **for** de escrita de valores.

```
for (indice = 0; indice < 10; indice++) {
     scanf("%d", &valores[indice] );
}
```

A estrutura de repetição **for** executa 10 vezes, variando o valor de `indice` desde 0 até 9. A cada repetição, o comando `scanf` lê um número inteiro e o armazena no `indice`-ésimo elemento do vetor `valores`. Note que tal como no `scanf` para variáveis comuns, a referência `valores[indice]` se comporta como o nome de uma variável comum de tipo **`int`** e é necessário precedê-la pelo símbolo **`&`**

```
for (indice = 9; indice >= 0; indice--) {
     printf("%d ", valores[indice]);
}
```

O vetor contém os 10 números lidos no **for** anterior. Agora, vamos usar novamente a estrutura de repetição **for** para imprimir o vetor de trás para frente. Por este motivo, fazemos o índice variar de 9 para 0.

Conteúdo inicial de vetores

Na declaração, pode-se definir o valor inicial de cada elemento de um vetor. A sintaxe é semelhante à declaração de uma variável comum com valor inicial, mas os elementos são listados entre as chaves { e } e separados por vírgula.

```
tipo variável[n] = { elem0, elem1, elem2, elem3, ... elemn-1 };
```

Como o número de elementos do vetor pode ser inferido a partir da lista entre chaves, podemos omitir o tamanho do vetor:

```
tipo variável[] = { elem0, elem1, elem2, elem3, ... elemn-1 };
```

Regras para acesso ao vetor

O programador deve observar algumas restrições na manipulação de variáveis que representam vetores.

Índices inválidos

Os elementos são numerados sempre de 0 até *tamanho-1*. Caso o programa tente acessar erroneamente um elemento de índice negativo ou de índice além do tamanho do vetor, as conseqüências poderão ser imprevisíveis. No melhor dos casos, o sistema operacional detectará essa anomalia e o programa será finalizado sinalizando um erro de execução.

Atribuir o valor de todos os elementos de uma só vez

Exceto na declaração do vetor, não é possível atribuir valores a todos os elementos em uma só linha. Cada elemento precisa ser acessado individualmente. Tampouco é possível usar um único scanf para ler todo o conteúdo do vetor. O código abaixo está, portanto, errado:

```
int vetor[10];
// inicializar todos os elementos com valor 0
vetor = 0;   // ERRADO!
```

O correto é utilizar uma estrutura de repetição **for** para atribuir o valor a cada elemento.

```
int vetor[10];
int indice;
// inicializar todos os elementos com o valor 0
for (indice = 0; indice < 10; indice++) {
   vetor[indice] = 0;
}
```

Copiar vetores

Tampouco é possível copiar o conteúdo de um vetor para um outro, mesmo que os dois sejam de mesmo tamanho e os elementos sejam de mesmo tipo.

```
int vetorA[10], vetorB[10];
```

```
// copiar o conteúdo do vetor B para o vetor A
vetorA = vetorB;   // ERRADO!
```

O correto é utilizar uma estrutura de repetição **for** para copiar um elemento de cada vez.

```
int vetorA[10], vetorB[10];
int indice;
// copiar o conteúdo do vetor B para o vetor A
for (indice = 0; indice < 10; indice++) {
   vetorA[indice] = vetorB[indice];
}
```

Vetor de tamanho variável

Na linguagem C, todos os vetores têm um tamanho fixo, e que não pode variar durante a execução do programa. Mas como proceder quando o tamanho do vetor não pode ser previsto até o momento da execução?

Considere um programa que lê *n* valores e os armazena no vetor. O valor *n* é informado pelo usuário durante a execução do programa. A linguagem C não oferece recursos para se declarar um vetor cujo tamanho se ajuste automaticamente ao número *n* de elementos.

