



ESTRUTURAS DE DADOS

Revisão de Ponteiros, Vetores e Matrizes

Variáveis do Tipo Ponteiro

- A linguagem C permite o armazenamento e a manipulação de valores de endereços de memória
- Para cada tipo existente, existe um tipo ponteiro correspondente

int a	int *a
float a	float *a
char c	char *c

Variáveis do Tipo Ponteiro

- O operador & (endereço de) aplicado a variáveis retorna o endereço da posição de memória reservada a variável
- O operador * (conteúdo de) aplicado a variáveis do tipo ponteiro, retorna o conteúdo do endereço de memória armazenado pela variável ponteiro

Variáveis do Tipo Ponteiro

```
int x;  
int *px;  
int t;
```

t		108
px		104
x		100

px = &x

t		108
px	100	104
x	8	100

```
x = 8;
```

t		108
px		104
x	8	100

t = *px;

t	8	108
px	100	104
x	8	100

Exemplo de Função

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

```
void somaprod(int x, int y, int *s, int *p) {  
    *s = x+y;  
    *p = x*y;  
}
```

```
int main (void) {  
    int s, p;  
    somaprod(3,5,&s,&p);  
    printf("'soma = %d, prod = %d'",s,p);  
    return 0;  
}
```

Exemplo de Função

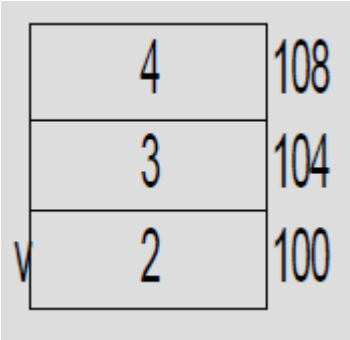
```
#include<stdio.h>
```

```
void troca(int *px, int *py) {  
    int temp = *px;  
    *px = *py;  
    *py = temp;  
}
```

```
int main (void) {  
    int a=5, b =7;  
    troca(&a,&b);  
    printf(''a=%d, b=%d'',a,b);  
    return 0;  
}
```

Vetor

- Estrutura de dados definindo um conjunto enumerável
- `int v[] = {2, 3, 4};`
- `int v[3] = {2, 3, 4};`
- `v[0] = 2` ou `*v = 2`
- `v[1] = 3` ou `*(v+1) = 3`
- `v[2] = 4` ou `*(v+2) = 4`
- De forma geral,
 - `v[i]` equivale a `*(v+i)`
 - `&v[i]` equivale a `(v+i)`



4	108
3	104
2	100

v

Somatório de um Vetor

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

```
float somatorio(int m, float v[]){  
    int i; float s =0.0F;  
    for(i=0;i<m;i++)  
        s += v[i];  
    return s;  
}
```

```
int main (void) {  
    float v[] = {2.0, 3.0, 4.0};  
    float s = somatorio(3,v);  
    printf("'somatorio=%.1f,media=%.1f'",s,s/3);  
    return 0;  
}
```


Alocação Dinâmica

- Quando o número de elementos não é conhecido em tempo de compilação
- O operador **sizeof** retorna o número de bytes de um tipo
- A função malloc recebe como parâmetro o número de bytes a serem alocados
- `int *v = (int*)malloc(3*sizeof(int));`
- `v[0] = 2;`
- `v[1] = 3;`
- `v[2] = 4;`

v	4	108
	3	104
	2	100

Somatório de um Vetor

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

float somatorio(int m, float v[]){ ... }

int main (void) {
    int n; int i; float *v; float s;
    scanf("%d",&n);
    v = (float*) malloc(n*sizeof(float));
    for(i=0; i<n; i++)
        scanf("%f",&v[i]);
    s = somatorio(n,v);
    printf("somatorio=%.1f,media=%.1f",s,s/n);
    return 0;
}
```

Alocação Dinâmica

- A função malloc pode retornar NULL, caso não tenha espaço suficiente

```
if (v==NULL) {  
    printf("'Memoria Insuficiente');  
    exit(1);  
}
```

- O espaço alocado deve ser liberado

