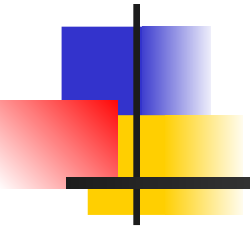
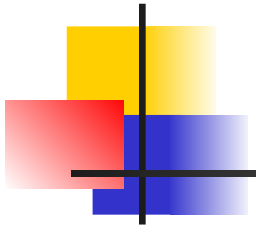


Programação Estruturada

Aula 01 - Vetores

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.

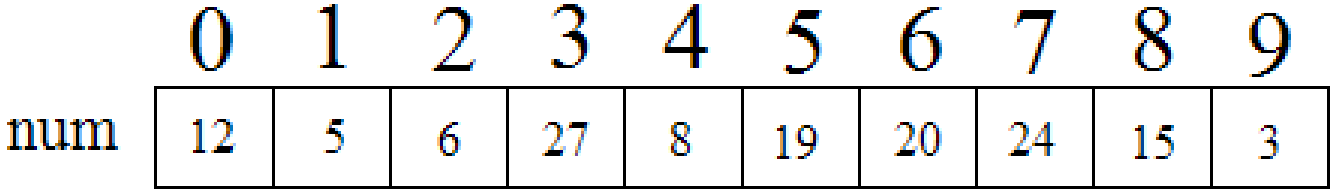
Vetores

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

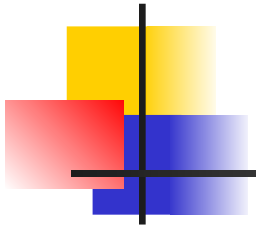
  int num[10]; //declara o vetor ***num*** com 10 números do tipo int

- 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:



- 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.

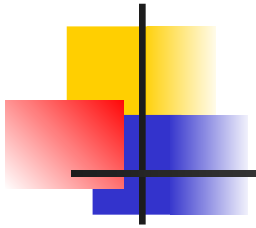
- Seu valor inicial é o lixo de memória contido no espaço reservado para ele.

- 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

```c
#include <stdio.h>

#define TAM 6

int main(){

    int i,
        vet1[4],
        vet2[5] = {1,2,3,4,5},
        vet3[TAM];

    /* Inicializar vet1 */
    vet1[0] = 7; vet1[1] = 3; vet1[2] = 4; vet1[3] = 1;

    /* Inicializar vet3 */
    printf("\nDigite os %d valores do vetor vet3: ", TAM);
    for(i = 0; i < TAM; i++)
        scanf("%d", &vet3[i]);

    /* Imprimir os vetores */
    printf("\n\nVetor 1: ");
    for(i = 0; i < 4; i++)
        printf("%d ", vet1[i]);

    printf("\n\nVetor 2: ");
    for(i = 0; i < 5; i++)
        printf("%d ", vet2[i]);

    printf("\n\nVetor 3: ");
    for(i = 0; i < TAM; i++)
        printf("%d ", vet3[i]);

    printf("\n");
    return 0;
}
```



String

- Na linguagem C, uma string é uma cadeia (vetor) de caracteres terminada pelo caractere nulo: ‘\0’.

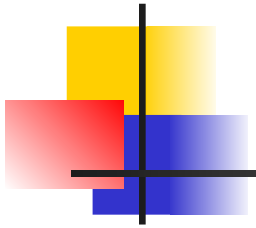
- Para especificar um vetor para armazenar uma string, deve-se sempre reservar um espaço a mais para o caractere nulo.

- 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

```c
#include <stdio.h>

int main(){

    char frase1[10] = {'M', 'a', 'r', 'i', 'a', '\0'},
         frase2[30] = "Programacao Estruturada",
         frase3[]   = "Linguagem C",
         nome[40];

    /* frase1 */
    printf("\nfrase1: ");
    printf("%s", frase1);

    /* frase2 */
    printf("\n\nfrase2: ");
    printf("%s", frase2);

    /* frase3 */
    printf("\n\nfrase3: ");
    printf("%s", frase3);

    /* nome */
    printf("\n\nDigite seu nome: ");
    scanf("%s", nome);

    printf("\nSeu nome e: %s\n\n", nome);

    /* Para digitar uma frase com espacos utilize:
       scanf("%[^\n]s", nome);
    */

    return 0;
}
```



String

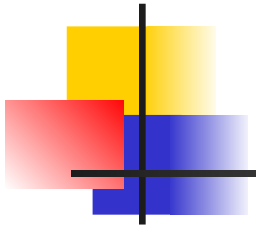
Algumas funções da biblioteca <string.h>

- **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

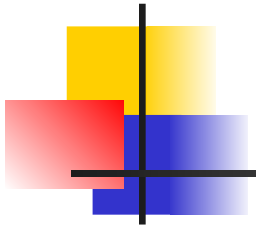
  - float funcao(float vetor[], int tamanho) //Como um vetor não-dimensionado

Passando Vetores por Parâmetro

```c
#include <stdio.h>

void preencheVetor(int *vetor, int tam);
void imprimeVetor(int *vetor, int tam);

int main(){

    int vetor[15];


    preencheVetor(vetor, 15);

    imprimeVetor(vetor, 15);


    return 0;
}

void preencheVetor(int *vetor, int tam){

    int i;

    printf("\nDigite os %d elementos do vetor: ", tam);
    for(i = 0; i < tam; i++)
        scanf("%d", &vetor[i]);

}

void imprimeVetor(int *vetor, int tam){

    int i;

    printf("\n\nElementos do vetor: ");
    for(i = 0; i < tam; i++)
        printf("%d ", vetor[i]);

    printf("\n\n");

}
```

Passando Vetores por Parâmetro

```c
#include <stdio.h>

void preencheVetor(int *vetor);
void imprimeVetor(int *vetor);

#define TAM 15

int main(){

    int vetor[TAM];

    preencheVetor(vetor);

    imprimeVetor(vetor);

    return 0;
}

void preencheVetor(int *vetor){

    int i;

    printf("\nDigite os %d elementos do vetor: ", TAM);
    for(i = 0; i < TAM; i++)
        scanf("%d", &vetor[i]);

}

void imprimeVetor(int *vetor){

    int i;

    printf("\n\nElementos do vetor: ");
    for(i = 0; i < TAM; i++)
        printf("%d ", vetor[i]);

    printf("\n\n");

}
```



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*.

3) Faça um programa que leia 20 números inteiros e armazene-os em um vetor. Depois leia um número inteiro x e informe se ele pertence ou não ao vetor. Faça um **procedimento** para preencher o vetor e uma **função** para a verificação. A leitura do número x, a declaração do vetor e a impressão da informação (se ele pertence ou não ao vetor) têm que ser realizadas na função *main*.