Métodos Computacionais

Vetores e Matrizes Dinâmicas

Vetores

- Um programa para o cálculo da média

Média $\Rightarrow$ $$m = \frac{\sum_{i=1}^{n} x_i}{n}$$

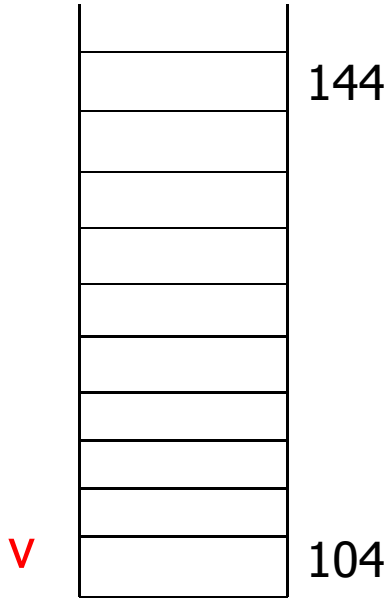
Variância $\Rightarrow$ $$v = \frac{\sum_{i=1}^{n} (x_i - m)^2}{n}$$

A forma mais simples de estruturar um conjunto de dados é por meio de vetores

Vetores na Memória

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

Aloca espaço para 10 valores inteiros, referenciados por **v**



É reservado um espaço de memória contínuo

`v[0]` → Acessa o primeiro elemento de v

⋮ ⋮

`v[9]` → Acessa o último elemento de v

Mas: `v [10]` → **Está errado !**

Checando os Limites dos Vetores

- Ao utilizar o índice para referenciar um elemento do vetor, este índice deve permitir o acesso a um elemento válido
  - Em um endereço de memória alocado para o vetor
  - Índice deve variar de *0* a *tamanho -1*
- C não avisa quando o limite de um vetor é excedido!
  - Valores serão armazenados em posições ocupadas por outros dados ou mesmo pelo código do próprio programa

O programador tem a responsabilidade de verificar o limite do vetor!

Inicializando Vetores

- Vetores podem ser inicializados na declaração

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

ou simplesmente:

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

Vetor é dimensionado pelo número de elementos inicializados

Vetores só podem ser inicializados no ato da declaração!

Checando os Limites dos Vetores

- São comuns, erros de programação no uso de vetores dentro de laços
  - Deve-se prestar atenção na parte de teste do laço

```
int main() {
      int pares[20];
      int i,somaPares = 0;
      for (i = 0; i <= 20; i++) {
        pares[i] = 2 * i ;
          somaPares = somaPares + pares[i];
      }
      ...
}
```

Teste deveria ser `i < 20`

Por causa do teste errado, esta linha gerará um erro

Calculando Média e Variância

```
#include <stdio.h>
int  main(){
  float v[10];
  float med = 0.0, var = 0.0 ;
  int i  ;
  for(i = 0; i < 10; i++)
      scanf ("%f",&v[i]) ;
  for(i = 0; i < 10;i++)
      med = med + v [i] ;
  med = med/10 ;
  for(i = 0; i < 10; i++)
      var = var +(med - v[i])*(med - v[i]);
  var = var/10  ;
  printf("Media = %f e Variancia = %f\n",med,var);
  return   0 ;
}
```

Lê cada elemento do vetor

Calcula a média

Calcula a variância

Associação entre Vetores e Ponteiros

- Considere a declaração:

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

  - O símbolo v
    - Representa o vetor
    - É uma constante que representa seu endereço inicial
    - Aponta para o primeiro elemento do vetor

Ponteiros e Vetores (matrizes)

- Em C existe um relacionamento muito forte entre ponteiros e vetores
  - O compilador entende todo vetor e matriz como ponteiros, pois a maioria dos computadores é capaz de manipular ponteiros e não vetores
  - Qualquer operação que possa ser feita com índices de um vetor pode ser feita com ponteiros
  - O identificador de um vetor representa um endereço, ou seja, um ponteiro

