

# Estruturas de Dados

## Módulo 5 – Vetores e Alocação Dinâmica



# Referências

Waldemar Celes, Renato Cerqueira, José Lucas Rangel,  
*Introdução a Estruturas de Dados*, Editora Campus  
(2004)

Capítulo 5 – Vetores e alocação dinâmica

# Tópicos

- Vetores
- Alocação dinâmica
- Vetores locais e funções

# Vetores

- Vetor:
  - estrutura de dados definindo um conjunto enumerável
  - Exemplo:
    - v = vetor de inteiros com 10 elementos

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

# Vetores

- Vetor (cont.):
  - vetor pode ser inicializado
  - Exemplo:
    - v = vetor de inteiros com 10 elementos

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

ou simplesmente:

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

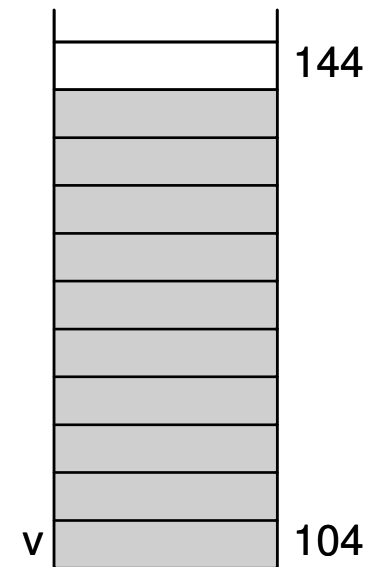
# Vetores

- Vetor (cont.):
  - acesso a cada elemento é feito através de indexação da variável
  - Exemplo:
    - v = vetor de inteiros com 10 elementos

```
v[0] = 0;          /* acessa o primeiro elemento de v    */
...
v[9] = 9;          /* acessa o último elemento de v      */
v[10] = 10;        /* ERRADO (invasão de memória)      */
```

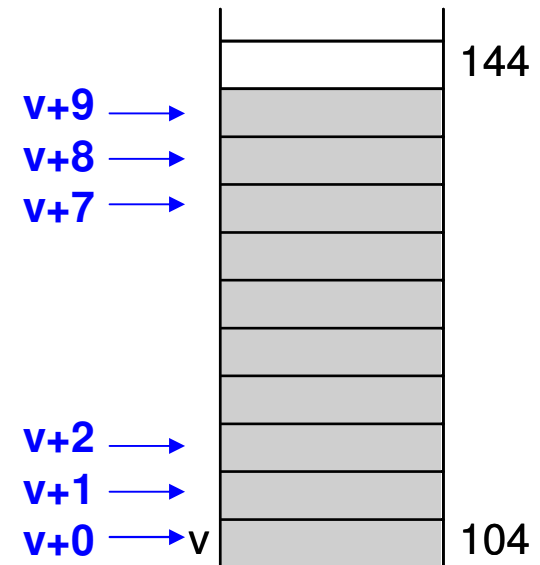
# Vetores

- Vetor (cont):
  - vetor é alocado em posições contíguas de memória
  - Exemplo:
    - $v$  = vetor de inteiros com 10 elementos
    - espaço de memória de  $v$  =  
10 x valores inteiros de 4 bytes =  
40 bytes



# Vetores

- Vetores e ponteiros:
  - nome do vetor aponta para endereço inicial
  - C permite aritmética de ponteiros
  - Exemplo:
    - $v+0$  : primeiro elemento de  $v$
    - ...
    - $v+9$  : último elemento de  $v$
  - Exemplo:
    - $\&v[i]$  é equivalente a  $(v+i)$
    - $*(v+i)$  é equivalente a  $v[i]$





# Vetores

- Exemplo:
  - cálculo da média e da variância de um conjunto de 10 números reais

$$m = \frac{\sum x}{N}, \quad v = \frac{\sum (x - m)^2}{N}$$


- implementação
  - valores são lidos e armazenados em um vetor de 10 posições
  - cálculos da média e da variância efetuados sobre o conjunto de valores armazenado

```
/* Cálculo da média 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 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 (digitados na mesma linha) */
```

