**7 VETORES (MATRIZES UNIDIMENSIONAIS)**

Vetores nada mais são que matrizes unidimensionais. Vetores são uma estrutura de dados muito utilizada. É importante notar que vetores, matrizes bidimensionais e matrizes de qualquer dimensão são caracterizadas por terem todos os elementos pertencentes ao mesmo tipo de dado. Para se declarar um vetor podemos utilizar a seguinte forma geral:

*tipo\_da\_variável nome\_da\_variável [tamanho];*

Veja a representação gráfica de uma matriz de uma matriz de 5 posições

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Valores | 10 | 18 | 25 | 05 | 23 |
| Índices | 0 | 1 | 2 | 3 | 4 |

Quando o C vê uma declaração como esta ele reserva um espaço na memória suficientemente grande para armazenar o número de células especificadas em tamanho. Por exemplo, se declararmos:

                float exemplo [20];

o C irá reservar 4x20=80 bytes. Estes bytes são reservados de maneira contígua.

**Na linguagem C a numeração começa sempre em zero.** Isto significa que, no exemplo acima, os dados serão indexados de 0 a 19. Para acessá-los vamos escrever:

              exemplo[0]

               exemplo[1]

                .

                .

                .

                exemplo[19]

Mas ninguém o impede de escrever:

                exemplo[30]

              exemplo[103]

Por quê? Porque **o C não verifica se o índice que você usou está dentro dos limites válidos**. Este é um cuidado que ***você*** deve tomar. Se o programador não tiver atenção com os limites de validade para os índices ele corre o risco de ter variáveis sobrescritas ou de ver o computador travar. Bugs terríveis podem surgir! Vamos ver agora um exemplo de utilização de vetores:

#include <stdio.h>

#include <stdlib.h>

int main (){

int num[10], cont;

printf("Informe 10 numeros inteiros:\n");

for(cont=0; cont<10; cont++)

scanf ("%d",&num[cont]);

printf("Numeros lidos:\n");

for (cont=0;cont<10; cont++)

printf (" %d",num[cont]);

printf("\n");

system("PAUSE");

return(0);

}

Agora um exemplo mais elaborado:

#include <stdio.h>

#include <stdlib.h>

int main (){

int num[100]; /\* Declara um vetor de inteiros de 100 posições \*/

int cont=0;

int totalnums;

do

{

printf ("\nEntre com um numero (-999 p/ terminar): ");

scanf ("%d",&num[cont]);

cont++;

} while (num[cont-1]!=-999);

totalnums=cont-1;

printf ("\n\n\n\t Os números que você digitou foram:\n\n");

for (cont=0;cont<totalnums;cont++)

printf (" %d",num[cont]);

system("PAUSE");

return(0);

}

No exemplo acima, o inteiro **cont** é inicializado em 0. O programa pede pela entrada de números até que o usuário entre com o Flag -999. Os números são armazenados no vetor **num**. A cada número armazenado, o contador do vetor é incrementado para na próxima iteração escrever na próxima posição do vetor. Quando o usuário digita o flag, o programa abandona o primeiro loop e armazena o total de números gravados. Por fim, todos os números são impressos. É bom lembrar aqui que nenhuma restrição é feita quanto a quantidade de números digitados. Se o usuário digitar mais de 100 números, o programa tentará ler normalmente, mas o programa os escreverá em uma parte não alocada de memória, pois o espaço alocado foi para somente 100 inteiros. **Isto pode resultar nos mais variados erros no instante da  execução do programa.**

Aqui temos um outro exemplo de um programa que NÃO DEVE ser implementado dessa forma. Note que o vetor tem 5 posições e as 3 primeiras são preenchidas (primeiro erro). Depois disso são mostradas as 10 primeiras posições do vetor (segundo erro). Provavelmente serão mostradas as primeiras 3 posições (corretamente), seguda de valores incorretos e poderemos ver ainda os valores das variáveis a e b como se fizessem parte do vetor.

#include <stdio.h>

#include <stdlib.h>

int main() {

int cont;

int vetor[5];

int a = 100;

int b = 200;

for(cont=0; cont<3;cont++){

vetor[cont] = cont;

}

for(cont=0; cont<10; cont++){

printf("%d - ", vetor[cont]);

}

}

EXERCÍCIOS

7.1 Construir um programa que lê 10 valores para um vetor de números inteiros. Em seguida localiza, mostra o maior valor armazenado e sua posição. Usar a pesquisa sequencial (pesquisar a partir da primeira posição) para procurar o maior valor.

