Cláudio C. Rodrigues
Faculdade da Computação -UFU

LINGUAGEM C – 04: VETORES, STRINGS E MATRIZES

Introdução & Objetivo

- O **vetor** é provavelmente um dos mais simples e importantes ***tipos agregados***.
- Através do seu uso, podemos armazenar e manipular grandes quantidades de dados.
- Nessa seção, introduzimos o uso de ***vetores*** em **C**, mostramos como ***strings*** e ***matrizes*** são implementadas a partir deles.

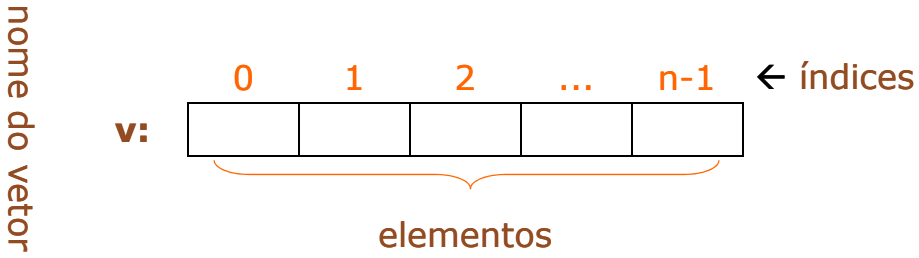
Vetores, Strings e Matrizes

- **Vetores**
  - Declaração e Inicialização de Vetores
  - Manipulação de Vetores
- **Strings**
  - Declaração e Inicialização de Strings
  - Manipulação de Strings
- **Matrizes**
  - Declaração e Inicialização de Matrizes
  - Manipulação de Matrizes.

1. Vetores

- Um **vetor** é uma coleção de variáveis de um mesmo tipo, que compartilham o mesmo nome e que ocupam posições consecutivas de memória.
- Cada uma dessas variáveis denomina-se *elemento* e é identificada por um *índice*.
- Se **v** é um vetor com **n** posições, seus elementos são v[0], v[1], v[2], ..., v[n-1].

1. Vetores



- Em **C** os vetores são sempre indexados a partir de **zero** e, portanto, o último elemento de um vetor de tamanho **n** ocupa a posição **n-1** do vetor.

1. Vetores: Declaração

- Para criarmos um vetor para armazenar 5 números inteiros declaramos a variável da seguinte maneira:
  - **int v[5];**
  - A palavra **int** indica que o vetor **v** é um grupo de variáveis inteiras e o sufixo **[5]** especifica que esse grupo possui cinco elementos.
- Em **C** os vetores são indexados a partir de *zero*, os elementos de **v** são v[0], v[1], v[2], v[3] e v[4].

Definição de Vetores (Arrays)

- O operador **sizeof()** retorna o tamanho em bytes de um tipo ou variável.
  - Exemplo:
    - int x;
    - printf("%d", sizeof(int)); // print 4
    - printf("%d", sizeof(x)); // print 4
- O operador [] associado a assinatura de um array denota um índice em um array e habilita acesso ao elemento determinado pelo índice endereçado.
  - Exemplo:
    - int idades[10]; // definição
    - idades[3] = 23; // acesso
    - printf("%d", idades[3]); // acess0
- Qual é o retorno de *sizeof(idades)*?

Notas em Arrays

- O computador (compilador) não realiza verificação de limites de arrays.
- O tamanho de um vetor (array) (número de elementos) deve ser constante.
  - A quantidade de memória necessária deve ser conhecida em tempo de compilação.
- Exemplo:
  - int i = 5;
  - int array[i]; // errado, i não é constante!
- Porém:
  - #define SIZE 5
  - int array[SIZE]; // OK, SIZE é uma constante simbólica!
- C não possui um operador específico para tratar arrays "como um todo":
  - Por exemplo: não podemos atribuir múltiplos valores simultaneamente a um array; não podemo imprimir todo o array de uma vez; etc…
  - Desafio: qual é a saída de - printf("%p", idades);

Desafio de Programação

- Crie tipos de vetores para armazenar:
  - as letras vogais do alfabeto.
  - as temperaturas diárias de uma semana.