Ponteiros e Vetores

- Como vimos, C permite aritmética de ponteiros
- Se tivermos a declaração

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

- Podemos acessar elementos do vetor através de aritmética de ponteiros

`v + 0` ⟶ Aponta para (igual ao endereço do) primeiro elemento do vetor

`v + 1` ⟶ Aponta para o segundo elemento do vetor

⋮

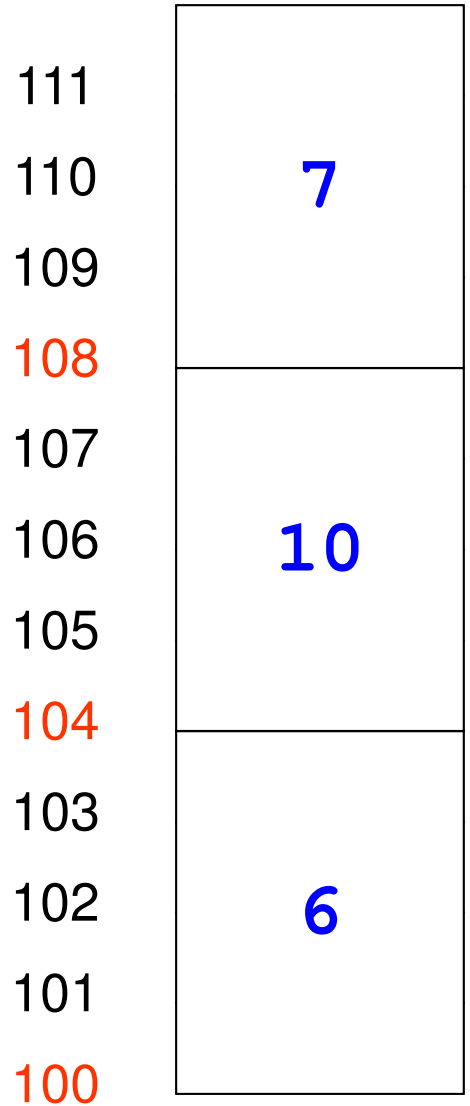
`v + 9` ⟶ Aponta para o último elemento do vetor

- Portanto: `&v[i]↔(v + i)` `v[i]↔ *(v + i)`

Representando Ponteiros e Vetores na Memória

Memória

Vetores podem ser tratados como ponteiros em C!




```
int v[] = {6,10,7};
```

```
*(v + 2) ↔ v[2] ↔ 7
```

```
v + 2 ↔ &v[2] ↔ 108
```

```
*(v + 1) ↔ v[1] ↔ 10
```

```
v + 1 ↔ &v[1] ↔ 104
```

```
*v ↔ v[0] ↔ 6
```

```
v  ↔ &v[0] ↔ 100
```

Usando Notação de Ponteiros para Vetores

Versão com Vetor

```
int main( ) {
    int nums[ ] = {1, 4, 8};
    int cont;
    for(cont=0; cont < 3; cont++) {
        printf("%d\n,nums[cont]);
    }
 }
```

Versão com Ponteiro

```
int main( ) {
    int nums[ ] = {1, 4, 8};
    int cont;
    for(cont=0; cont < 3; cont++) {
        printf("%d\n,*(nums + cont));
    }
 }
```

Ponteiros Constantes x Ponteiros Variáveis

```
int main( ) {
    int nums[ ] = {1, 4, 8};
    int cont;
    for(cont=0; cont < 3; cont++) {
       printf("%d\n",*(nums++));
    }
 }
```

Declaração de uma constante do tipo ponteiro para inteiros (ponteiro constante)

Errado!

Tenta incrementar endereço armazenado na constante `nums` e atualizar a constante com novo endereço

Ponteiros Constantes x Ponteiros Variáveis

```
int main( ) {
    int nums[ ] = {1, 4, 8};
    int* pnums = nums;
    int cont;
    for(cont=0; cont < 3; cont++) {
       printf("%d\n,*(pnums++) );
    }
 }
```

Declaração de uma variável do tipo ponteiro para inteiros (ponteiro variável)

Certo!

Incrementa endereço armazenado na variável pnums e atualiza a variável com novo endereço

Ponteiros Constantes x Ponteiros Variáveis

```
int a[10];
int *pa;
pa = a;
```

Atribui a uma variável um novo endereço: CERTO!

```
int a[10];
int *pa;
a = pa;
```

Atribui a uma constante um novo endereço: ERRADO!