A solução mais simples é declarar o vetor com o tamanho máximo necessário para tratar o pior caso. O número de elementos realmente utilizado ficará armazenado em uma outra variável. Uma abordagem mais flexível será apresentada mais tarde, utilizando alocação dinâmica de memória, mas esta é uma técnica mais complexa.

Continuando o exemplo, vamos supor que foi utilizado algum critério para descobrir que o vetor nunca precisará armazenar mais do que 100 elementos.

```
int valores[100];
int numero_elementos;
```

Para imprimir todos os elementos do vetor, procedemos como antes, utilizando uma estrutura de repetição **for**. Mas, ao invés de utilizar o tamanho máximo como condição de parada, comparamos o índice com o tamanho dado pela variável `numero_elementos`.

```
for (indice = 0; indice < numero_elementos; indice++) {
     printf("%d", vetor[indice];
}
```

Apesar do número de celas do vetor que estamos usando ser "variável" (o número é dado pelo conteúdo da variável `numero_elementos`) a capacidade máxima do vetor foi declarada como 100 elementos. O programa precisa garantir que a quantidade de elementos utilizada nunca extrapole o número máximo de elementos declarados para o vetor. Caso contrário, o resultado da execução do programa será imprevisível! **A não verificação do valor usado como índice em um vetor é uma das causas mais freqüentes de falha de software e violação de segurança.**

Exemplo

Programa que lê *n* valores, armazena-os em um vetor e em seguida os imprime em ordem inversa.

Código fonte:

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

int main(int argc, char *argv[]) {

     int valores[100];
```

```
        int numero_valores;
        int indice;

        printf("Quantos valores? (no máximo 100): ");
        scanf("%d", &numero_valores);
        if ( (numero_valores > 100) || (numero_valores < 0) ) {
             printf("Número de valores inválido\n");
             return 1;
        }

        printf("Escreva %d números inteiros: ", numero_valores);
        for (indice = 0; indice < numero_valores; indice++) {
             scanf("%d", &valores[indice] );
        }

        printf("Valores em ordem reversa:\n");
        for (indice = numero_valores-1; indice >= 0; indice--) {
             printf("%d ", valores[indice]);
        }

        return 0;
    }
```

Consulte: Vetor\Reverso02\Reverso02.vcproj

Descrição passo a passo:

```
int valores[100];
```

A variável `valores` é declarada como um vetor de 100 elementos de tipo inteiro. Ele armazenará os valores digitados pelo usuário.

```
int numero_valores;
```

A variável `numero_valores` armazena quantos elementos do vetor `valores` estão realmente sendo utilizados para armazenar os valores digitados pelo usuário.

```
int indice;
```

A declaração cria uma variável contadora para o **`for`** de leitura e o **`for`** de escrita de valores.

```
printf("Quantos valores? (no máximo 100): ");
scanf("%d", &numero_valores);
```

Pede ao usuário para informar de quantos elementos consiste a lista de números. Esse valor será armazenado na variável `numero_valores` e passará a controlar o número de repetições do **`for`**.

```
if ( (numero_valores > 100) || (numero_valores < 0) ) {
      printf("Número de valores inválido!\n");
      return 1;
}
```

Verifica se o número de elementos que o usuário digitou está dentro do limite suportado pelo programa. Esta verificação é essencial para garantir que nenhuma operação de indexação do vetor seja realizada com índice maior que a capacidade do vetor, ou com índice negativo, o que poderia comprometer a integridade do programa.

```
for (indice = 0; indice < numero_valores; indice++) {
      scanf("%d", &valores[indice] );
}
```

A estrutura de repetição **for** executa numero_valores vezes, variando o conteúdo de indice de 0 até numero_valores-1. A cada repetição, o comando scanf lê um número inteiro e o armazena no indice-ésimo elemento do vetor valores. Note que tal como no scanf para variáveis comuns, o nome do vetor é precedido pelo símbolo &.

