

Laboratório de programação

Vetores e Strings

Universidade Estadual Vale do Acaraú – UVA

Paulo Regis Menezes Sousa

paulo_regis@uvanet.br

Arrays

Declarando Arrays

Strings

Arrays em funções

Exercícios

- Para **declarar um array** se especifica o tipo de cada elemento e o numero de elementos exigidos por array de forma que o computador possa reservar a quantidade apropriada de memória.

```
int v[12];
```

- Pode-se reservar memória para vários arrays com uma única declaração.

```
int a[10], b[15];
```

- Os arrays podem ser declarados para conter outros tipos de dados. Por exemplo, pode ser usado um array do tipo **char** para armazenar uma string de caracteres.

- O programa abaixo usa uma estrutura de repetição **for** para inicializar com zeros os elementos de um array inteiro **n** de dez elementos, e imprime o array sob a forma de uma tabela.

Código 1: init-array-zeros.c

```
1  #include <stdio.h>
2  int main(){
3      int n[10], i;
4      for (i = 0; i <= 9; i++) // inicializa o array
5          n[i] = 0;
6      printf("Índice Valor\n");
7      for(i =0;i<=9;i++)
8          printf ("%6d %5d\n", i, n[i]);
9      return 0;
10 }
```

- Um array pode ser inicializado na declaração com um sinal de igual e uma lista de valores separada por vírgulas (entre chaves).
- Se houver menos valores que o número de elementos, os elementos restantes são inicializados com zero. Por exemplo:

```
int n[10] = {0}; //inicializa todo o vetor com zeros
```

Código 2: init-array-decl.c

```
1  #include <stdio.h>
2  int main(){
3      int i, n[10] = {32, 27, 64, 18, 95, 14, 90, 70, 60, 37};
4      printf("Índice Valor\n");
5      for(i = 0; i <= 9; i++)
6          printf ("%6d %5d\n", i, n[i]);
7      return 0;
8  }
```

Warning!

É importante lembrar que os **arrays não são inicializados automaticamente com zero**. O programador deve inicializar pelo menos o primeiro elemento com zero para que os elementos restantes sejam automaticamente zerados.

Warning!

A seguinte declaração de array

```
int n[5] = {32 ,27 ,67 ,18 ,95 ,14};
```

causaria um erro de sintaxe porque há seis inicializadores e apenas 5 elementos no Array.

- Se o tamanho do array for omitido de uma declaração com uma lista de inicializadores, o número de elementos do array será o número de elementos da lista de inicializadores. Por exemplo,

```
int n[] = {1 , 2 , 3 , 4 , 5};
```

criaria um array com cinco elementos.

- A diretiva `#define` do pré-processador define uma constante simbólica.
- Uma constante simbólica é um identificador que é substituído por um texto de substituição pelo pré-processador antes do programa ser compilado.

`#define TAMANHO 10`

- Quando o programa é pré-processado, todas as ocorrências da constante simbólica **TAMANHO** são substituídas pelo texto de substituição **10**.

Código 3: symbolic-constant.c

```
1  #include <stdio.h>
2  #define TAM 100
3
4  int main() {
5      int s[TAM], j;
6
7      for(j = 0; j < TAM; j++) /* Define os valores */
8          s[j] = 2 + 2 * j;
9
10     printf("Índice Valor\n");
11     for(j = 0; j < TAM; j++)
12         printf("%6d %5d\n", j, s[j]);
13     return 0;
14 }
```

Warning!

Se a diretiva `#define` anterior fosse encerrada com um ponto-e-vírgula, todas as ocorrências da constante simbólica **TAMANHO** no programa seriam substituídas pelo texto **10**; pelo pré-processador. Isso poderia levar a erros de sintaxe em tempo de compilação, ou a erros lógicos em tempo de execução.

Lembre-se de que o pré-processador não é C – ele é apenas um manipulador de textos.

Exercício 16

Foi perguntado a quarenta alunos o nível de qualidade da comida na cantina estudantil, em uma escala de 0 a 10 (0 significa horrorosa e 10 significa excelente).