Passagem de Vetores para Funções

- Passagem de vetores para funções implica em ter um parâmetro do tipo ponteiro nesta função

```
#include <stdio.h>
float media(int n,float* v) {
  int i;
  float s = 0.0;
  for(i = 0; i < 10; i++)
      s  +=  v [ i ] ;
  return  s/n;
}

float variancia(int n, float* v, float m ){
  int i;
  float  s = 0.0 ;
  for(i = 0; i < 10;i ++)
  s += (v[i] - m)*(v[i]- m ) ;
     return  s / n ;
}
```

Endereço da primeira posição do vetor

Passagem de Vetores para Funções

```
int  main(){
   float  v[10];
   float  med,var ;
   int i;
   for(i = 0; i < 10; i++)
       scanf("%f",&v[i]) ;
   med  =  media(10, v) ;
   var  =  variancia(10, v,med ) ;
   printf ("Media = %f e Variancia = %f\n",med,var);
   return 0 ;
}
```

Endereço da primeira posição do vetor

Alterando Valores de Vetores Dentro de Funções

```
# include <stdio.h>
void incrementa(int n, int* v){
   int i ;
   for(i = 0; i < n; i++)
       v[i]++;
}

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

Como endereço é passado, podemos alterar um vetor dentro de uma função

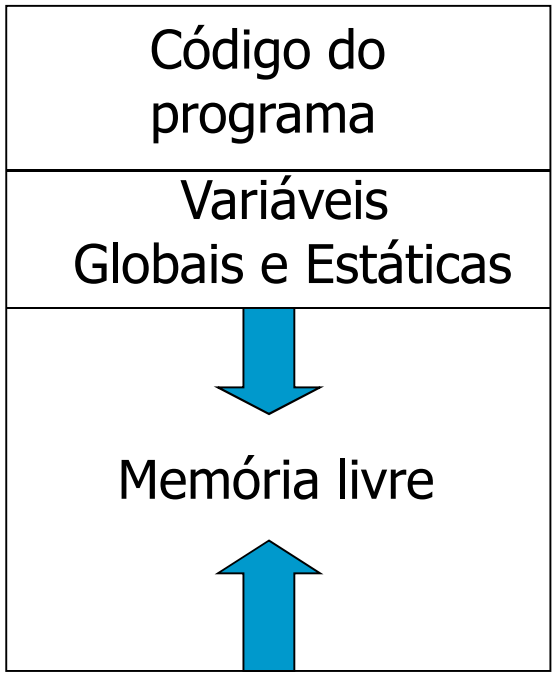
Vetores e Alocação Dinâmica

- Alocação Dinâmica consiste em requisitar memória em tempo de execução

- Modos de reservar espaço em memória:
  - Variáveis globais (e estáticas)
    - Espaço reservado existe enquanto o programa estiver sendo executado
  - Variáveis locais
    - Espaço existe enquanto a função, que declarou a variável, estiver sendo executada.
  - Requisitar memória em tempo de execução
    - Espaço alocado dinamicamente permanece reservado até que seja ***explicitamente*** liberado pelo programa.

.

Vetores e Alocação Dinâmica



Esquema de alocação de memória da pilha de execução

Alocação Dinâmica em C

- Função básica para alocar memória é **malloc** presente na biblioteca **stdlib.h**

```
void* malloc(unsigned qtdBytes);
```

  - Recebe como argumento um número inteiro sem sinal que representa a quantidade de bytes que se deseja alocar
  - Retorna o endereço inicial da área de memória alocada.

Alocação Dinâmica em C com `malloc`

- Aloca somente a quantidade de memória necessária
  - Exemplo:

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

Se a alocação for bem sucedida, v armazenará o endereço inicial de uma área contínua de memória suficiente para armazenar 10 valores inteiros (40 bytes)

Alocação Dinâmica em C com `malloc`

- Uso do comando **`sizeof`** para ter independência de plataforma de desenvolvimento
  - Exemplo:

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

- Função *malloc* retorna um ponteiro genérico, para qualquer tipo, representado por *void
  - Faz-se a conversão para o tipo apropriado usando o operador de molde de tipo *(cast)*

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

Erro na Alocação

- Se não houver espaço livre suficiente para realizar a alocação, a função **`malloc`** retorna um **endereço nulo**
  - É representado pelo símbolo **`NULL`**
  - É uma boa prática de programação testar se a alocação foi bem sucedida para evitar erros de execução

Liberando Espaço Alocado

- Uso da função **`free`** para liberar espaço de memória alocada dinamicamente

