#### Vetores, *Strings* e Matrizes em C

**Introdução**

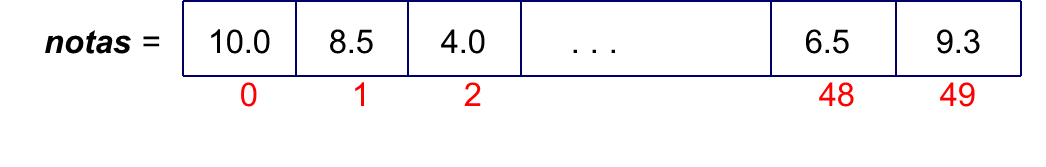
Nos capítulos anteriores focou-se muito nas estruturas de controle de execução de programas na linguagem C. Foram estudadas e praticadas estruturas sequenciais, estruturas de decisão e seleção e de repetição. Entretanto a manipulação da memória se restringiu ao uso de variáveis de tipos primitivos, que permitem o armazenamento de um único valor a cada instante de tempo.

Este capítulo vai focar uma forma de armazenamento de coleções de dados. Neste contexto, uma importante estrutura de armazenamento de informações na memória principal são os vetores e matrizes. Estes são conhecidos como ***estruturas homogêneas***, uma vez que permitem armazenamento e manipulação de dados de um mesmo tipo.

Como exemplo, podemos citar a necessidade de tratar listas de dados, como:

* + - Notas de uma turma de 50 alunos;
    - Nomes de 1.000 produtos vendidos em uma loja;
    - Cadeia de 100 caracteres.

Todos os casos acima são adequadamente representados por um ***arranjo*** ou ***vetor***. Graficamente pode-se representar um vetor como na figura a seguir, que descreve o exemplo das notas da turma de alunos:



Diz-se então que *notas* é uma estrutura ***homogênea unidimensional*** de dados:

* + - * ***Homogênea***: pois todos os componentes são do mesmo **tipo de dados** (*float* ou *double*);
      * ***Unidimensional***: pois os elementos indexados em uma única linha, formando uma sequência de dados.
      * Na figura, as caixinhas representam cada elemento de dados e a numeração abaixo das caixinhas representam o índice de cada elemento, que, na linguagem C começa sempre pela posição **0**. Por exemplo, o elemento de índice **1** armazena o valor de **nota = 8.5**.
      * O vetor utiliza sempre posições contíguas de memória.

**Vetores na Linguagem C**

A declaração de um vetor na linguagem C é muito parecida com a declaração de uma variável simples ou não estruturada, bastando apenas acrescentar entre colchetes o tamanho do vetor, ou seja, a quantidade de elementos que o programandor deseja que o vetor tenha.

**Sintaxe**:

**tipo nome[tamanho];**

**Semântica:**

É alocada na memória, em posições contíguas, uma variável de nome ***nome*** e área igual à quantidade de bytes do tipo ***tipo*** vezes o ***tamanho***declarado*.*

**Exemplo:**

**float notas[50];**

**int nums [10];**

**char palavra [20];** *// neste caso, um exemplo de vetor que forma uma “string”*

**Referenciando elementos do vetor na lógica de programas C:**

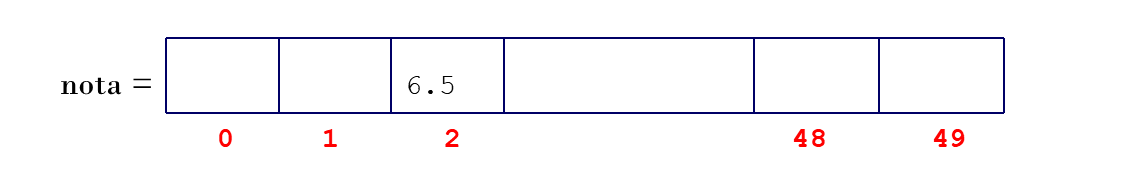
Um vetor, como qualquer variável, é declarado para armazenar dados e ser manipulado pelos programas. Porém, o vetor não é manipulado como um todo. O que é manipulável é cada elemento do vetor, um de cada vez. Para referenciar o elemento do vetor, basta escrever o nome do vetor, seguido da posição do elemento, entre colchetes.

