

Subprogramação

Programando com funções e procedimentos

**Passando vetores e matrizes como
argumentos para funções**

Ponteiros e Arranjos

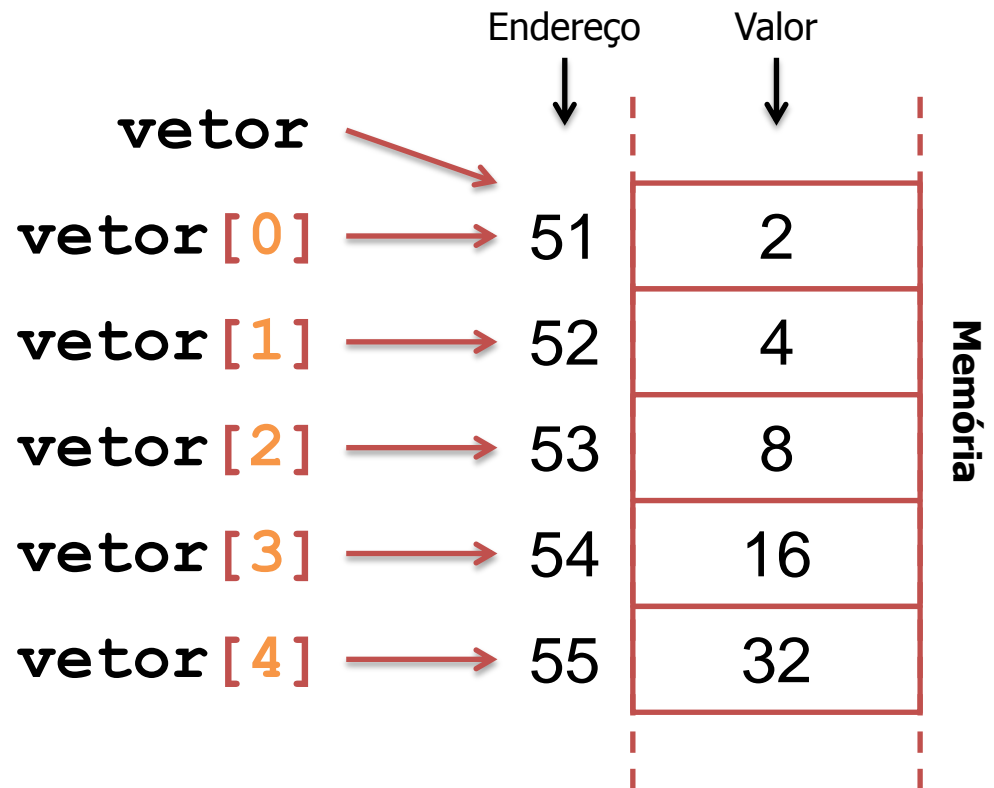
- Utilizamos ponteiros intrinsecamente para manipulação de arranjos
- Considerando um vetor de inteiros de 5 posições:
`int vetor[5];`
- O nome da variável **vetor** é um ponteiro que aponta para a primeira posição do vetor (índice zero)

```
//imprime o endereço de memória do vetor  
printf("%p", vetor);
```

Vetor na memória

```
int vetor[5] = {2, 4, 8, 16, 32};
```

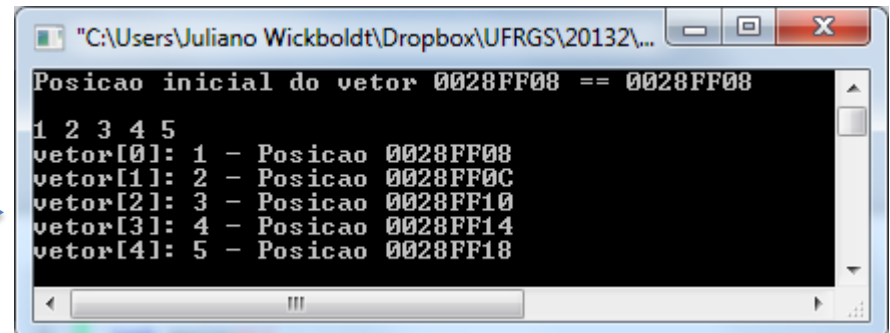
A variável **vetor** contém o valor da posição de memória da primeira posição do arranjo, ou seja, ela aponta para o início do vetor