Coloque as quarenta respostas em um array inteiro e resuma os resultados da pesquisa.

Código 4: 16.c

```
1  #include <stdio.h>
2  #define TAM_RES 40
3  #define TAM_FREQ 11
4  int main(){
5      int opiniao, nivel, frequencia[TAM_FREQ] = {0};
6      int resp[TAM_RES] = {1,2,6,4,8,5,9,7,8,10, 1,6,3,8,6,10,3,8,2,7,
                           6,5,7,6,5,7,6,0,6,7, 5,6,6,5,6,7,5,6,4,8};
7
8      for (opiniao = 0; opiniao < TAM_RES; opiniao++)
9          frequencia[resp[opiniao]]++;
10
11     printf("Nivel Frequencia\n");
12     for (nivel = 0; nivel < TAM_FREQ; nivel++)
13         printf("%5d%10d\n", nivel, frequencia[nivel]);
14
15     return 0;
16 }
```

Exercício 17

Crie um programa que lê números inteiros de um array e imprime um histograma horizontal. Como no exemplo a seguir:

Elemento	Histograma
0	*****
1	***
2	*****
3	*****
4	*****
5	*****
6	*****
7	*****
8	*****
9	*

Código 5: 17.c

```
1  #include <stdio.h>
2  #define TAM 10
3  int main(){
4      int n[TAM] = {19,3,15,7,11,9,13,5,17,1};
5      int i, j;
6
7      printf("Elemento Histograma\n");
8
9      for (i = 0; i <= TAM - 1; i++) {
10         printf("%8d ", i);
11
12         for (j = 1; j <= n[i]; j++) /* imprime uma barra */
13             printf("*");
14
15         printf("\n");
16     }
17     return 0;
18 }
```

Strings

Strings são cadeias de caracteres e na linguagem C estas cadeias são tratadas na forma de arrays do tipo `char`.

- Um array de caracteres pode ser inicializado usando uma string. Por exemplo:

```
char string1[] = "primeiro";
```

- O tamanho do array `string1` na declaração anterior é determinado pelo compilador, com base no comprimento da string.

Caractere de terminação de string

Em C uma string deve conter um caractere especial, chamado *caractere nulo*, que indica o fim da string. Esse caractere nulo é representado literalmente como `'\0'`.

- Os arrays de caracteres também podem ser inicializados com caracteres isolados

```
char string1[] = {'p','r','i','m','e','i','r','o','\0'};
```

- Podemos também fornecer uma string diretamente a um array de caracteres usando `scanf` e o formatador `%s`. Por exemplo:

```
char string2[20];
```

armazena uma string de 19 caracteres mais um caractere nulo de término.

- A instrução

```
scanf("%s", string2);
```

lê uma string do teclado e a armazena em `string2`.

Warning!

Observe que o nome do array é passado para **scanf** sem ser precedido por **&**, usado com outras variáveis.

```
scanf("%s", string2);
```

o nome de um array é o endereço do início do array; portanto, o **&** não é necessário.

Warning!

A função **scanf** lê caracteres do teclado até que o primeiro caractere em branco seja encontrado, ela não é limitada pelo tamanho do array.

- Para passar um argumento array a uma função, especifique o nome do array sem colocar colchetes. Por exemplo

```
1      int vetor[20];  
2      // ...  
3      ordenarVetor(vetor, 20);
```

- Para uma função receber um array, sua lista de parâmetros deve especificar que um array será recebido. Por exemplo, a declaração da função **ordenarVetor** pode ser escrita como

```
void ordenarVetor(int b[], int tamanho){...
```

- Omitindo os nomes dos parâmetros no **protótipo da função** teríamos

```
void ordenarVetor(int [], int);
```