- Codifique um programa para solicitar 5 números, via teclado, e exibí-los na ordem inversa àquela em que foram fornecidos.

1. Vetores: indexação

- Em geral, um vetor **v** pode ser indexado com qualquer *expressão* cujo valor seja de tipo inteiro.
- Essa *expressão* pode ser uma simples constante, uma variável ou então uma expressão propriamente dita, contendo operadores, constantes e variáveis.

- Exemplo:
  - v['B'−'A'] ≡ v[1]
  - Se i=3, então v[i%3] ≡ v[0], v[i−1] ≡ v[2], v[i] ≡ v[3], v[i+1] ≡ v[4].

Desafio de Programação

- Seja **w** um vetor contendo **9** elementos inteiros. Supondo que **i** seja uma variável inteira valendo **5**, que valores estarão armazenados em **w** após as atribuições a seguir?

1. w[0] = 17;
2. w[i/2] = 9;
3. w[2*i-2] = 95;
4. w[i-1] = w[8]/2;
5. w[i] = w[2];
6. w[i+ 1] = w[i]+ w[i-1];
7. w[w[2]-2] = 78;
8. w[w[i]-1] = w[1]*w[i];

P.S.: O compilador C não faz consistência da indexação de vetores, deixando essa responsabilidade a cargo do programador.

1. Vetores: Iniciação

- Vetores globais e estáticos são automaticamente **zerados** pelo compilador.
  - Mas, se for desejado, podemos iniciá-los explicitamente no momento em que os declaramos.
- Nesse caso, os valores iniciais devem ser fornecidos entre chaves e separados por vírgulas.
- Exemplo: Iniciação de vetor estático:

```
int main(void) {
  static float moeda[5] = {1.00, 0.50, 0.25, 0.10, 0.05};
  ...
}
```

Notas sobre vetores:

- Um fato interessante a respeito de vetores é que o **nome** de um vetor representa o **endereço** em que ele está alocado na memória.
- Então, se for desejado saber o endereço de um vetor **v**, em vez de escrever **&v[0]**, podemos escrever simplesmente **v**.

Exemplo:

- Endereços de vetores declarados consecutivamente:

```
#include <stdio.h>
int main(void)
{
  int x[3], y[4];
  printf("\n x =  %p e y =  %p", x, y);
}
```

- O formato %p é usado em C para a exibição de endereços.

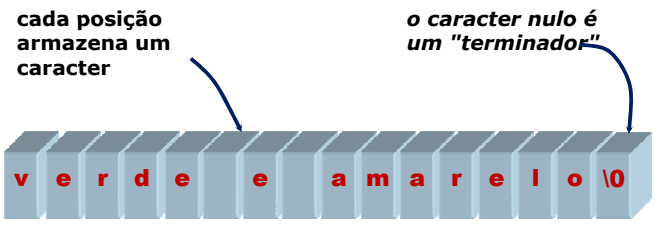
2. Strings

- A ***string*** é talvez uma das mais importantes formas de armazenamento de dados
- Em C, uma ***string*** é uma série de caracteres terminada com um *caracter nulo* , representado por '\0'.

- Como o vetor é um tipo de dados capaz de armazenar uma série de elementos do mesmo tipo e a ***string*** é uma *série de caracteres*, é bastante natural que ela possa ser representada por um vetor de caracteres.

2. Strings

- Na forma de uma constante, a string aparece como uma série de caracteres delimitada por aspas.



- ✓ Devido à necessidade do '\0', os vetores que armazenam **strings** devem ter sempre uma posição a mais do que o número de caracteres a serem armazenados

2. Strings:

- Exemplo: leitura de string via teclado.

```
#include <stdio.h>
int main(void) {
   char n[21];
   printf("Qual o seu nome? ");
   gets(n);
   printf("Olá, %s!",n);
   return 0;
}
```


2. Strings: iniciação

- Como qualquer outro vetor, ***strings*** também podem ser inicializadas quando são declaradas.
- Nessa iniciação, podemos usar a sintaxe padrão, em que os caracteres são fornecidos entre chaves e separados por vírgulas, ou então podemos usar a sintaxe própria para strings, na qual os caracteres são fornecidos entre aspas.