7.2 Construir um programa que lê 20 valores para um vetor de inteiros e guarda no vetor1. Em seguida guarda os números pares no vetor 2 e os impares no vetor 3. Ao final mostra os 3 vetores.

7.3 Ordenar o vetor par e mostrá-lo depois de ordenado.

7.4 Construir um programa que leia e guarde em um vetor 10 caracteres (usar a função getchar() e fflush(stdin) e em seguida solicita ao usuário um outro caractere. Pesquisar se esse caractere existe no vetor e em que posições está localizado. Caso não esteja em nenhuma das 10 posições, mostrar uma mensagem informando.

**8 STRINGS**

No C uma string é um vetor de caracteres terminado com um caractere nulo. O caractere nulo é um caractere com valor inteiro igual a zero (código ASCII igual a 0). O terminador nulo também pode ser escrito usando a convenção de barra invertida do C como sendo '\0'. Para declarar uma string, podemos usar o seguinte formato geral:

*char nome\_da\_string[tamanho];*

Isto declara um vetor de caracteres (uma string) com número de posições igual a tamanho. **Note que, como temos que reservar um caractere para ser o terminador nulo, temos que declarar o comprimento da string como sendo, no mínimo, um caractere maior que a maior string que pretendemos armazenar**. Vamos supor que declaremos uma string de 7 posições e coloquemos a palavra João nela. Teremos:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| J | o | a | o | \0 | ... | ... |

No caso acima, as duas células não usadas têm valores indeterminados. Isto acontece porque o C *não* inicializa variáveis, cabendo ao programador esta tarefa. Portanto as únicas células que são inicializadas são as que contêm os caracteres 'J', 'o', 'a', 'o' e '\0' .

Se quisermos ler uma string fornecida pelo usuário podemos usar a função **gets()**. Um exemplo do uso desta função é apresentado abaixo. A função **gets()** coloca o terminador nulo na string, quando você pressiona a tecla "<Enter>".

#include <stdio.h>

#include <stdlib.h>

int main (){

char string[100];

printf ("Digite uma string: ");

gets (string);

printf ("Voce digitou: %s\n", string);

system("PAUSE");

return(0);

}

Neste programa, o tamanho máximo da string que você pode entrar é uma string de 99 caracteres. Se você entrar com uma string de comprimento maior, o programa irá aceitar, mas os resultados podem ser desastrosos.

Como as strings são [vetores de caracteres](http://www.ead.cpdee.ufmg.br/cursos/C/aulas/c520.html), para se acessar um determinado caractere de uma string, basta "indexarmos", ou seja, usarmos um índice para acessarmos o caractere desejado dentro da string. Suponha uma string chamada *str*. Podemos acessar a **segunda** letra de *str* da seguinte forma:

   str[1] = 'a';

Por que se está acessando a segunda letra e não a primeira? Na linguagem C, o índice ***começa em zero.*** Assim, a primeira letra da string sempre estará na posição 0. A segunda letra sempre estará na posição 1 e assim sucessivamente. Segue um exemplo que imprimirá a segunda letra da string "Joao", apresentada acima. Em seguida, ele mudará esta letra e apresentará a string no final.

#include <stdio.h>

#include <stdlib.h>

int main() {

char str[10] = "Joao";

printf("\n\nString: %s", str);

printf("\nSegunda letra: %c", str[1]);

str[1] = 'U';

printf("\nAgora a segunda letra eh: %c", str[1]);

printf("\nString resultante: %s\n", str);

system("PAUSE");

return(0);

}

Nesta string, o terminador nulo está na posição 4. Das posições 0 a 4, sabemos que temos caracteres válidos, e, portanto, podemos escrevê-los. Note a forma como inicializamos a string **str** com os caracteres 'J' 'o' 'a' 'o'  e '\0'  simplesmente declarando char str[10] = "Joao".

Devemos lembrar que o tamanho da string deve incluir o '\0' final. A biblioteca padrão do C possui diversas funções que manipulam strings. Estas funções são úteis pois não se pode, por exemplo, igualar duas strings:

string1=string2;        **/\* \*\*\* NÃO faça isto \*\*\* \*/**

Fazer isto é um desastre. **As strings devem ser igualadas elemento a elemento** ou deve-se usar uma função adequada para isso.