Código 6: pow-array.c

```
1  #include <stdio.h>
2  #define LENGTH 5
3
4  void powArray(int [], int);
5  void powValue(int);
6
7  int main() {
8      int a[LENGTH] = {0, 1, 2, 3, 4}, i, j;
9
10     powArray(a, LENGTH);
11
12     for (i = 0; i <= LENGTH - 1; i++)
13         printf("%d ", a[i]);
14     printf("\n");
```

```
16     powValue(a[3]);
17     printf("In main() - > %d\n", a[3]);
18
19     return 0;
20 }
21
22 void powArray(int b[], int length){
23     int j;
24     for (j = 0; j < length; j++)
25         b[j] = b[j]*b[j];
26 }
27
28 void powValue(int e){
29     e = e*e;
30     printf("In powValue() -> %d\n", e);
31 }
```

- A linguagem C fornece o qualificador especial de tipo **const** para evitar a modificação do valor de um parâmetro em uma função.
- A tentativa de modificar um parâmetro **const** resulta em um erro em tempo de compilação.

```
1  #include <stdio.h>
2
3  void failPowArray(const int b[], int length){
4      int j;
5      for (j = 0; j < length; j++)
6          b[j] = b[j]*b[j];
7  }
8  int main(){
9      int a[3] = {10, 20, 30};
10     failPowArray(a, 3);
11     printf ("%d %d %d\n", a[0], a[1], a[2]);
12     return 0;
13 }
```

- O programa a seguir classifica os valores dos elementos de um array a de dez elementos na ordem ascendente.
- A técnica utilizada é chamada classificação de bolhas (*bubble sort*) ou classificação de submersão (*sinking sort*).
- Nessa técnica os valores menores “sobem” gradualmente para o topo do array, da mesma forma que bolhas de ar sobem na água, enquanto os valores maiores afundam “submergem” para a parte de baixo do array.

Código 7: bubble-sort.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #define TAM 7
4
5  void printVector(int [], int);
6
7  int main(){
8      int vetor[TAM] = {7,3,2,6,4,1,5}, x = 0, y = 0, aux = 0;
9
10     printVector(vetor, TAM);
11
12     for (x = 0; x < TAM-1; x++ ) {
13         for(y = 0; y < TAM-1-x; y++) {
```

```
14         if (vetor[y] > vetor[y+1]) {
15             printf("%d<->%d ", vetor[y], vetor[y+1]);
16
17             aux = vetor[y];
18             vetor[y] = vetor[y+1];
19             vetor[y+1] = aux;
20         }
21     }
22     printf("\n");
23     printVector(vetor, TAM);
24 }
25 return 0;
26 }
27 void printVector(int vetor[], int n){
28     int x;
29     for(x = 0; x < n; x++ )
30         printf("[%d] ", vetor[x]);
31     printf("\n");
32 }
```


Exercício 18

Faça um programa que preencha dois vetores de dez posições cada, determine e mostre um terceiro contendo os elementos dos dois vetores anteriores ordenados de maneira decrescente.

Exercício 19

Faça um programa que preencha três vetores com dez posições cada um: o primeiro vetor, com os nomes de dez produtos; o segundo vetor, com os códigos dos dez produtos; e o terceiro vetor, com os preços dos produtos. Mostre um relatório apenas com o nome, o código, o preço e o novo preço dos produtos que sofrerão aumento. Sabe-se que os produtos que sofrerão aumento são aqueles que possuem código par ou preço superior a R\$ 1.000,00. sabe-se ainda que, para os produtos que satisfazem as duas condições anteriores, código e preço, o aumento será de 20%; para aqueles que satisfazem apenas a condição de código, o aumento será de 15%; e para aqueles que satisfazem apenas a condição de preço, o aumento será de 10%.

Exercício 20

Faça um programa que receba o nome de cinco produtos e seus respectivos preços. Calcule e mostre:

1. a quantidade de produtos com preço inferior a R\$ 50,00;
2. o nome dos produtos com preço entre R\$ 50,00 e R\$ 100,00;
3. a média dos preços dos produtos com preço superior a R\$ 100,00

Exercício 21

Faça um programa que leia um vetor com quinze posições para números inteiros. Crie, a seguir, um vetor resultante que contenha todos os números primos do vetor digitado. Escreva o vetor resultante.