```
void free(void* endereco);
```

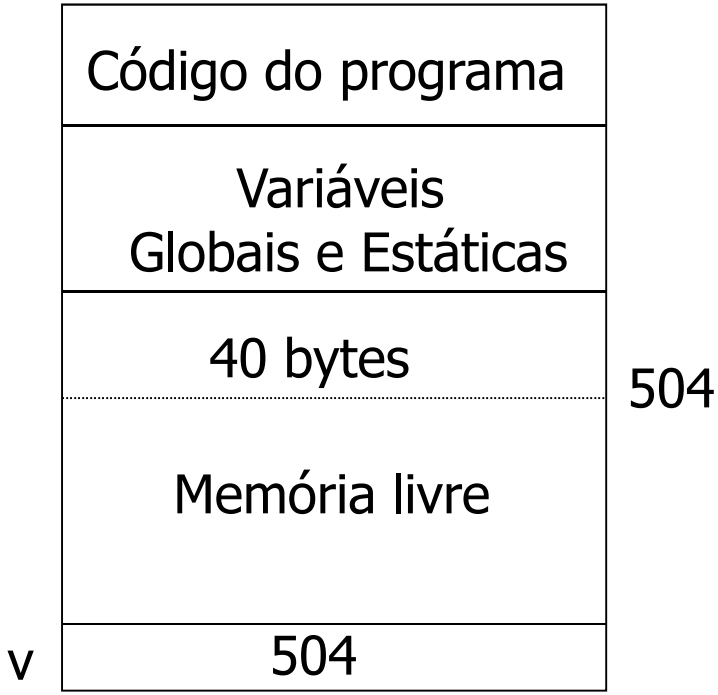
  - Recebe como parâmetro o ponteiro da memória a ser liberada
  - O espaço de memória fica livre para ser alocado futuramente pelo próprio programa ou outro programa
  - Recomenda-se liberar espaço de memória previamente alocado que não é mais necessário
    - Evita desperdício

Memória e Alocação Dinâmica

1) Declaração: int *v ; 2) v=(int*)malloc(10*sizeof (int));

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

Reserva-se o espaço de memória da área livre e atribui o endereço à v



Usando Alocação Estática para Calcular Média com Vetores

```
int  main(){
  float  v[2000];
  float  med,var ;
  int i, qtdNumeros;
  do{
      printf("Quantidade de numeros? (< 2000):\n");
      scanf("%d", &qtdNumeros);
   } while (qtdNumeros <= 0 || qtdNumeros > 2000);

  for(i = 0; i < qtdNumeros; i++)
      scanf("%f",&v[i]) ;
  med  =  media(10, v) ;
  var  =  variancia(10, v,med ) ;
  printf ("Media = %f e Variancia = %f\n",med,var);
  return 0 ;
}
```

Declaração estática do tamanho do vetor limita aplicação

Tamanho pode ser insuficiente ou grande demais (desperdício)

Vetores Dinâmicos

- Declaração de vetores implicam em alocação estática de memória
- Com alocação dinâmica, podemos criar algo como um vetor cujo tamanho é decidido em tempo de execução, ou seja um **vetor dinâmico**
- Para tal, usaremos variáveis do tipo ponteiro que receberão os endereços iniciais do espaço alocado dinamicamente
  - Com o endereço inicial, podemos navegar pelo vetor

Usando Alocação Dinâmica para Calcular Média com Vetores Dinâmicos