**Exemplo**:

Atribuido valor **6.5** ao elemento de índice **2** do vetor *nota*:

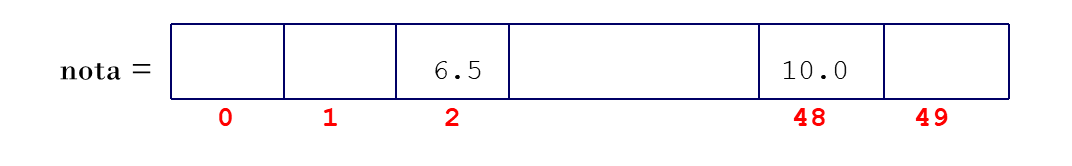
nota[2] = 6.5;

Esta instrução produz o seguinte resultado:



Agora, armazenando na posição **48** do vetor *nota* um valor informado pelo teclado:

scanf(“%d”, &nota[48]);// suponha que seja informado o valor **10.0**



Elementos de vetores também podem ser utilizados em expressões aritméticas ou lógicas, como qualquer variável. Exemplo: usando um elemento do vetor *nota* como operando de uma expressão:

for(int i = 0; i < 50; i++)

Soma = Soma + **nota[i]**;

Exemplo de um programa completo que lê a nota de 20 alunos e depois verifica os alunos aprovados, isto é, alunos com nota ≥ 5,0:

void main (){

float nota[20];

int i;

for(i = 0; i < 20; i++){ // laço que lê as 20 notas e carrega o vetor

printf(“\nInforme nota do aluno nr. %d: “, i+1);

scanf(“%d”, &nota[i]);

}

for(i = 0; i < 20; i++){ // laço que verifica os aprovados/reprovados

if(nota[i] >= 5)

printf(“\nAluno nr. %d aprovado com %.1f.“, i+1,nota[i]);

else

printf(“\nAluno nr. %d reprovado com %.1f.“,i+1,nota[i]);

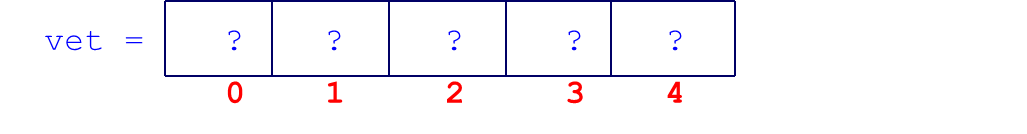
}

}

**Inicialização de vetor na linguagem C:**

Ao ser declarado, um vetor não é iniciado, ficando seus elementos preenchidos com valores indefinidos. Exemplo:

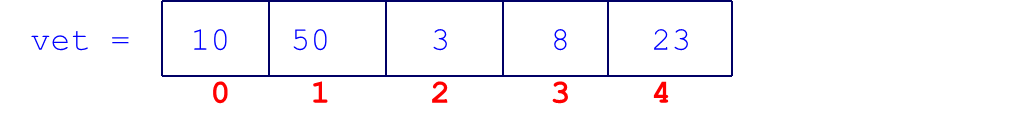
int vet[5];



Para iniciar um vetor, pode-se utilizar a construção a seguir:

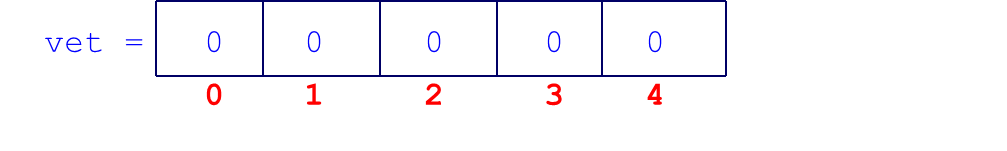
int vet[5] = {10, 50, 3, 8, 23};

Neste caso, o vetor é devidamente iniciado como na imagem abaixo:

****

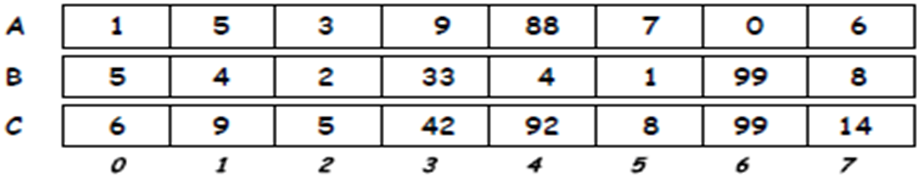
Se definir o vetor nas classes de armazenamento ***static*** ou ***extern***, o vetor é iniciado com valor *default* do tipo. Exemplo:

static int vet[5];

****

**Exemplo de um programa completo aplicando *vetor*:**

Programa que lê dados de dois vetores, A e B, com oito elementos inteiros cada e gera um 3º vetor, C, com a soma dos elementos dos dois, mostrando seu conteudo. Exemplo:



void main(){

static int A[] = {1, 5, 3, 9,88, 7, 0, 6};

static int B[8] = {5, 4, 2, 3, 4, 1,99, 8};

int i, C[8];

for(i = 0; i < 8; i++)