Quando vamos fazer programas que tratam de string muitas vezes podemos fazer bom proveito do fato de que uma string termina com '\0' (isto é, o número inteiro 0). Veja, por exemplo, o programa abaixo que serve para igualar duas strings (isto é, copia os caracteres de uma string para outro) :

#include <stdio.h>

#include <stdlib.h>

int main (){

int count;

char str1[10]="Joao", str2[10];

for (count=0; str1[count]; count++)

str2[count]=str1[count];

str2[count]='\0';

printf("String original = %s\n", str1);

printf("String copiado = %s\n", str2);

system("PAUSE");

return(0);

}

A condição no laço [**for**](../../Sistemas%20de%20Informação/Linguagem%20C/Programacao%20C/Tutorial%20UFMG/aulas/c430.html) acima é baseada no fato de que a string que está sendo copiada termina em '\0'. Quando o elemento encontrado em **str1[count]** é o '\0', o valor retornado para o teste condicional é falso (nulo). Desta forma a expressão que vinha sendo verdadeira (não zero) continuamente, torna-se falsa.

Em seguida é apresentado um programa em que são guardadas 19 letras em uma string e em seguida são deixados acessíveis apenas as 10 primeiras posições, ao colocar NULL (ou ‘\0’) na 11ª posição.

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

main(){

int x,tam, cont=0;

char string[20];

for(x=0; x<18; x++)

string[x] = 'A' + x;

string[10]=NULL;

printf("A string = %s\n",string);

system("PAUSE");

return(0);

}

Sempre é bom lembrar:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | A |  |  | A | \0 |  |
|  | ‘A’ |  |  | “A” | |  |

Vamos ver agora algumas funções básicas para manipulação de strings, sendo elas **gets, fgets, strpy, strcat, strlen, strcmp, strlwr e strupr.**

#### - gets

A função **gets()** lê uma string do teclado. Sua forma geral é:

*gets (nome\_da\_string);*

  O programa abaixo demonstra o funcionamento da função **gets()**:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

int main (){

char string[100];

printf ("Digite o seu nome: ");

gets (string);

printf ("\nOi %s. Seja bem vindo!\n",string);

system("PAUSE");

return(0);

}

#### - fgets

Como o gets pode provocar problemas de segurança ao permitir o armazenamento de caracteres além da capacidade da string, a solução mais segura é usar fgets, que limita o tamanho máximo a ser lido. Sua forma geral é

*fgets (nome\_da\_string, tamanho, stdin);*

#### O programa abaixo apresenta um exemplo com gets e fgets:

#### #include <stdio.h>

#### #include <stdlib.h>

#### #include <string.h>

#### int main (){

#### char nome1[21], nome2[21];

#### printf ("Digite um nome: ");

#### gets(nome1);

#### 

#### printf ("Digite outro nome: ");

#### fgets(nome2, 21, stdin); // 21 pois são 20 caracteres + \0

#### 

#### printf ("\nNomes: \n %s - %s\n",nome1, nome2);

#### system("PAUSE");

#### return(0);

#### }

Lembrar que o “<Enter>” será armazenado na string lida também.

#### - strcpy

Sua forma geral é:

*strcpy (string\_destino,string\_origem);*

A função **strcpy()** copia a string-origem para a string-destino. As funções apresentadas nestas seções estão no arquivo cabeçalho **string.h**. A seguir apresentamos um exemplo de uso da função **strcpy()**:

#### #include <stdio.h>

#### #include <stdlib.h>

#### #include <string.h>

#### int main (){

#### char str1[100],str2[100],str3[100];

#### printf ("Entre com uma string: ");

#### gets (str1);

#### strcpy (str2, str1); /\* Copia str1 em str2 \*/

#### strcpy (str3,"Você digitou a string "); /\* Copia a frase "Você digitou a string" em str3 \*/

#### printf ("\n\n%s%s\n",str3,str2);

#### system("PAUSE");

#### return(0);

#### }

#### - strcat

A função **strcat()** tem a seguinte forma geral:

*strcat (string\_destino,string\_origem);*

A string de origem permanecerá inalterada e será anexada ao fim da string de destino. Um exemplo:

#### #include <stdio.h>

#### #include <stdlib.h>

#### #include <string.h>

#### int main (){

#### char str1[100],str2[100];

#### printf ("Entre com uma string: ");

#### gets (str1);

#### strcpy (str2,"Voce digitou a string ");

#### strcat (str2, str1); /\*str2 armazenara' Voce digitou a string + o conteudo de str1 \*/

#### printf ("\n\n%s\n",str2);

#### system("PAUSE");

#### return(0);

#### }

#### - strlen

Sua forma geral é:

*strlen (string);*

A função **strlen()** retorna o comprimento da string fornecida. O terminador nulo não é contado. Isto quer dizer que, de fato, o comprimento do vetor da string deve ser um a mais que o inteiro retornado por **strlen()**.