```
for (indice = numero_valores-1; indice >= 0; indice--) {
        printf("%d ", valores[indice]);
}
```

O vetor contém numero_valores números lidos no **for** anterior. Agora, vamos usar novamente a estrutura de repetição **for** para imprimir o vetor de trás para frente. Por este motivo, fazemos o índice variar de numero_valores-1 para 0.

Matrizes e vetores multidimensionais

A noçao de matriz é a generalização imediata da noção de vetor. Uma matriz é uma tabela de vários valores do mesmo tipo, armazenados seqüencialmente e fazendo uso de um mesmo nome de variável para acessar esses valores.

Cada elemento da tabela pode ser acessado individualmente através de dois índices com valores inteiros. Estes índices poderiam ser interpretados como a linha e a coluna da matriz.

A linguagem C define uma matriz como um vetor, cujos elementos *são novamente vetores* de mesmo tamanho e tipo. Na verdade, o número de linhas corresponde ao número de elementos do vetor externo, e o número de colunas é o tamanho dos vetores internos que constituem cada elemento dos vetores externos. A Figura 3 ilustra este conceito:

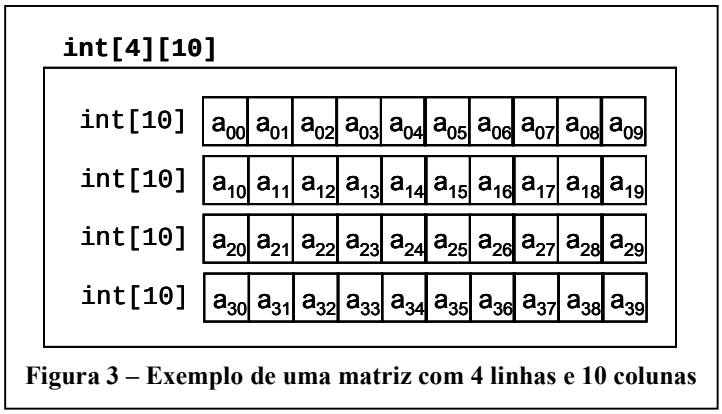

Figura 3 – Exemplo de uma matriz com 4 linhas e 10 colunas

Todos os conceitos e todas as regras válidas para os vetores também são válidas para as matrizes. A matriz tem tamanho fixo, definido na declaração. As linhas são numeradas de 0 até *linhas*-1 e as colunas de 0 até *colunas*-1.

Declaração de matrizes

A declaração de matrizes obedece a mesma sintaxe que a declaração de vetores, exceto pela adição de uma nova dimensão escrita entre colchetes [ ].

```
tipo variável[linhas][colunas];
```

Por exemplo, para declarar uma matriz de números inteiros com 4 linhas e 10 colunas:

```
int matriz[4][10];
```

A atribuição ou leitura de valores em uma matriz utiliza dois índices. O primeiro elemento é armazenado em matriz[0][0] o segundo em matriz[0][1] e assim por diante, até o último elemento, que é armazenado em matriz[*linhas*-1][*colunas*-1].

Para atribuir o valor 3 na linha 2, coluna 5, escrevemos:

```
matriz[1][4] = 3;
```

Note como a linha 2 é numerada como 1, pois começamos a contar a partir do zero. Idem, a coluna 5 é numerada como 4.

Para percorrer os elementos de uma matriz, são necessárias duas estruturas de repetição **for**, uma dentro da outra. O **for** externo percorre as linhas da matriz, o **for** interno percorre as colunas de uma determinada linha que está fixada pelo **for** externo. Por exemplo, para imprimir todos os elementos de uma matriz 4x10, linha por linha:

```
int linha, coluna;
int matriz[4][10];
...
for (linha = 0; linha < 4; linha++) {
     for (coluna = 0; coluna < 10; coluna++) {
          printf("%d ", matriz[linha][coluna]);
     }
     printf("\n");
}
```

para cada valor de `linha` fixado pelo **for** externo entre 0 e 3, o **for** interno executa, varrendo as colunas correspondentes àquela linha, de 0 a 9. Assim, quando o **for** interno excecuta, o valor de `linha` está fixado pelo **for** externo, fazendo com que o `printf` imprima os valores da linha `linha` da matriz. Antes de retornar ao **for** externo imprimimos um caracter para mandar o cursor para a próxima linha.

Declaração de vetores multidimensionais

Para cada nova dimensão que desejarmos adicionar ao vetor, devemos adicionar um novo tamanho entre colchetes `[ ]`. Para um vetor com *n* dimensões:

```
tipo variável[tamanho1][tamanho2]...[tamanhoN];
```

Textos ou Cadeias (strings)

Já utilizamos seqüências de caracteres (que também chamaremos de textos) diversas vezes, em conjunto com os comandos `printf` e `scanf`:

```
printf("Digite dois números: ");
scanf("%d %d", &a, &b);
```

Neste caso, `"Digite dois números: "` e `"%d %d"` são dois exemplos de textos em C.

Armazenamento de seqüências de texto

Em capítulo anterior aprendemos como utilizar o tipo *char* para representar uma *única* letra, um *único* dígito ou um *único* símbolo. De fato, o tipo *char* armazena o índice da tabela ASCII correspondente ao caractere. Mas ele por si só não é capaz de armazenar um texto formado por vários caracteres.

Para obter esse efeito na linguagem C, armazenamos o texto como um vetor de caracteres, onde cada caractere do texto é um elemento do vetor. O fim do texto deve ser demarcado

por um caractere adicional e especial: o *caractere nulo*, que é simplesmente um caractere cujo valor numérico é zero. Ele é representado como `'\0'`. Por este motivo, o comprimento do vetor de caracteres é sempre *no mínimo* o número de caracteres no texto mais 1. Como veremos, em alguns casos, o caractere de valor zero é adicionado automaticamente ao texto quando este é manipulado diretamente na linguagem C.

Por ser um vetor de caracteres, cujo primeiro elemento tem íncice 0, o acesso a um determinado índice *i* retornará o caractere que está na posição *i*+1 do texto.

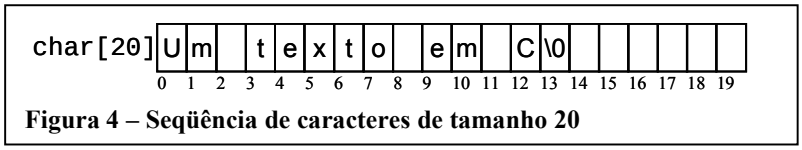

Figura 4 – Seqüência de caracteres de tamanho 20

A Figura 4 ilustra como um programa escrito em C armazenaria a seqüência "Um texto em C", que tem 13 símbolos, em um vetor de caracteres de tamanho 20. Como o vetor é indexado a partir do número 0, os caracteres que formam texto ficarão guardados do índice 0 até o índice 12. Logo em seguida, no índice 13, é armazenado o caractere `'\0'`. Ele foi adicionado automaticamente e indica que o texto terminou na posição anterior. Esta marcação é importante, pois o comprimento do texto é menor que o tamanho máximo disponível no vetor. No espaço restante, do índice 14 até 19, o vetor contém caracteres indeterminados. Para saber a quinta letra do texto, consulta-se `texto[4]`, que retornará o caractere `'e'`.

Diferentemente de linguagens de programação de mais alto nível, a linguagem C oferece um conjunto mais limitado de recursos para manipular textos diretamente. No entanto, o arquivo de inclusão `string.h` contém várias rotinas pré-compiladas para se lidar com cadeias de caracteres em C.

Declaração de variáveis de tipo texto

Uma variável para armazenar textos é declarada como um vetor de caracteres:

```
char texto[tamanho];
```

Note que `tamanho` é o comprimento do vetor, ou seja, ele pode armazenar um texto com no máximo `tamanho-1` caracteres (pois a última posição conterá sempre o caractere `'\0'`). Quando não sabemos de antemão o comprimento do texto que desejamos armazenar, será necessário utilizar um limitante superior de pior caso para o tamanho. Esta abordagem foi discutida para vetores na seção anterior.

Para declarar uma variável que pode armazenar até 100 caracteres, devemos declarar um vetor de tamanho 101 (uma posição a mais para o caractere `'\0'`):

```
char texto[101];
```

O texto armazenado pela variável pode ser atribuído logo na declaração:

```
char texto[tamanho] = "texto";
```

Novamente, o `tamanho` deve ser pelo menos uma unidade maior que o comprimento de `"texto"`, para poder abrigar o caractere `'\0'` no final da seqüência, que é adicionado automaticamente pelo compilador C. Note como o texto é uma seqüência de caracteres colocados entre *aspas duplas*.

Para declarar uma variável com o texto da Figura 4, declaramos:

```
char texto[20] = "Um texto em C";
```

Quando uma variável deve armazenar um texto que não será alterado no decorrer do programa, podemos omitir o tamanho na declaração. O compilador determinará automaticamente o tamanho do vetor de caracteres.

```
char texto[] = "texto";
```

Escrita

Para imprimir o conteúdo de um vetor de caracteres utilizamos o comando `printf`, com o formatador `%s`:

```
char texto[] = "Um texto em C";
printf("A variavel texto contem: %s", texto);
```

Que imprimirá:

```
A variavel texto contem: Um texto em C
```

Leitura

O comando `scanf`, com o modificador `%s`, é capaz de ler uma seqüência de caracteres, terminada com fim de linha ou espaço em branco.

```
char texto[20];
printf("Escreva uma palavra: ");
scanf("%s", texto);
```

Observações:

- Note que neste caso particular de `scanf` *não há* o símbolo `&` em frente do nome da variável! Isto porque a variável, na verdade, representa um vetor, e não uma variável simples.
- A variável, do tipo vetor de caracteres, precisa de espaço suficiente para receber o texto digitado pelo usuário. Se isso não for verdade, o resultado será imprevisível.
- O comando `scanf` adiciona automaticamente o caractere especial `'\0'` após o texto lido.
- A leitura terminará após a primeira seqüência de caracteres não nulos (aqueles diferentes de "nova linha" e de tabulação) que for encontrada.

Para ler uma linha completa podemos utilizar `gets:`

```
char texto[20];
gets(texto);
```

O comando `gets` lê todos os caracteres até que o usuário pressione ENTER. O caractere `'\0'` é adicionado no fim do texto. Note que `gets` está no aquivo `string.h`, o qual deve ser incluído. Neste caso, portanto, se o usuário digitar mais de 19 caracteres teremos problemas.

Atribuição

Por designarem vetores de caracteres, não se pode usar o operador de atribuição com variáveis de tipo texto:

```
char texto[20] = "Um texto em C";
texto = "Outro texto em C";   //ERRADO!
```

Para esta operação, a linguagem C oferece a função `strcpy`, definida na biblioteca `string.h`:

```
#include <string.h>
...
char texto[20] = "Um texto em C";
strcpy(texto, "Outro texto em C");
```

O vetor de caracteres destino é o primeiro argumento da função `strcpy`, o texto a ser copiado está no segundo argumento. A função `strcpy` automaticamente adicionará o caractere `'\0'` ao primeiro vetor, para indicar a posição do fim do texto. O vetor destino precisa ter tamanho suficiente para receber o novo texto mais o caractere `'\0'`. Há várias outras funções que manipulam cadeias em C. Consulte a documentação do seu compilador.

Se desejarmos modificar apenas um caractere, podemos utilizar seu índice, aproveitando o fato de C representar textos como vetores de caracteres:

```
char texto[20] = "Um texto em C";
texto[12] = 'B';
printf("%s", texto);
```

Imprimirá:

```
Um texto em B
```