C[i] = A[i] + B[i];

printf(“\n C = { “);

for(i = 0; i < 8; i++

printf(“%d, “, C[i]);

printf(“ }; \n“);

}

***Strings* na Linguagem C**

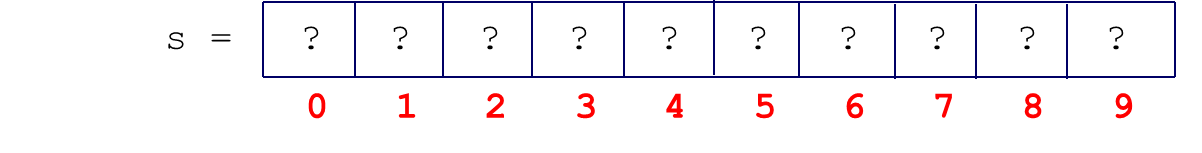
***Strings*** são cadeias de caracteres, ou seja, sequências de valores do tipo ***char***. Em muitas linguagens, *String* é um tipo de dado *primitivo*, porém a linguagem **C** não oferece este tipo de dado.

Em **C**, *String* é implementado como um *vetor de caracteres*.

Exemplo:

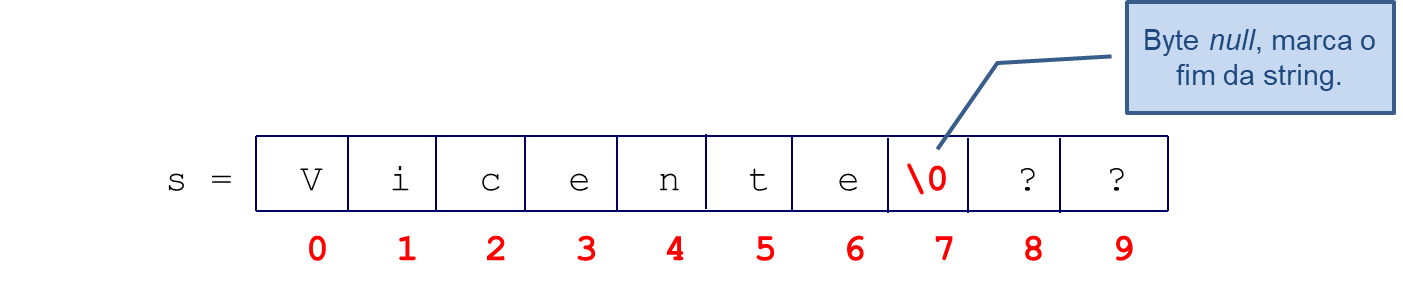
char s[10];

A instrução acima produz um vetor de caracteres que pode ser abstraído na figura a seguir:



Um vetor de *char* pode ser iniciado como no exemplo abaixo:

char s[10] = “Vicente”;



... mas não pode ser atribuído assim:

char s[10];

s = “Vicente”; **/\* ERRO! \*/**

Por outro lado, pode ser atribuído assim:

char s[10];

s[0] = 'A';

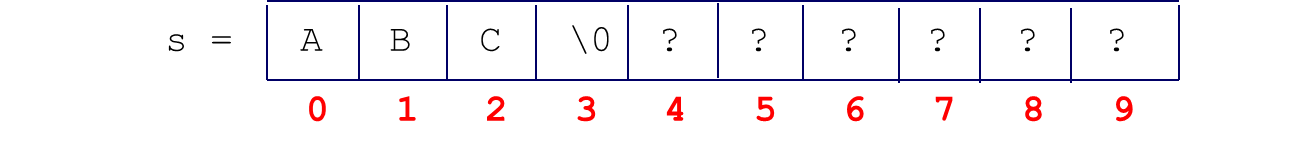
s[1] = 'B';

s[2] = 'C';

s[3] = '\0';

s[4] = 'D';

O conteúdo de s fica:



Note que o caracter 'D', atribuído na posição 4 foi perdido! Por que????