Exemplo de impressão dos endereços de cada posição de um vetor

```
1  #include <stdio.h>
2
3  int main(){
4      int vetor[5], i;
5
6      printf("Posicao inicial do vetor %p == %p\n\n", vetor, &vetor[0]);
7
8      for(i=0;i<5;i++){
9          scanf("%d", &vetor[i]);
10     }
11     for(i=0;i<5;i++){
12         printf("vetor[%d]: %d - Posicao %p\n", i, vetor[i], &vetor[i]);
13     }
14 }
```

Os endereços não são contíguos
porque cada int ocupa 4 bytes
na memória



```
"C:\Users\Juliano Wickboldt\Dropbox\UFRGS\20132\..."
Posicao inicial do vetor 0028FF08 == 0028FF08

1 2 3 4 5
vetor[0]: 1 - Posicao 0028FF08
vetor[1]: 2 - Posicao 0028FF0C
vetor[2]: 3 - Posicao 0028FF10
vetor[3]: 4 - Posicao 0028FF14
vetor[4]: 5 - Posicao 0028FF18
```

Matrizes

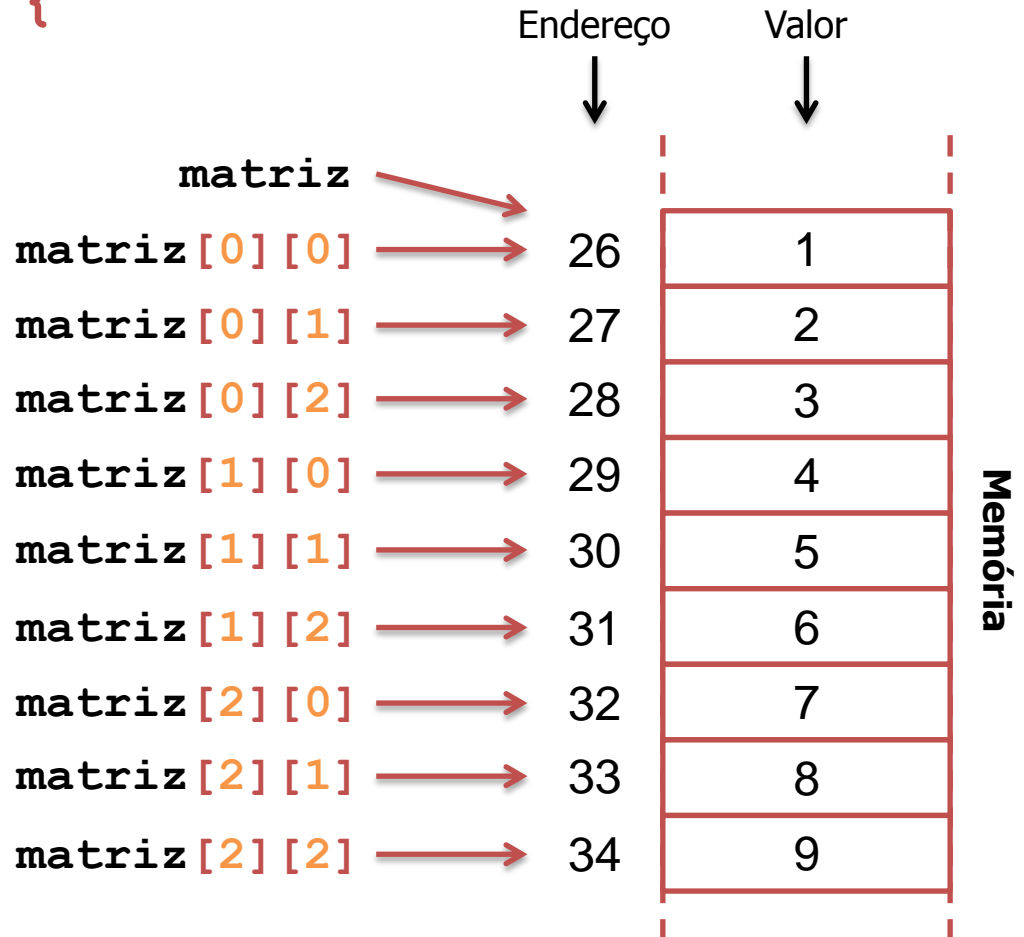
- Assim como vetores, matrizes também utilizam ponteiros para controlar o acesso aos seus elementos
- Considerando uma matriz de inteiros 3x3:
`int matriz[3][3];`
- A cada índice da primeira dimensão da matriz é um ponteiro que aponta para o início de uma linha específica

```
//imprime o endereço de memória do início  
//da segunda linha da matriz  
printf("%p", matriz[1]);
```

Matriz na memória

```
int matriz[3][3] = {  
    {1, 2, 3},  
    {4, 5, 6},  
    {7, 8, 9},  
};
```