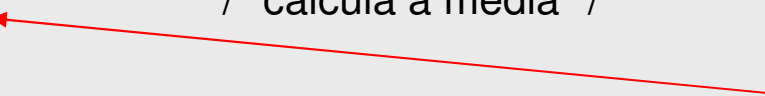


endereço da i-ésima  
posição de v

```
/* 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 com zero */
for ( i = 0; i < 10; i++ )
    var = var+(v[i]-med)*(v[i]-med);      /* acumula */
var = var / 10;                          /* calcula a variância */

/* exibição do resultado */
printf ( "Media = %f  Variancia = %f \n", med, var );
return 0;
}
```



comando não pertence  
ao corpo do "for"

# Vetores

- Passagem de vetor para função:
  - consiste em passar o endereço da primeira posição do vetor
  - função deve ter parâmetro do tipo ponteiro para armazenar valor
    - “passar um vetor para uma função” é equivalente a “passar o endereço inicial do vetor”
    - elementos do vetor não são copiados para a função
    - argumento copiado é apenas o endereço do primeiro elemento
  - Exemplo:
    - chamada a função passando vetor de `int`
    - função deve ter um parâmetro do tipo `int`

```
/* Cálculo da média e da variância de 10 reais (segunda versão) */
```

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

```
/* Função para cálculo da média */
```

```
float media (int n, float* v)
```

```
{
```

```
    int i;
```


```
    float s = 0.0;
```

```
    for (i = 0; i < n; i++)
```

```
        s += v[i];
```

```
    return s/n;
```

```
}
```



parâmetro do tipo  
ponteiro para float

```
/* Função para cálculo da variância */  
float variancia (int n, float* v, float m)  
{  
    int i;  
    float s = 0.0;  
    for (i = 0; i < n; i++)  
        s += (v[i] - m) * (v[i] - m);  
    return s/n;  
}
```

```
int main ( void )
{
    float v[10];
    float med, var;
    int i;

    /* leitura dos valores */
    for ( i = 0; i < 10; i++ )
        scanf("%f", &v[i]);

    med = media(10,v);
    var = variancia(10,v,med);

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

# Vetores

- Passagem de vetor para função (cont.):
  - função pode alterar os valores dos elementos do vetor pois recebe o endereço do primeiro elemento do vetor (e não os elementos propriamente ditos)
  - Exemplo:
    - função incrementando todos os elementos de uma unidade



```
/* Incrementa elementos de um vetor */
#include <stdio.h>
void incr_vetor ( int n, int *v )
{
    int i;
    for (i = 0; i < n; i++)
        v[i]++;
}

