



# Estrutura de Dados

Ricardo José Cabeça de Souza

[www.ricardojcsouza.com.br](http://www.ricardojcsouza.com.br)

[ricardo.souza@ifpa.edu.br](mailto:ricardo.souza@ifpa.edu.br)

Parte 5



# VETORES

- **VETORES**

- Forma de estruturar um conjunto de dados

```
int v[10];
```

- Reserva um espaço de memória **contínuo** para armazenar 10 valores inteiros

- Cada **int** ocupa 4 bytes, a declaração acima reserva um espaço de memória de 40 bytes



# VETORES

- **VETORES**

- Acesso aos dados
- A indexação de um vetor varia de **zero** a  **$n-1$** , onde  **$n$**  representa a dimensão do vetor

```
v[0]    →   acessa o primeiro elemento de v  
v[1]    →   acessa o segundo elemento de v  
...  
v[9]    →   acessa o último elemento de v
```

```
v[10]   →   está ERRADO (invasão de memória)
```



# VETORES

- **VETORES**  
– Exemplo

```
/* Cálculo da media e da variância de 10 números reais */
#include <stdio.h>

int main ( void )
{
    float v[10];          /* declara vetor com 10 elementos */
    float med, var;      /* variáveis para armazenar a média e a variância */
    int i;                /* variável usada como índice do vetor */

    /* leitura dos valores */
    for (i = 0; i < 10; i++)          /* faz índice variar de 0 a 9 */
        scanf("%f", &v[i]);         /* lê cada elemento do vetor */

    /* cálculo da média */
    med = 0.0;                        /* inicializa média com zero */
    for (i = 0; i < 10; i++)
        med = med + v[i];             /* acumula soma dos elementos */
    med = med / 10;                   /* calcula a média */

    /* cálculo da variância */
    var = 0.0;                        /* inicializa variância com zero */
    for ( i = 0; i < 10; i++ )
        var = var+(v[i]-med)*(v[i]-med); /* acumula quadrado da diferença */
    var = var / 10;                   /* calcula a variância */

    printf ( "Media = %f   Variancia = %f  \n", med, var );
    return 0;
}
```



# VETORES

- **VETORES**

- Os vetores também podem ser inicializados na declaração

```
int v[5] = { 5, 10, 15, 20, 25 };
```

ou simplesmente:

```
int v[] = { 5, 10, 15, 20, 25 };
```



# VETORES

- **PASSAGEM DE VETORES PARA FUNÇÕES**
  - Consiste em passar o endereço da primeira posição do vetor
  - A função chamada deve ter um parâmetro do tipo ponteiro para armazenar este valor
  - **FORMATOS:**
    - int \*V
    - int V[]
    - int V[TAM]



# VETORES

- **ALOCAÇÃO DINÂMICA**
  - Na declaração de um vetor, foi preciso dimensioná-lo
  - Pré-dimensionamento do vetor é um fator limitante
  - Alocação dinâmica: fazer a alocação do vetor dinamicamente, sem desperdício de memória