Matrizes são
armazenadas
linearmente na
memória



```

1  #include <stdio.h>
2
3  int main(){
4      int matriz[3][3], i, j;
5
6      printf("Posicao inicial da matriz %p == %p\n\n", matriz, &matriz[0]);
7      printf("Posicao linha 0 da matriz %p == %p\n\n", matriz[0], &matriz[0][0]);
8      printf("Posicao linha 1 da matriz %p == %p\n\n", matriz[1], &matriz[1][0]);
9      printf("Posicao linha 2 da matriz %p == %p\n\n", matriz[2], &matriz[2][0]);
10
11     for(i=0;i<3;i++)
12         for(j=0;j<3;j++)
13             scanf("%d", &matriz[i][j]);
14
15     for(i=0;i<3;i++){
16         printf("--- Linha %d\n", i);
17         for(j=0;j<3;j++){
18             printf("matriz[%d][%d]: %d - Posicao %p\n",
19                 i, j, matriz[i][j], &matriz[i][j]);
20         }
21     }
22 }

```

Exemplo de impressão dos endereços de cada posição de uma matriz

Os endereços não são contíguos porque cada int ocupa 4 bytes na memória



```

"C:\Users\Juliano Wickboldt\Dropbox\UFRGS\20132\Se...
Posicao inicial da matriz 0028FEF4 == 0028FEF4
Posicao linha 0 da matriz 0028FEF4 == 0028FEF4
Posicao linha 1 da matriz 0028FF00 == 0028FF00
Posicao linha 2 da matriz 0028FF0C == 0028FF0C
1 2 3 4 5 6 7 8 9
--- Linha 0
matriz[0][0]: 1 - Posicao 0028FEF4
matriz[0][1]: 2 - Posicao 0028FEF8
matriz[0][2]: 3 - Posicao 0028FEFC
--- Linha 1
matriz[1][0]: 4 - Posicao 0028FF00
matriz[1][1]: 5 - Posicao 0028FF04
matriz[1][2]: 6 - Posicao 0028FF08
--- Linha 2
matriz[2][0]: 7 - Posicao 0028FF0C
matriz[2][1]: 8 - Posicao 0028FF10
matriz[2][2]: 9 - Posicao 0028FF14

```

Passando arranjos como argumento de funções

- Vimos que sempre que definimos um arranjo na verdade estamos definindo também ponteiros
- Sendo assim, ao passar um arranjo para uma função passamos sempre uma **referência** para o início do arranjo
- Não é possível passar arranjos para funções por cópia de valores em C

Passando um vetor para uma função

- Forma 1:

- Declaração:

```
void imprime(int * v, int tam){  
    int i;  
    for (i=0; i<tam; i++){  
        printf("%d ", v[i]);  
    }  
    printf("\n");  
}
```

A função recebe um ponteiro para um inteiro, no caso será um vetor, e um valor que representa o tamanho do vetor

Dentro da função acessamos as posições do vetor normalmente através dos índices

- Chamada:

```
int v[MAX];  
imprime(v, MAX);
```

Na chamada passamos simplesmente o ponteiro (v) sem necessidade do & para indicar a referência

Passando um vetor para uma função

- Forma 2:

- Declaração:

```
void imprime(int v[MAX], int tam){  
    int i;  
    for (i=0; i<tam; i++){  
        printf("%d ", v[i]);  
    }  
    printf("\n");  
}
```

Aqui a função recebe um vetor, mas o efeito prático é o mesmo. Continuamos passando uma referência para o vetor declarado no main

Dentro da função acessamos as posições do vetor novamente através dos índices

- Chamada:

```
int v[MAX];  
imprime(v, MAX);
```

A chamada permanece exatamente igual

Passando uma matriz para uma função

- Forma simplificada:

- Declaração:

```
void imprime(int m[LIN][COL], int lin, int col){  
    int i, j;  
    for (i=0; i<lin; i++){  
        for (j=0; j<col; j++){  
            printf("%d ", m[i][j]);  
        }  
        printf("\n");  
    }  
}
```

A função recebe a especificação da matriz com suas linhas e colunas

Dentro da função acessamos as posições da matriz normalmente

Na chamada passamos simplesmente o ponteiro (m) sem necessidade do & para indicar a referência

- Chamada:

```
int m[LIN][COL];  
imprime(m, LIN, COL);
```

É possível omitir uma ou até as duas dimensões passando apenas os ponteiros. No entanto, o acesso as linhas e colunas da matriz deve ser feito através de aritmética de ponteiros, o que complica bastante.