int main ( void )
{
    int a[ ] = {1, 3, 5};
    incr_vetor(3, a);
    printf("%d %d %d \n", a[0], a[1], a[2]);
    return 0;
}
```

saída do programa será 2 4 6

# Alocação Dinâmica

- Uso da memória:
  - uso de variáveis globais (e estáticas):
    - espaço reservado para uma variável global existe enquanto o programa estiver sendo executado
  - uso de variáveis locais:
    - espaço existe apenas enquanto a função que declarou a variável está sendo executada
    - liberado para outros usos quando a execução da função termina
  - variáveis globais ou locais podem ser simples ou vetores:
    - para vetor, é necessário informar o número máximo de elementos pois o compilador precisa calcular o espaço a ser reservado

# Alocação Dinâmica

- Uso da memória:
  - alocação dinâmica:
    - espaço de memória é requisitada em tempo de execução
    - espaço permanece reservado até que seja explicitamente liberado
      - depois de liberado, espaço estará disponibilizado para outros usos e não pode mais ser acessado
      - espaço alocado e não liberado explicitamente, será automaticamente liberado quando ao final da execução

# Alocação Dinâmica

- Uso da memória:
  - **memória estática:**
    - código do programa
    - variáveis globais
    - variáveis estáticas
  - **memória dinâmica:**
    - variáveis alocadas dinamicamente
    - **memória livre**
    - variáveis locais

memória estática	Código do programa
	Variáveis globais e Variáveis estáticas
memória dinâmica	Variáveis alocadas dinamicamente
	<b>Memória livre</b>
	Variáveis locais (Pilha de execução)

# Alocação Dinâmica

- Uso da memória:
  - alocação dinâmica de memória:
    - usa a memória livre
    - se o espaço de memória livre for menor que o espaço requisitado, a alocação não é feita e o programa pode prever tratamento de erro
  - pilha de execução:
    - utilizada para alocar memória quando ocorre chamada de função:
      - sistema reserva o espaço para as variáveis locais da função
      - quando a função termina, espaço é liberado (desempilhado)
    - se a pilha tentar crescer mais do que o espaço disponível existente, programa é abortado com erro

memória estática	Código do programa
	Variáveis globais e Variáveis estáticas
memória dinâmica	Variáveis alocadas dinamicamente
	Memória livre
	Variáveis locais (Pilha de execução)

# Alocação Dinâmica

- Funções da biblioteca padrão “stdlib.h”
  - contém uma série de funções pré-definidas:
    - funções para tratar alocação dinâmica de memória
    - constantes pré-definidas
    - ....

# Alocação Dinâmica

- Função “sizeof”:
  - retorna o número de bytes ocupado por um tipo
- Função “malloc”:
  - recebe como parâmetro o número de bytes que se deseja alocar
  - retorna um ponteiro genérico para o endereço inicial da área de memória alocada, se houver espaço livre:
    - ponteiro genérico é representado por void\*
    - ponteiro é convertido automaticamente para o tipo apropriado
    - ponteiro pode ser convertido explicitamente
  - retorna um endereço nulo, se não houver espaço livre:
    - representado pelo símbolo NULL

# Alocação Dinâmica

- Exemplo:
  - alocação dinâmica de um **vetor de inteiros com 10 elementos**
    - malloc retorna o endereço da área alocada para armazenar valores inteiros
    - ponteiro de inteiro recebe endereço inicial do espaço alocado

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



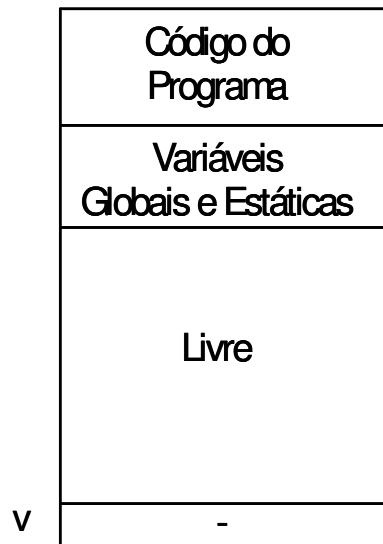
# Alocação Dinâmica

- Exemplo (cont.):

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

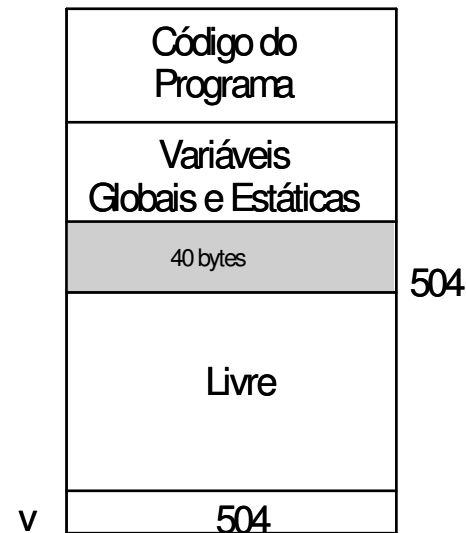
1 - Declaração: `int *v`

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



# Alocação Dinâmica

- Exemplo (cont.):
  - v armazena endereço inicial de uma área contínua de memória suficiente para armazenar 10 valores inteiros
  - v pode ser tratado como um vetor declarado estaticamente
    - v aponta para o início da área alocada
    - v[0] acessa o espaço para o primeiro elemento
    - v[1] acessa o segundo
    - .... até v[9]

# Alocação Dinâmica

- Exemplo (cont.):
  - tratamento de erro após chamada a `malloc`
    - imprime mensagem de erro
    - aborta o programa (com a função `exit`)