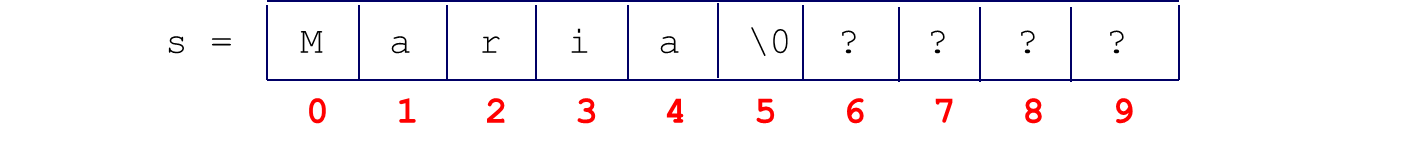
Como já foi visto anteriormente, para entrar com uma *string* pelo teclado, a biblioteca **<stdio.h>** oferece a função **gets( )**:

char s[10];

printf(“Informe um cadeia qualquer: “);

gets(s);

Supondo que o usuário digite a cadeia “Maria<enter>”, o conteudo de ***s*** será, após a execução do código acima, como na figura a seguir, onde pode ser observado que o caractere '\0', que define o fim da string, é acrescentado automaicamente:



A biblioteca **<stdio.h>** oferece a função **puts( ),** que imprime na tela uma string especificada como parâmetro, incluíndo uma quebra de linha ('\n' ). Exemplo:

printf(“S = “);

puts(s);

O código acima produz a saída:

*S = Maria*

\_ (com o cursor indo para a linha de baixo).

**Manipulando *strings* em C**

A biblioteca <string.h> oferece uma série de funções para manipulação de strings. A seguir veremos as principais:

**Função *strcpy***

Sintaxe: strcpy(string de destino, string de origem);

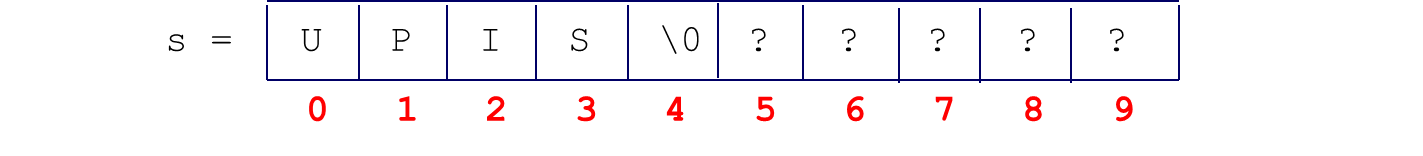
Semântica: copia a “string de origem” para a “string de destino”, acrescentando o caractere '\0'.

Exemplo:

char s[10];

strcpy(s, “UPIS”);

O conteúdo de *s* será:



**Dica**: a função *strcpy* é a forma “natural” de atribuir um valor a uma *string*, já que o comando de atribuição não funciona com este tipo de dado na linguagem C, exceto no momento da declaração da string.

**Função *strncpy***

Sintaxe: strncpy(string destino, string origem, tamanho);

Semântica: copia “tamanho” caracteres da “string origem” para a “string destino” e acrescenta '\0'.

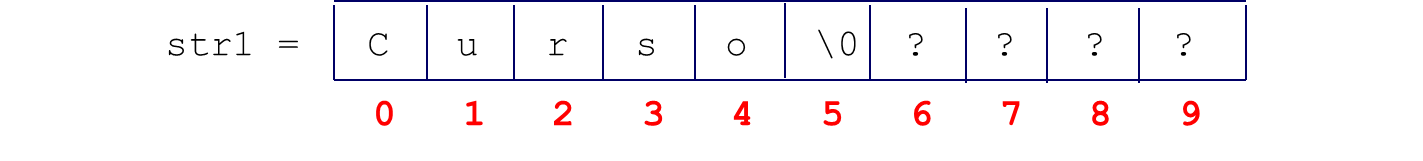
Exemplo:

char str1[10];

char str2[10] = “Curso C”;

strncpy(str1, str2, 5);

O conteúdo de ***str1*** será:



**Função *strcat***

Sintaxe: strcat(string destino, string origem);

Semântica: concatena “string origem” no final de “string destino” e acrescenta '\0'.

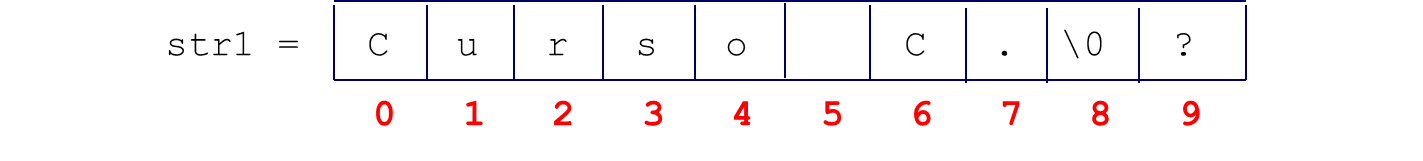
Exemplo:

char str1[10] – “Curso”;

char str2[10] = “ C.”;

strcat(str1, str2);

O conteúdo de ***str1*** será:



**Função *strncat***

Sintaxe: strncat(string destino, string origem, tamanho);

Semântica: copia “tamanho” caracteres da “string origem” para o final da “string destino” e acrescenta '\0'.