2. Strings: exemplo

- Problema com a iniciação padrão de strings.

```
#include <stdio.h>
int main(void) {
   char x[] = "um";              /* inclui '\0'     */
   char y[] = {'d','o','i','s'};    /* não inclui '\0' */
   printf("%s  %s", x, y);
   return 0;
}
```

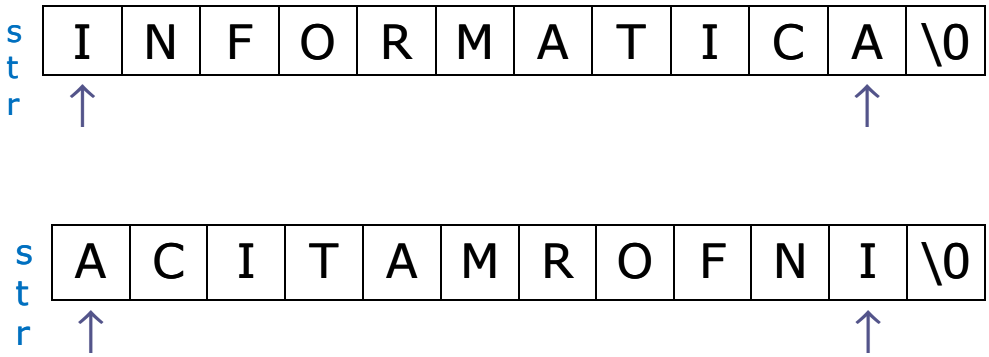

2. Strings: Questões de Uso

- Problema com o uso de operadores relacionais com strings.

```
#include <stdio.h>
int main(void)
{
   char x[] = "um";
   char y[] = "um";
   printf("%s == %s resulta em %s", x, y,
    x==y ? "verdade" : "falso");
   return 0;
}
```

Desafio de Programação

- Codifique uma função *inv_str* que inverte um string.



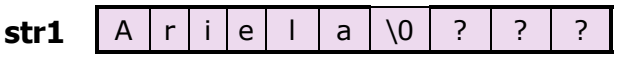
String Library

- A manipulação de String pode ser feita pelo uso da biblioteca **<string.h>**
  - #include <string.h>
- Esta biblioteca possui diversas funções que possibilitam cópia, comparação, busca, etc… em strings.
- Nota: strings enviadas para funções como argumentos devem ser inicializadas (no mínimo) com null.
- Exemplo de função:
- **int strlen(char str[])**
  - Retorna o comprimento da string str (sem incluir a terminação de string '\0').

```
int length;
char str1[12] = "Programacao";
length = strlen(str1); // length = 11
```

String Library

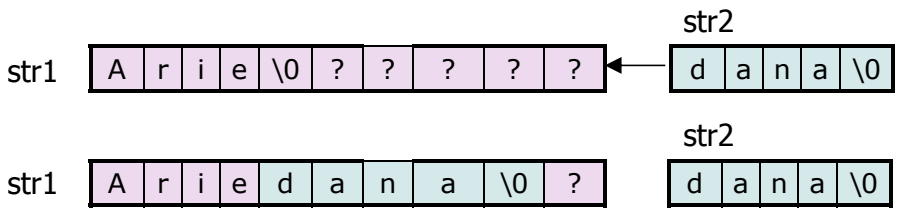
- int strcmp(char str1[], char str2[])
  - Compara duas strings (lexicograficamente);
  - Retorna:
    - 0 - as strings são iguais.
    - >0 - a primeira string é maior.
    - <0 - a segunda string é maior.
    - int compare;
    - char str1[10] = "Texto", str2[] = "texto";
    - compare = strcmp(str1, str2); // compare<0
- char *strcpy(char str1[], char str2[])
  - Copia str2 para str1 (sobrescreve).
    - (Usada no lugar de: str1=str2;).
  - Nota: str1 não precisa ser inicializada.
    - char str1[10] = "Ariela", str2[] = "dana";
    - strcpy(str1, str2);// str1 deve ser maior do que str2



String Library

- char *strcat(char str1[], char str2[])
  - Concatena str2 a str1.
  - Nota: str1 deve ser suficientemente longa.