Um exemplo do seu uso:

#### #include <stdio.h>

#### #include <stdlib.h>

#### #include <string.h>

#### int main (){

#### int size;

#### char str[100];

#### printf ("Digite uma string: ");

#### gets (str);

#### size=strlen (str);

#### printf ("\nA string que voce digitou tem tamanho %d\n",size);

#### system("PAUSE");

#### return(0);

#### }

#### - strcmp

Sua forma geral é:

*strcmp (string1,string2);*

A função **strcmp()** compara a string 1 com a string 2.

**ATENÇÃO!!** Se as duas forem **idênticas** a função retorna **zero**. Se elas forem **diferentes** a função retorna **não-zero**. Um exemplo da sua utilização:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

int main (){

char str1[100],str2[100], cont;

printf ("Entre com uma string: ");

gets (str1);

printf ("\n\nEntre com outra string: ");

gets (str2);

if (strcmp(str1,str2)) printf ("\n\nAs duas strings são diferentes.\n");

else printf ("\n\nAs duas strings são iguais.\n");

system("PAUSE");

return(0);

}

- **strlwr e strupr**

Quando desejarmos converter a string para maiúsculas ou para minúsculas usaremos essas funções, respectivamente. Uma das situações em que isso poderá ser útil é quando desejarmos comparar as strings.

Vejamos o exemplo:

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

main(){

char texto[15] = "Texto misturaDo";

printf("%s\n", strupr(texto));

printf("%s\n", strlwr(texto));

system("PAUSE");

return(0);

}

Existem muitas outras funções que podem ajudar na manipulação de strings, entre elas estão sscanf e sprintf

**- sscanf**

As vezes precisamos, em linguagem C, quebrar uma cadeia de caracteres (string) em pequenas partes para que estes dados sejam manipulados. A função sscanf pode ser utilizada para identificar o dia, e mês e o ano de uma data, conforme o programa exemplo abaixo:

#include <stdlib.h>

#include <stdio.h>

int main(){

int dia, mes, ano;

char data[11];

printf("Informe a data de nascimento (dd/mm/aaaa): ");

gets(data);

sscanf(data, "%d/%d/%d", &dia, &mes,&ano);

printf("dia=%d mes=%d ano=%d\n", dia, mes, ano);

system("PAUSE");

return(0);

}

|  |  |
| --- | --- |
| Sua forma geral é *sscanf(char str, frmt, & var1, ...)* |  |

***str:*** string de onde os dados serão lidos.  
***frmt***: formato usado na string lida – no exemplo foi o caractere “/”.

Em caso de sucesso, **a função retorna o número de itens lidos com sucesso, convertidos e armazenados**. O valor retornado não inclui campos que não foram armazenados. Se uma falha de correspondência acontecer, este número pode variar entre o valor esperado até zero. Retorna EOF se a função tentar ler um EOF (entrada vazia, por exemplo).

Para um correto funcionamento da função, recomenda-se usá-la juntamente com um if (como no exemplo abaixo), para testar se a entrada foi associada às variáveis corretamente. Caso ocorra algum erro na correspondência (entrada exceder ao tamanho do dado, tipo do dado diferente, etc), o erro é mostrado.

#include <stdlib.h>

#include <stdio.h>

int main() {

int dia, mes, ano, qtd;

char data[11];

printf("Informe a data de nascimento (dd/mm/aaaa): ");

gets(data);

qtd = sscanf(data, "%d/%d/%d", &dia, &mes,&ano);

if (qtd) printf("dia=%d mes=%d ano=%d\n", dia, mes, ano);

else printf("A data não foi informada no formato indicado.\n ");

system("PAUSE");

return(0);

}

**-sprintf**

A função sprintf é semelhante ao sscanf, a diferença é que invés de quebrar a string em variáveis, ela junta variáveis e monta tudo em uma string.