```
int main() {
    float* v;
    float med, var;
    int i, qtdNumeros;
    do{
       printf("Quantidade de numeros?:\n");
       scanf("%d", &qtdNumeros);
    } while (qtdNumeros <= 0);
    v = (float*) malloc(qtdNumeros*sizeof(float));
    if (v == NULL)
       printf("Memoria insuficiente");exit(1);
    for(i = 0; i < qtdNumeros; i++)
       scanf("%f",&v[i]) ;
  med  =  media(10, v) ;
  var  =  variancia(10, v,med ) ;
  printf ("Media = %f e Variancia = %f\n",med,var);
  return 0 ;
}
```

Ponteiro recebe endereço de espaço alocado dinamicamente (vetor dinâmico)

Tamanho do vetor é determinado pelo usuário

Vetores e Alocação Dinâmica

- Vetores Locais e 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  ;
}
```

ERRADO! – Endereço local é retornado

A variável retornada é declarada localmente. Por isso, sua área de memória deixa de ser válida quando a função termina.

Vetores e Alocação Dinâmica

- Vetores Locais e Funções

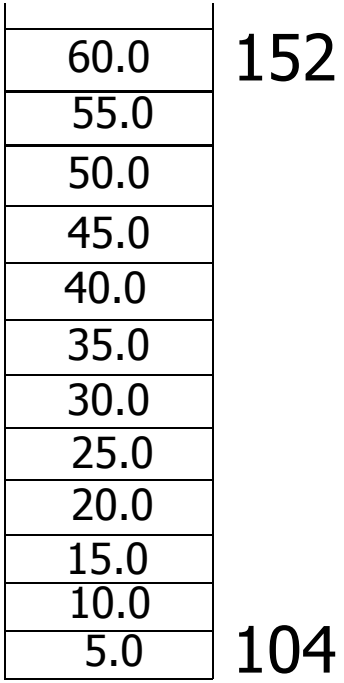
```
float*   prod_vetorial (float*  u , float*  v) {
      float*  p = (float*) malloc(3 * sizeof(float)) ;
      p[0] = u [ 1 ] * v [ 2 ]  -  v [ 1 ]  * u [ 2 ] ;
      p[1] = u [ 2 ] * v [ 0 ]  -  v [ 2 ]  * u [ 0 ] ;
      p[2] = u [ 0 ] * v [ 1 ]  -  v [ 0 ]  * u [ 1 ] ;
      return  p  ;
}
```

CERTO! - Endereço alocado dinamicamente fica disponível até que seja liberado explicitamente

Matrizes

- Vetores Bidimensionais e Matrizes

```
float mat [ 4 ] [ 3 ];
float mat [ 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} }
```



É reservado um espaço de memória contínuo para armazenar os 12 elementos.

Após a declaração estática, a variável **mat** representa um ponteiro para o primeiro "vetor-linha", composto de 3 colunas.

Inicializando Matrizes

- A inicialização de uma matriz também pode ser feita das seguintes formas:

```
float mat[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}}
```

```
float mat[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}
```

```
float mat [][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}
```

Deve ser passada a segunda dimensão

Passando Matriz para Função

```
void imprimeMatriz(int linhas, int mat[][2]) {
   int i,j;
   for (i=0; i < linhas; i++){
       for (j=0;j < 2; j++){
             printf("%d ",mat[i][j]);
       }
       printf("\n");
   }
}
int main() {
   int matriz[][2] = {{6,10},{7,11}};
   imprimeMatriz(2,matriz);
   return 0;
}
```