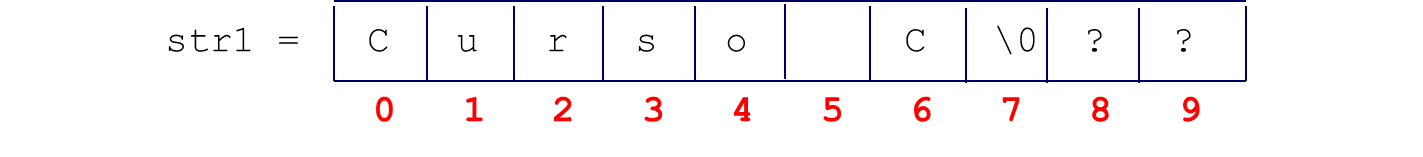
Exemplo:

char str1[10] = “Curso”;

char str2[10] = “ C.”, ;

strncat(str1, str2, 2);

O conteúdo de ***str1*** será:



**Função *strlen***

Sintaxe: strlen(string);

Semântica: retorna o tamanho dinâmico de “string”.

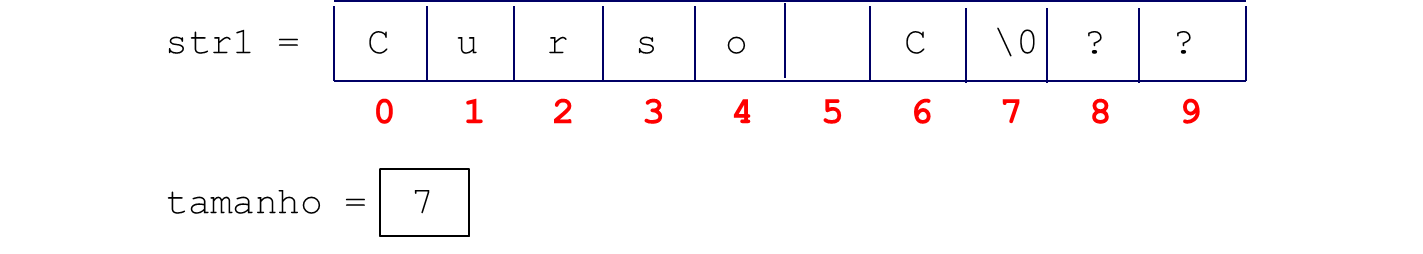
Exemplo:

char str1[10] = “Curso C”;

int tamanho;

tamanho = strlen(str1);

O conteúdo de ***str1*** e ***tamanho*** será:



**Função *strcmp***

Sintaxe: strcmp(string1, string2);

Semântica: compara o conteúdo de “string1” com “string2”. Retorna:

0 : se as duas strings são iguais;

< 0 : se “str1” for menor que “str2”;

> 0 : se “str1” for maior que “str2”.

Exemplo:

char str1[10] = “ABC”;

char str2[10] = “ABC”;

int tamanho = strcmp(str1, str2); **/\* tamanho = 0 \*/**

char str1[10] = “ABC”;

char str2[10] = “ABD”;

int tamanho = strcmp(str1, str2); **/\* tamanho = -1 \*/**

char str1[10] = “ABE”;

char str2[10] = “ABC”;

int tamanho = strcmp(str1, str2); **/\* tamanho = +2 \*/**

**Exemplo de programa manipulando *string***:

Vetor de Strings – programa que leia 4 palavras pelo teclado e armazene cada palavra em uma *string*. Depois concatene todas as *strings* lidas numa única *string* e apresente esta como resultado final.

#include<string.h>

void main(){

char palavras[4][20];

char frase[80];

int i;

for(i = 0; i < 4; i++){ **/\* lê as 4 palavras \*/**

printf("Digite uma palavra: ");

gets(palavras[i]);

}

strcpy(frase, palavras[0]); **/\* copia a 1ª palavra \*/**

for(i = 1; i < 4; i++) **/\* concatena demais palavras \*/**

strcat(frase, palavras[i]);

printf("Frase = ");

puts(frase);

for(i = 0; i < 4; i++)

printf("\n Palavra %d = %s." , i+1, palavras[i]);

}