# VETORES



- **USO DA MEMÓRIA**

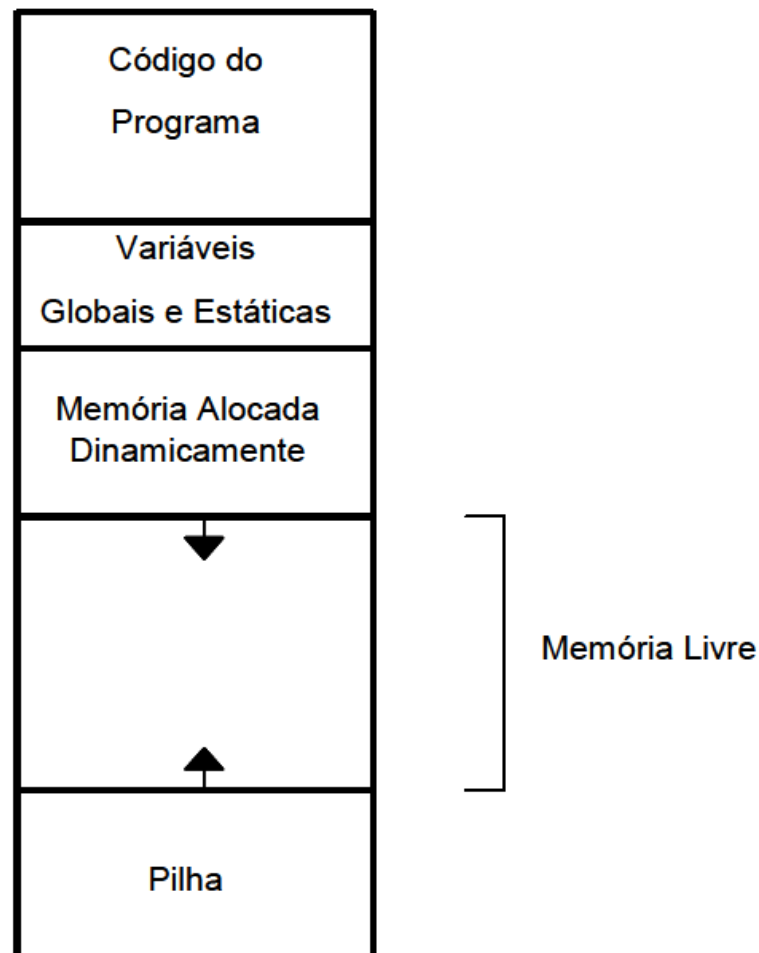
- Existem três maneiras de reservarmos espaço de memória para o armazenamento de informações:
  - uso de variáveis globais (e estáticas)
  - uso de variáveis locais
  - requisitando ao sistema, em tempo de execução, um espaço de um determinado tamanho





# VETORES

- **USO DA MEMÓRIA**





# VETORES

- **FUNÇÕES DA BIBLIOTECA PADRÃO**

- Existem funções, presentes na biblioteca padrão *stdlib*, que permitem alocar e liberar memória dinamicamente
- A função básica para alocar memória é **malloc**
- Ela recebe como parâmetro o número de bytes que se deseja alocar e retorna o endereço inicial da área de memória alocada

```
int *v;
```

```
v = malloc(10*4);    v = malloc(10*sizeof(int));
```

