

Programação Estruturada

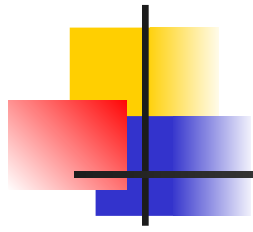
- Encontro 02 -

Sistemas de Informação
Prof.º Philippe Leal
philippeleal@yahoo.com.br



Vetores

- As variáveis declaradas até agora são capazes de armazenar um único valor por vez. Sempre que atribuímos um novo valor a uma variável, o valor anterior é perdido.
- Isso ocorre porque cada variável está associada a uma única posição de memória e dentro dela é possível armazenar um valor do tipo especificado. Assim, para usar mais de um valor, é preciso usar mais de uma variável.
- Um array ou vetor é a forma mais simples e comum de dados estruturados da linguagem C. Trata-se simplesmente de um conjunto de variáveis de um mesmo tipo, com a vantagem de estarem todas associadas ao mesmo nome e igualmente acessíveis por índices consecutivos.



Vetores

Declaração

- A forma geral para declarar um vetor é:

tipo nome_vetor[tamanho];

Exemplos:

int num[10]; //declara o vetor *num* com 10 números do tipo int

float valores[20]; //declara o vetor *valores* com 20 números do tipo float

double vet[100]; //declara o vetor *vet* com 100 números do tipo double

- Obs.: Na linguagem C, a numeração do índice do vetor começa sempre do 0 (zero) e termina sempre em $n-1$, onde n é o tamanho do vetor.



Vetores

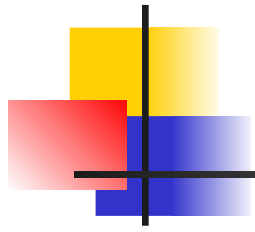
- Após a declaração é reservado espaço na memória para todos os elementos do vetor:

```
int num[10]; //declara o vetor num com 10 números do tipo int
```

- Em C, somente podemos acessar e processar elemento a elemento do vetor. Não tem como processar todos os elementos de uma só vez.
- Cada elemento é acessado usando `num[i]`, onde `i` é o índice do elemento:

	0	1	2	3	4	5	6	7	8	9
num	12	5	6	27	8	19	20	24	15	3

- É válido declarar um vetor de tamanho n e armazenar valores nas primeiras m posições, desde que m não ultrapasse n .



Vetores

Inicialização

- Vimos que vetores são variáveis. Assim, a declaração de um vetor apenas reserva espaço na memória e não associa a eles nenhum valor.
- Seu valor inicial é o lixo de memória contido no espaço reservado para ele.
- A depender do programa que estamos construindo, é preciso iniciar todas as posições do vetor com algum valor predefinido.
- Vejamos algumas maneiras de inicializar um vetor:



Vetores

```
1 #include <stdio.h>
2
3 #define TAM 6
4
5 int main(){
6
7     int i,
8         vet1[4],
9         vet2[5] = {1,2,3,4,5},
10        vet3[TAM];
11
12     /* Inicializar vet1 */
13     vet1[0] = 7; vet1[1] = 3; vet1[2] = 4; vet1[3] = 1;
14
15     /* Inicializar vet3 */
16     printf("\nDigite os %d valores do vetor vet3: ", TAM);
17     for(i = 0; i < TAM; i++)
18         scanf("%d", &vet3[i]);
19
20     /* Imprimir os vetores */
21     printf("\n\nVetor 1: ");
22     for(i = 0; i < 4; i++)
23         printf("%d ", vet1[i]);
24
25     printf("\n\nVetor 2: ");
26     for(i = 0; i < 5; i++)
27         printf("%d ", vet2[i]);
28
29     printf("\n\nVetor 3: ");
30     for(i = 0; i < TAM; i++)
31         printf("%d ", vet3[i]);
32
33     printf("\n");
34     return 0;
35 }
```



String

- Na linguagem C, uma string é uma cadeia (vetor) de caracteres terminada pelo caractere nulo: `'\0'`.
- Para especificar um vetor para armazenar uma string, deve-se sempre reservar um espaço a mais para o caractere nulo.
- Por exemplo, para armazenar uma string de até 40 caracteres, deve-se declarar um vetor de 41 caracteres: `char string[41];`
- Vejamos como declarar e inicializar uma string:



String

```
1 #include <stdio.h>
2
3 int main(){
4
5     char frase1[10] = {'M', 'a', 'r', 'i', 'a', '\0'},
6         frase2[30] = "Programacao Estruturada",
7         frase3[]    = "Linguagem C",
8         nome[40];
9
10    /* frase1 */
11    printf("\nfrase1: ");
12    printf("%s", frase1);
13
14    /* frase2 */
15    printf("\n\nfrase2: ");
16    printf("%s", frase2);
17
18    /* frase3 */
19    printf("\n\nfrase3: ");
20    printf("%s", frase3);
21
22    /* nome */
23    printf("\n\nDigite seu nome: ");
24    scanf("%s", nome);
25
26    printf("\nSeu nome e: %s\n\n", nome);
27
28    /* Para digitar uma frase com espacos utilize:
29       scanf("%[^\n]s", nome); ou fgets(nome, 40, stdin);
30    */
31
32    return 0;
33 }
```




String

Algumas funções da biblioteca <string.h>

- **char *strcat(char *dest, const char *orig):** concatena a string *orig* ao final de *dest*. O primeiro caractere de *orig* substitui o caractere nulo de *dest*.
- **int strcmp(const char *str1, const char *str2):** compara as duas strings. Retorna zero se as duas strings forem iguais.
- **size_t strlen(const char *str1):** retorna o comprimento da string (sem contar o caractere nulo). Não confundir o tamanho da string com o tamanho do vetor que armazena a string.
- **char *strcpy(char *dest, const char *orig):** copia a string *orig* para *dest*. A string *dest* deve ter espaço suficiente para armazenar a *orig*.



Passando Vetores por Parâmetro

- Quando passamos vetores por parâmetro, independente do seu tipo, o que é de fato passado para a função/procedimento é o endereço do primeiro elemento do vetor.
- Vetores são sempre passados por referência para uma função/procedimento.
- Vejamos como podemos passar um vetor por parâmetro para uma função/procedimento.



Passando Vetores por Parâmetro

- Vetores podem ser passados por parâmetro para funções/procedimentos de três maneiras:

✓ `int funcao(float *vetor, int tamanho)` //Como ponteiro

✓ `void procedimento(int vetor[10], int tamanho)` //Como um vetor dimensionado

✓ `float funcao(float vetor[], int tamanho)` //Como um vetor não-dimensionado



Passando Vetores por Parâmetro

```
1 #include <stdio.h>
2 |
3 void preencheVetor(int *vetor, int tam);
4 void imprimeVetor(int *vetor, int tam);
5
6 int main(){
7
8     int vetor[15];
9
10
11     preencheVetor(vetor, 15);
12
13     imprimeVetor(vetor, 15);
14
15
16     return 0;
17 }
18
19 void preencheVetor(int *vetor, int tam){
20
21     int i;
22
23     printf("\nDigite os %d elementos do vetor: ", tam);
24     for(i = 0; i < tam; i++)
25         scanf("%d", &vetor[i]);
26
27 }
28
29 void imprimeVetor(int *vetor, int tam){
30
31     int i;
32
33     printf("\n\nElementos do vetor: ");
34     for(i = 0; i < tam; i++)
35         printf("%d ", vetor[i]);
36
37     printf("\n\n");
38
39 }
```



Passando Vetores por Parâmetro

```
1 #include <stdio.h>
2 |
3 void preencheVetor(int *vetor);
4 void imprimeVetor(int *vetor);
5
6 #define TAM 15
7
8 int main(){
9
10     int vetor[TAM];
11
12     preencheVetor(vetor);
13
14     imprimeVetor(vetor);
15
16     return 0;
17 }
18
19 void preencheVetor(int *vetor){
20
21     int i;
22
23     printf("\nDigite os %d elementos do vetor: ", TAM);
24     for(i = 0; i < TAM; i++)
25         scanf("%d", &vetor[i]);
26
27 }
28
29 void imprimeVetor(int *vetor){
30
31     int i;
32
33     printf("\n\nElementos do vetor: ");
34     for(i = 0; i < TAM; i++)
35         printf("%d ", vetor[i]);
36
37     printf("\n\n");
38
39 }
```



Exercícios

- 1)** Faça um programa que leia 15 números reais e armazene-os em um vetor e depois imprima somente os números que estão nas posições pares desse vetor.
- 2)** Faça um programa que leia 10 números inteiros e imprima-os na ordem inversa. Faça dois procedimentos: um para preencher o vetor e outro para imprimi-lo em ordem inversa. A declaração do vetor tem que ser realizada na função *main*.