```
...  
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 */  
}  
...
```

# Alocação Dinâmica

- função “free”:
  - recebe como parâmetro o ponteiro da memória a ser liberada
    - a função free deve receber um endereço de memória que tenha sido alocado dinamicamente

```
free (v);
```

```
/* 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;  
}
```

# Vetores Locais a Funções

- Área de memória de uma variável local:
  - só existe enquanto a função que declara a variável estiver sendo executada
  - requer cuidado quando da utilização de vetores locais dentro de funções
- Exemplo:
  - produto vetorial de dois vetores **u** e **v** em 3D, representados pelas três componentes x, y, e z

$$\mathbf{u} \times \mathbf{v} = \{ u_y v_z - v_y u_z, \quad u_z v_x - v_z u_x, \quad u_x v_y - v_x u_y \}$$

# Vetores Locais a Funções

```
float* prod_vetorial (float* u, float* v)
{
    float p[3];
    p[0] = u[1]*v[2] - v[1]*u[2];
    p[1] = u[2]*v[0] - v[2]*u[0];
    p[2] = u[0]*v[1] - v[0]*u[1];
    return p; /* ERRO: não podemos retornar endereço de área local (p é vetor) */
}
```

– variável p declarada localmente:

- área de memória que a **variável p** ocupa deixa de ser válida quando a função **prod\_vetorial** termina
- função que chama **prod\_vetorial** não pode acessar a área apontada pelo valor retornado



# Vetores Locais a Funções

```
float* prod_vetorial (float* u, float* v)
{
    float *p = (float*) malloc(3*sizeof(float));
    p[0] = u[1]*v[2] - v[1]*u[2];
    p[1] = u[2]*v[0] - v[2]*u[0];
    p[2] = u[0]*v[1] - v[0]*u[1];
    return p;
}
```

- variável `p` alocada dinamicamente
  - área de memória que a **variável `p`** ocupa permanece válida mesmo após o término da função **`prod_vetorial`**
  - função que chama **`prod_vetorial`** pode acessar o ponteiro retornado
  - problema - alocação dinâmica para cada chamada da função:
    - ineficiente do ponto de vista computacional
    - requer que a função que chama seja responsável pela liberação do espaço

# Vetores Locais a Funções

```
void prod_vetorial (float* u, float* v, float* p)
{
    p[0] = u[1]*v[2] - v[1]*u[2];
    p[1] = u[2]*v[0] - v[2]*u[0];
    p[2] = u[0]*v[1] - v[0]*u[1];
}
```

- espaço de memória para o resultado passado pela função que chama:
  - função `prod_vetorial` recebe três vetores,
    - dois vetores com dados de entrada
    - um vetor para armazenar o resultado
  - solução mais adequada pois não envolve alocação dinâmica

# Resumo

## Vetor:

- alocação de vetor: `tipo nome[ #elementos];`
- acesso a cada elemento: `através de indexação da variável`
- nome do vetor: `aponta para endereço inicial`

## Funções para gerência de memória:

- `sizeof` retorna o número de bytes ocupado por um tipo
- `malloc` recebe o número de bytes que se deseja alocar  
retorna um ponteiro para o endereço inicial, ou  
retorna um endereço nulo (NULL)
- `free` recebe o ponteiro da memória a ser liberada