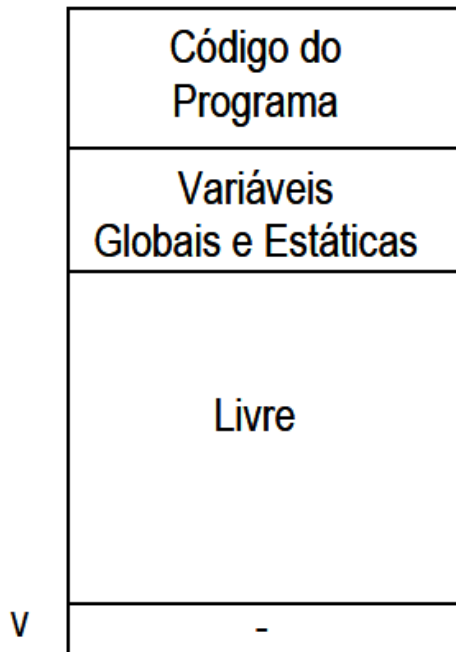


# VETORES

## • FUNÇÕES DA BIBLIOTECA PADRÃO

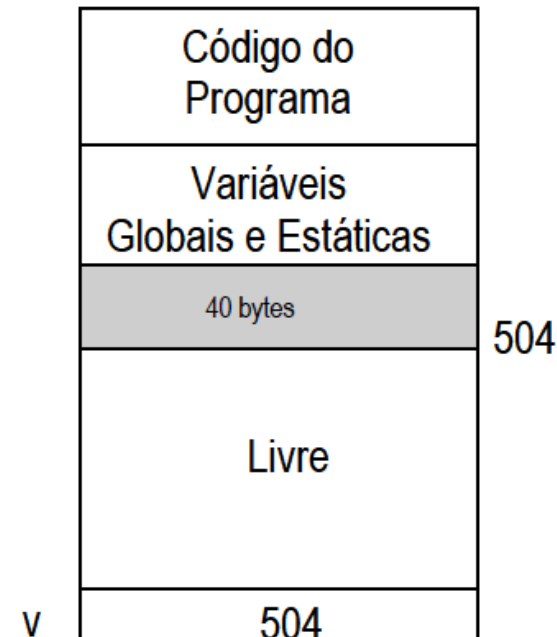
1 - Declaração: `int *v`

Abre-se espaço na pilha para o ponteiro (variável local)



2 - Comando: `v = (int *) malloc (10*sizeof(int))`

Reserva espaço de memória da área livre e atribui endereço à variável





# VETORES

- **FUNÇÕES DA BIBLIOTECA PADRÃO**
  - Se, porventura, não houver espaço livre suficiente para realizar a alocação, a função retorna um endereço nulo (representado pelo símbolo **NULL**, definido em *stdlib.h*)
  - Podemos cercar o erro na alocação do programa verificando o valor de retorno da função **malloc**

```
...
v = (int*) malloc(10*sizeof(int));
if (v==NULL)
{
    printf("Memoria insuficiente.\n");
    exit(1); /* aborta o programa e retorna 1 para o sist. operacional */
}
...
```



# VETORES

- **LIBERANDO ESPAÇO NA MEMÓRIA**
  - Para liberar um espaço de memória alocado dinamicamente, usamos a função **free**
  - Recebe como parâmetro o ponteiro da memória a ser liberada

```
free (v) ;
```



# VETORES

- **LIBERANDO ESPAÇO NA MEMÓRIA**

```
/* Cálculo da média e da variância de n reais */

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

...

int main ( void )
{
    int i, n;
    float *v;
    float med, var;

    /* leitura do número de valores */
    scanf("%d", &n);
    /* alocação dinâmica */
    v = (float*) malloc(n*sizeof(float));
    if (v==NULL)    {
        printf("Memoria insuficiente.\n");
        return 1;
    }
    /* leitura dos valores */
    for (i = 0; i < n; i++)
        scanf("%f", &v[i]);
    med = media(n,v);
    var = variancia(n,v,med);
    printf("Media = %f    Variancia = %f  \n", med, var);
    /* libera memória */
    free(v);
    return 0;
}
```

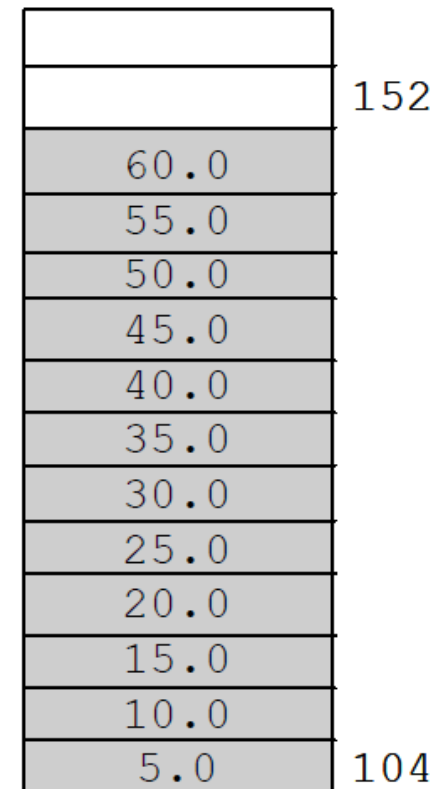
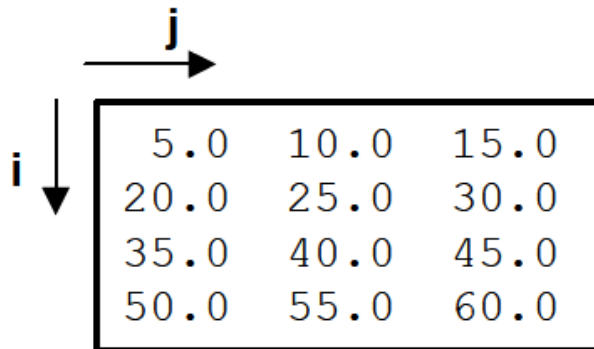