```
free(v);
```

Matrizes

- Vetor Bidimensional (ou matriz)

```
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}};
```

- $m[0][0] = 5.0$
- $m[2][1] = 40.0$

5,0	10,0	15,0
20,0	25,0	30,0
35,0	40,0	45,0
50,0	55,0	60,0

	60,0	144
	55,0	140
	50,0	136
	45,0	132
	40,0	128
	35,0	124
	30,0	120
	25,0	116
	20,0	112
	15,0	108
	10,0	104
m	5,0	100

Somatório de uma Matriz

```
float somatorio(int m, float v[][3]) {  
    int i, j; float md = 0.0F;  
    for (i=0; i<m; i++)  
        for (j=0; j<3; j++)  
            md += v[i][j];  
    return md;  
}
```

Alocação Dinâmica de Matrizes

- Matriz Representada como um Vetor
- `float *mat = (float*)malloc(m*n*sizeof(float));`
 - `mat[i][j] → mat[i*n+j]`
 - `mat[2][1] → mat[2*3+1]=40`

5,0	10,0	15,0
20,0	25,0	30,0
35,0	40,0	45,0
50,0	55,0	60,0

60,0	144
55,0	140
50,0	136
45,0	132
40,0	128
35,0	124
30,0	120
25,0	116
20,0	112
15,0	108
10,0	104
5,0	100

m

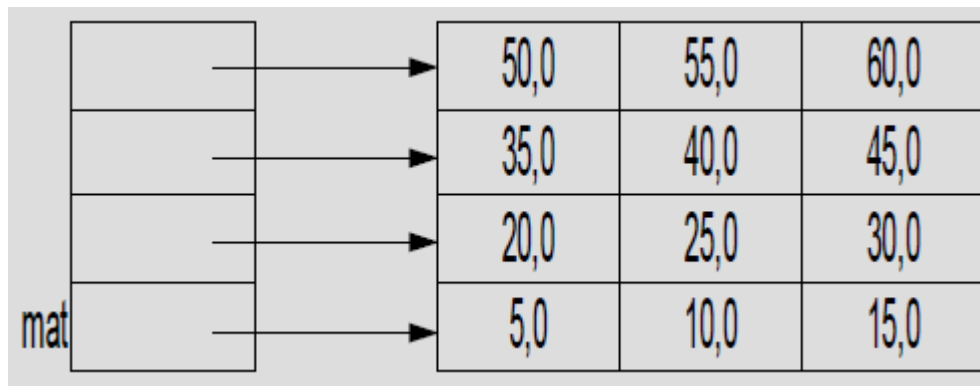
Somatório de uma Matriz

```
float somatorio(int m, int n, float *v) {  
    int i, j; float md = 0.0F;  
    for(i=0; i<m; i++)  
        for(j=0; j<n; j++)  
            md += v[i*n+j];  
    return md;  
}
```

Alocação Dinâmica de Matrizes

- Matriz Representada como um vetor de ponteiros

```
float **mat = (float**)malloc(m*sizeof(float*));  
for(i=0; i<m; i++)  
    mat[i] = (float*)malloc(n*sizeof(float));
```



Somatório de uma Matriz

```
float somatorio(int m, int n, float **v) {  
    int i, j; float md = 0.0F;  
    for(i=0; i<m; i++)  
        for(j=0; j<n; j++)  
            md += v[i][j];  
    return md;  
}
```

Alocação Dinâmica de Matrizes

- Para Alocar Memória

```
float **mat =  
    (float**) malloc (m*sizeof(float*));  
for(i=0; i<m; i++)  
    mat[i] = (float*) malloc (n*sizeof(float));
```

- Para Liberar a Memória

```
for(i=0; i<m; i++)  
    free(mat[i]);  
free(mat);
```

- Slides baseados no livro **Introdução a Estruturas de Dados**, Waldemar Celes, Renato Cerqueira e José Lucas Rangel, Editora Campus, 2004.