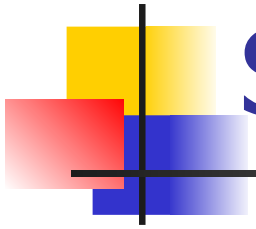
SCC0221 - Introdução à Ciência de Computação I

Prof.: Dr. Rudinei Goularte

(rudinei@icmc.usp.br)

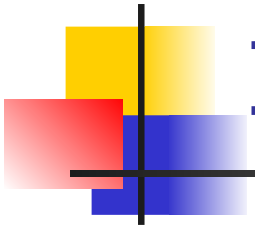
Aulas 16 e 17 - Arrays (Arranjos/Vetores)

Instituto de Ciências Matemáticas e de Computação - ICMC
Sala 4-229



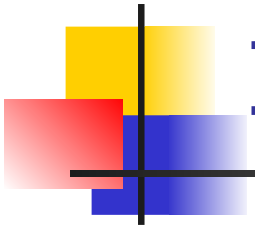
Sumário

- 1. Introdução
- 2. Arrays em C
- 3. Arrays e Memória *Stack*
- 4. Detalhe Importante



1. Introdução

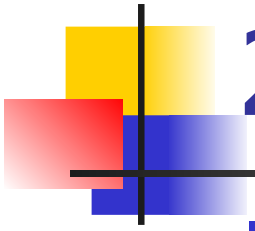
- Um *array* é um arranjo de memória, que no caso **unidimensional** é também chamado de **vetor**.
- Os nomes *array* e *vetor* serão usados indistintamente
- Pode ser entendido como uma coleção de **variáveis do mesmo tipo** referenciadas por um nome (identificador) comum.
- Uma variável específica do vetor (da coleção) é chamada de **elemento** do vetor.
- Os elementos de um vetor podem ser acessados, individualmente, por meio de índices.



1. Introdução

- Em C, os elementos de um *array* ocupam posições contíguas na memória.
- Em C, **arrays** podem ter uma, duas ou várias dimensões.

- Por ora vamos estudar o caso unidimendional.



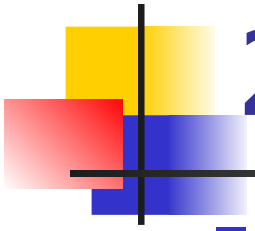
2. Arrays em C

- Forma geral da declaração - vetor:
  - **tipo** *identificador* **[***tamanho***];**
  - Exemplos

    float salario[100];

    int numeros[15];

    double distancia[43];
- Em C, todo vetor começa pelo índice 0.
  - char p[10];
  - é um vetor de caracteres que possui 10 elementos, de p[0] a p[9].



2. Arrays em C

- Em C, o i-ésimo elemento de um vetor pode ser acessado usando-se i-1 como índice através do operador [ ].
- Exemplo:

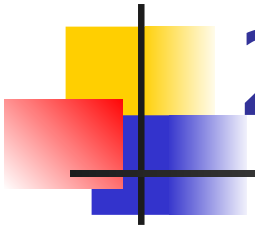
  int A[10];

  Para acessar o 1º elemento: A[0]

  A[0] = 1;

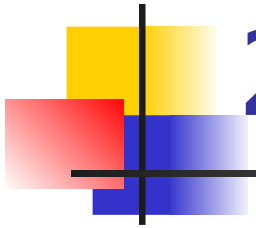
  Para acessar o 8º elemento: A[7]

  A[7] = 8;



2. Arrays em C

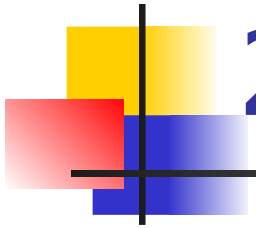
- Forma geral:
  - **A[***expressão***]**
    - A é um identificador.
    - *expressão* é qualquer expressão válida em C que retorne um **valor inteiro**.
- Exemplos: int A[10], c=8;
  - A[1.5] – não é permitido.
  - A[‘c’] – não é permitido.
  - A[(1 >= 10)] – é permitido.
  - A[c] – é permitido.
  - A[(c+2)*3] – é permitido.



2. Arrays em C

- Quantos bytes são necessários para guardar um vetor?
  - O espaço de memória necessário para um vetor está diretamente relacionado com o seu tipo.

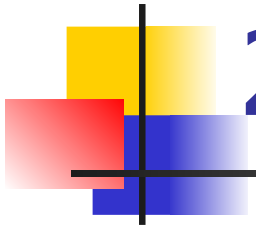
    total em bytes = **sizeof (**tipo**)** * tamanho do vetor.