# MATRIZES

- **MATRIZ**

– Vetores bidimensionais, declarados estaticamente

```
float m[4][3] = {{ 5.0, 10.0, 15.0},  
                 {20.0, 25.0, 30.0},  
                 {35.0, 40.0, 45.0},  
                 {50.0, 55.0, 60.0}};
```





# MATRIZES

- **MATRIZ**

- Os elementos da matriz são acessados com indexação dupla: `mat[i][j]`
- O primeiro índice, **i**, acessa a linha e o segundo, **j**, acessa a coluna
- As matrizes também podem ser inicializadas na declaração

```
float mat[4][3] = {{1,2,3},{4,5,6},{7,8,9},{10,11,12}};
```

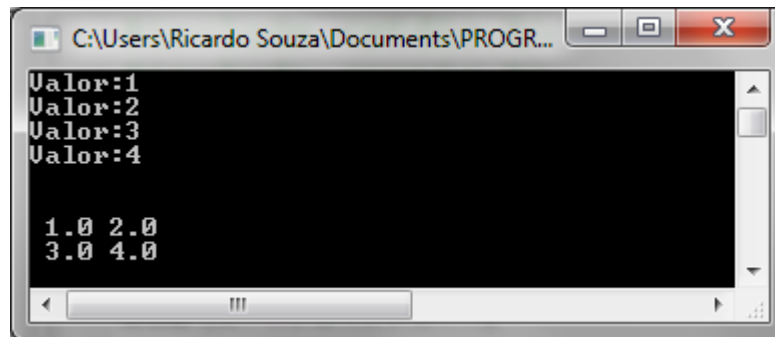




# MATRIZES

## • PASSAGEM DE MATRIZES PARA FUNÇÕES

- float x[][TAM]
- float x[tam][tam]



```
C:\Users\Ricardo Souza\Documents\PROGR...  
Valor:1  
Valor:2  
Valor:3  
Valor:4  
  
1.0 2.0  
3.0 4.0
```

```
#include <stdio.h>  
#include <conio.h>  
#define T 2  
void LER (float X[][T])  
{  
int i,j;  
for(i=0;i<T;i++)  
{  
for(j=0;j<T;j++)  
{  
printf("Valor:");  
scanf("%f",&X[i][j]);  
} //Fim j  
} //Fim i  
} //Fim LER  
main()  
{  
float MAT[T][T];  
int i,j;  
LER(MAT);  
//Exibição da Matriz  
printf("\n\n");  
for(i=0;i<T;i++)  
{  
for(j=0;j<T;j++)  
{  
printf(" %.1f",MAT[i][j]);  
} //Fim j  
printf("\n");  
} //Fim i  
getch();  
}
```

# Estrutura de Dados



- **REFERÊNCIAS**
- Tenenbaum, Aaron M. Langsam, Yedidyah, Augenstein, Moshe J. **Estruturas de dados usando C**. São Paulo : MAKRON *Books*, 1995.
- Veloso, Paulo. et. al. **Estrutura de dados**. Rio de Janeiro: Campus, 1986.
- Moraes, Celso Roberto. **Estrutura de dados e algoritmos**. 2. ed. São Paulo: Futura, 2003.
- Celes, W. Rangel, J. L. **Curso de Estrutura de Dados**. PUC-Rio, 2002.
- W. Celes, R. Cerqueira, J.L. Rangel. **Introdução a Estruturas de Dados - com técnicas de programação em C**. Rio de Janeiro: Campus, 2004.