char str1[10] = "Arie", str2[] = "dana";
strcat(str1, str2);



Array como argumento de uma função

- Quando um array é passado como argumento para uma função, somente o seu endereço é enviado.
- Há duas maneiras de definir parâmetros de tipos arrays:
  - **type[]** (exemplos: char[], int[], etc…)
  - **type *** (exemplos: char *, int *, etc…)

Array como argumento de uma função

```
void print_reverse(char *string);
int sum_array(int vector[], int size);
```

sizeof(arr) retorna o número de bytes em arr (12)

```
void main()
{
    int sum, size, arr[]={1,2,3};
    char str[]="sharon";
    size = sizeof(arr)/sizeof(int);
    sum = sum_array(arr, size);
    print_reverse(str);
}
```

mas

…

sizeof(vector) retorna 4 bytes pois somente o endereço de arr foi enviado, dado que um endereço possui quatro bytes

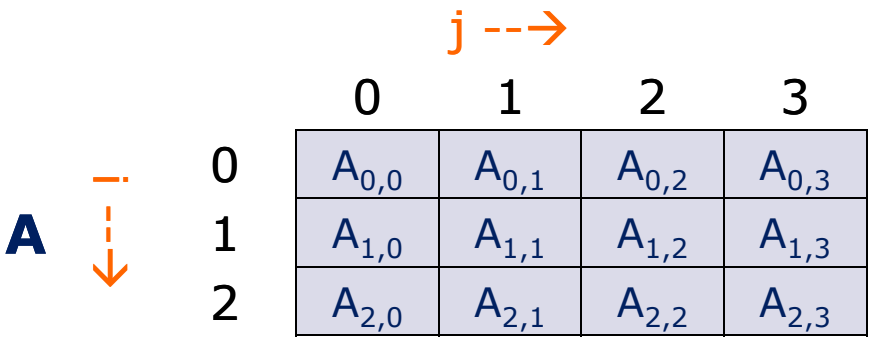
```
void print_reverse(char *string)
{
    int i, size = strlen(string);
    for(i=size-1; i>=0; i--)
        printf("%c", string[i]);
}
int sum_array(int vector[], int size)
{
    int sum = 0, i;
    for(i=0; i<size; i++)
        sum += vector[i];
    return sum;
}
```

3. Matrizes

- Uma ***matriz*** é uma coleção homogênea multidimensional, cujos elementos são distribuídos em linhas e colunas.
- Se **A** é uma matriz **mxn**, então suas linhas são indexadas de **0** a **m-1** e suas colunas de **0** a **n-1**
- Para acessar um particular elemento de **A**, escrevemos **A[i][j]**, sendo **i** o índice da linha e **j** o índice da coluna que o elemento ocupa.

3. Matrizes

- Uma matriz $\mathbf{A}_{3\times4}$ de números inteiros.
  - int **A[3][4]**;



3. Matrizes: Exemplo

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
  int i, j;
  int A[3][3] =  {{1,2,3},{4,5,6},{7,8,9}};
  printf("Matriz A:\n\n");
  for(i=0; i<3; i++)
 {
    for(j=0; j<3; j++)
      printf("A[%d][%d]=%d\t",i, j,A[i][j]);
    putchar('\n');
  }
  getch();
  return 0;
}
```


Desafio de Programação

- Escreva uma função em C que soma duas matrizes quadradas $A_{n,n} + B_{n,n} = C_{n,n}$
  - Declare as matrizes A, B e C
  - Passe estas matrizes como parâmetros para a função soma.
  - Imprima a matriz resultante C.

Dúvidas?

4. Bibliografia

- Pereira, S.L., Linguagem C – Distribuição gratuita
- Schildt, H., C Completo e Total, Editora Makron Books do Brasil Editora Ltda, 1996.
- Evaristo, J., Aprendendo a programar programando em linguagem C, Book Express, 2001.
- Mizrahi, V.V., Treinamento em Linguagem C, Curso Completo, Módulos 1 e 2, Makron Books do Brasil Editora Ltda, 1990.
- Kernighan, B.W & Ritchie, D. M., C a Linguagem de Programação, Editora Campus, 1986.