2. Arrays em C

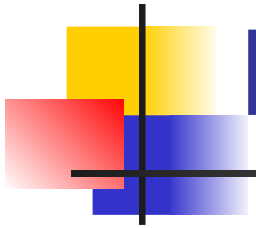
- O quê acontecerá com o trecho de código abaixo?

```
int c[10], i;
for (i = 0; i < 100; i++)
        c[i] = i;
```



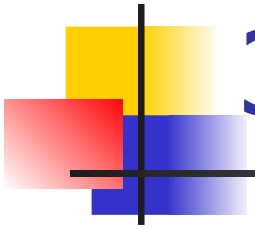
2. Arrays em C

- O código anterior compila sem erros, mas está **incorreto**.
- C não possui verificação de limites de *arrays*. É tarefa do programador fazer a verificação dos limites onde for necessário!
- No exemplo anterior ocorrerá invasão de memória – escrevendo nos dados de outra variável ou mesmo na área de código do programa.



Exercício

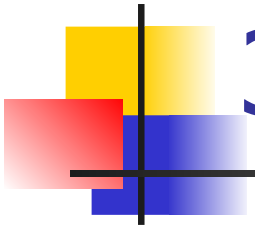
- Suponha que o vetor A contenha valores inteiros em todas as suas posições. Faça um laço, em C, para somar todos os seus elementos.



3. Arrays e Memória *Stack*

- Quando um vetor é declarado:
  - O compilador aloca, contiguamente, espaço suficiente na memória *stack* para conter todos os seus elementos.
  - O endereço base do vetor é o endereço do 1º elemento.
  - O nome do vetor sem índice retorna o endereço do 1º elemento:

```
int A[5] , B;
A[0] = 5; B = A; printf("%d", B);
```

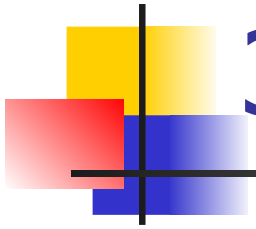


3. Arrays e Memória *Stack*

- Seja a seguinte declaração:

```
#define N 100
int A[N], i, *p, s =0;
```

- Digamos que:
  - o sistema alocou 300 como endereço base de A.
  - Inteiros possuem 4 bytes.
- Então:
  - 300, 304, 308 ..., 696 são, respectivamente, os endereços de A[0], A[1], A[2], ..., A[99].

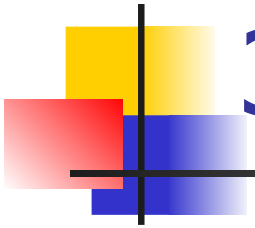


3. Arrays e Memória *Stack*

- p = A é equivalente a p = &A[0]
  - p recebe 300.
- p = A + 1 é equivalente a p = &A[1];
  - p recebe 304.

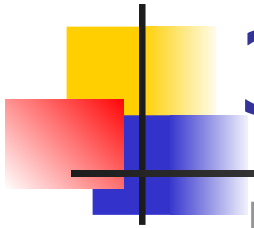
3. Arrays e Memória *Stack*

```
for (i = 0; i < N; i++)
   s += A[i];


p = A;
for (i = 0; i < N; i++)
   s += p[i];
```



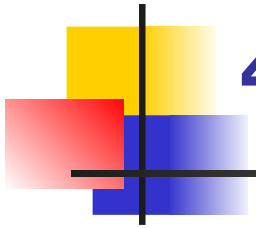
3. Arrays e Memória *Stack*

```
p = A;
for (i = 0; i < N; i++)
   s += p[i];

for (p = A; p < &A[N]; p++)
   s += *p;
```



3. Arrays e Memória *Stack*

```
p = A;
for (i = 0; i < N; i++)
   s += p[i];

for (p = A; p < &A[N]; p++)
   s += *p;

for (i = 0; i < N; i++)
   s += *(A + i);
```

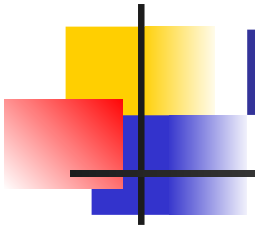


4. Detalhe Importante!

- Identificadores de vetores são ponteiros *constantes.*

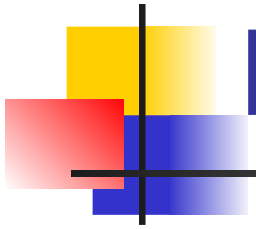
  int A[10];

  - A : ponteiro para o endereço de memória do primeiro elemento de A – **endereço base**.
  - [ ]: índice a partir do endereço base.
- Não podemos mudar o valor de A.
  - Não permitido: A = p, ++A, A += 2.



Exercício 2

- Faça um programa que preencha um vetor com 10 números inteiros, positivos, fornecidos pelo usuário. Após esse passo, o programa deve percorrer o vetor para contar a quantidade de números pares maiores que 10 presentes no vetor. Imprima a quantidade.



Exercício 3

- Desenvolva um programa que encontre o maior elemento em um vetor de números reais, positivos, de tamanho n. n e os elementos devem ser fornecidos pelo usuário em uma etapa anterior à procura pelo maior elemento. Imprima o maior elemento.


Fim.