Sintaxe: *sprintf(char \*str, const char \*frmt, arg1, ...);*

***str***: é a variável string que receberá os dados formatados.  
***frmt:*** Pode ser usado qualquer caractere de formatação da função printf, como: %c, %s, %d, %f, etc, ou qualquer texto que desejar acrescentar à formatação.

#include <stdlib.h>

#include <stdio.h>

int main() {

int dia, mes, ano, qtd;

char data1[11], data2[11];

printf("Informe a data de nascimento (dd/mm/aaaa): ");

gets(data1);

qtd = sscanf(data1, "%d/%d/%d", &dia, &mes,&ano);

if (qtd) {

**sprintf(data2,"%d/%d/%d", mes, dia, ano);**

printf("\nData lida => dia=%d mes=%d ano=%d\n", dia, mes, ano);

printf("\nFormato americano=> %s\n", data2);

}

else printf("A data não foi informada no formato indicado.\n ");

system("PAUSE");

return(0);

}

Acentuação / Leitura de caracteres especiais

A linguagem C não aceita palavras acentuadas se forem digitadas diretamente do teclado. Caso façamos isso, os valores apresentados serão símbolos diferentes dos desejados. Para inserirmos uma letra acentuada, por exemplo, ou ç (cedilha), utilizamos códigos no sistema octal. Esse sistema é composto pelos algarismos: 0, 1, 2, 3, 4, 5, 6 e 7, ou seja, os oito primeiros algarismos do sistema decimal. Os caracteres, tanto letras como números, podem ser totalmente escritos utilizando o sistema octal, na linguagem C.

Abaixo está a lista de todos os valores em octal das vogais e dos cedilhas:

Á = \665 Ú = \751 ò = \225

À = \667 Ù = \753 õ = \744

Ã = \707 Û = \752 ô = \223

Â = \666 á = \240 ú = \243

É = \220 à = \205 ù = \227

È = \724 ã = \706 û = \226

Ê = \722 â = \203 Ç = \200

Í = \726 é = \202 ç = \207

Ì = \736 è = \212

Î = \214 ê = \210

Ó = \740 í = \641

Ò = \743 ì = \736

Ô = \742 î = \727

Õ = \745 ó = \242

#include <stdio.h>

#include <stdlib.h>

#include <locale.h>

int main() {

char palavra1[12], palavra2[12], procura[]="Computa**\207\706**o";

int cont, conta=0;

setlocale(LC\_ALL,"Portuguese");

printf("Informe 2 palavras:\n");

fgets(palavra1,12, stdin);

fflush(stdin);

fgets(palavra2,12, stdin);

// if(palavra2[strlen(palavra2)-1]=='\n'){ palavra2[strlen(palavra2)-1]='\0';}

// if(palavra1[strlen(palavra1)-1]=='\n'){ palavra1[strlen(palavra1)-1]='\0';}

if(!strcmp(palavra2, procura)) printf("É Computação\n");

else printf("Não é Computação\n");

printf("Tamanho da primeira palavra = %d\n", strlen(palavra1));

printf("Tamanho da segunda palavra = %d\n", strlen(palavra2));

//strcpy(palavra1, "Programação");

//printf("%s\n", palavra1);

strcat(palavra1, "XX"); // outra prova "XX" no final – retirando ou não o \n

printf("%s\n", palavra1);

return 0;

}

**EXERCÍCIOS**

8.1 Faça um programa que leia três palavras (máximo 10 caracteres) e armazene cada palavra em uma string. Caso a primeira string seja igual a segunda ou a terceira, deve ser mostrada uma mensagem avisando. Caso contrário, mostre uma quarta string com as 3 primeiras concatenadas. (usar a biblioteca string.h)

8.2 Ler uma palavra e mostrá-la na tela tantas vezes quantos forem seus caracteres

8.3 Ler uma frase e mostrar a primeira letra em maiúscula e as demais em minúscula. Usar as funções *toupper* e *tolower*, mas lembrar que elas funcionam para um caractere de cada vez.

8.4 Ler uma string e mostrar em letras maiúsculas e depois em letras minúsculas. Usar as funções *strlwr* e *strupr*.

8.5 Ler uma string do teclado (máximo 50 símbolos), contar e mostrar quantas letras ‘A’ (maiúsculas ou minúsculas) existe na string lida.