Parâmetro do tipo vetor de vetores de int

Parâmetro deve indicar número de colunas

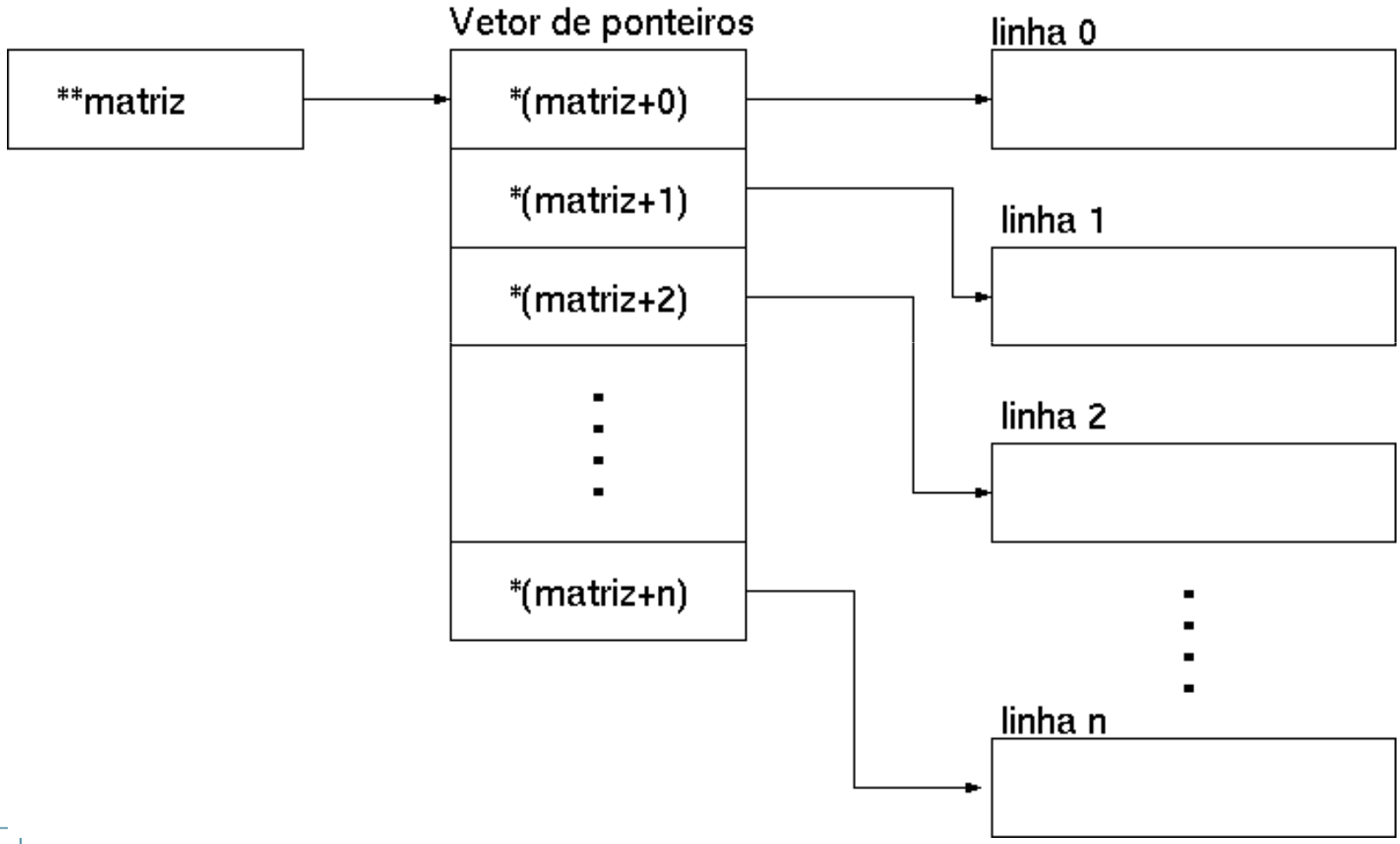
Criando Matrizes Dinâmicas

- Deve-se usar um ponteiro para ponteiro

```
int** matriz;
```

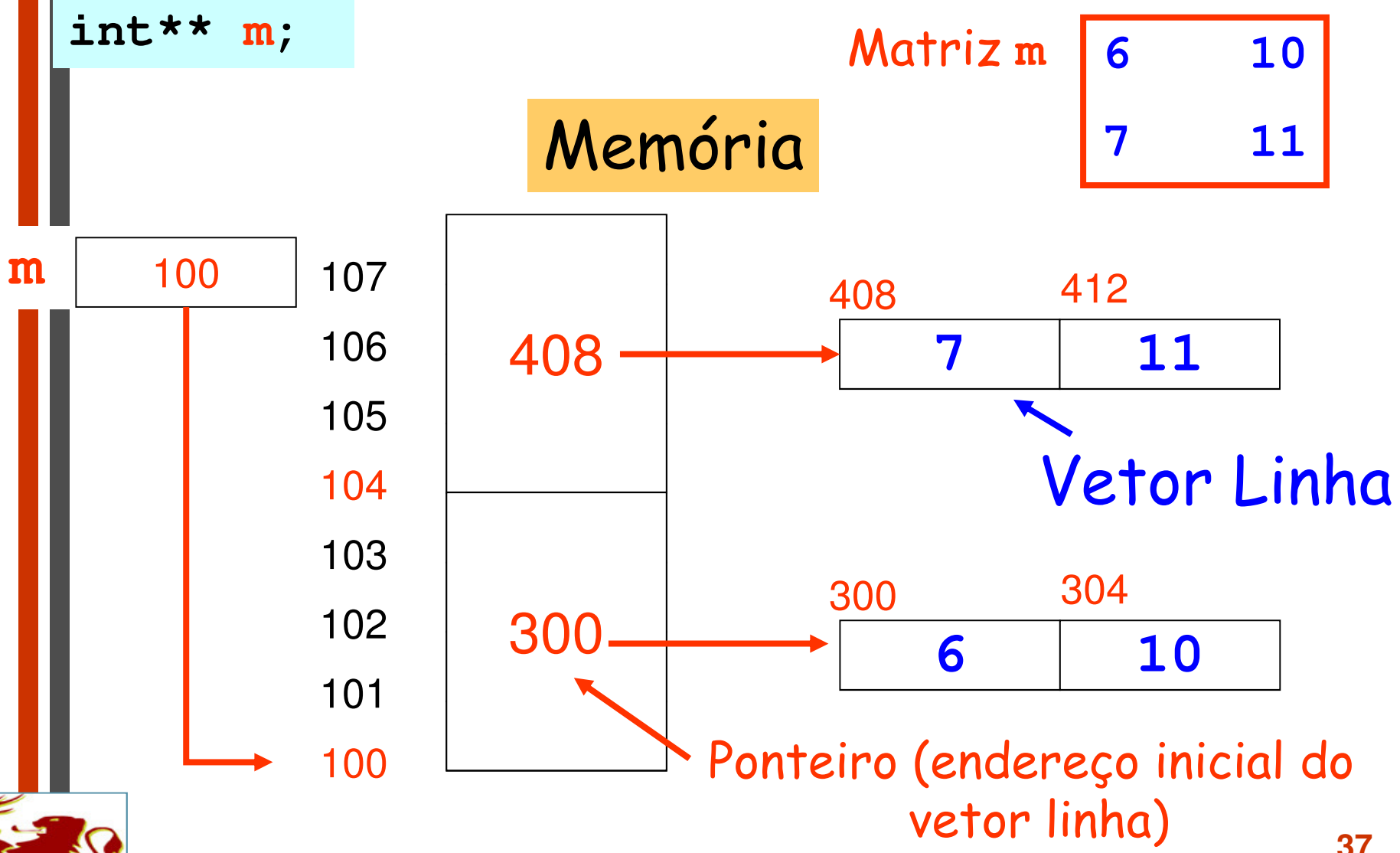
- Pode-se pensar em uma matriz dinâmica como um vetor de ponteiros
  - Cada elemento do vetor contém o endereço inicial de uma linha da matriz (vetor-linha)
  - Para alocar uma matriz com malloc, é preciso fazer a alocação do vetor de ponteiros e, em seguida, de cada vetor-linha
  - Da mesma forma, a liberação da memória é feita em partes

Ponteiro para ponteiro



Representando Matrizes Dinâmicas na Memória

```
int** m;
```



Alocando uma Matriz Dinâmica

```
int main () {
     int linhas, colunas,i,j;
   printf("Numero de linhas e colunas da matriz:\n");
   scanf("%d %d",&linhas,&colunas);
   float** mat ;
   mat = (float**) malloc(linhas * sizeof(float*));
   for (i = 0 ; i< linhas ; i++) {
       mat[i]= (float*) malloc(colunas * sizeof(float)) ;
   }
   printf("\n Digite os elementos da matriz:\n");
   for(i=0; i< linhas; i++) {
       for(j=0; j< colunas; j++) {
         printf("Elemento [%d][%d] = ",i,j);
         scanf("%f",&mat[i][j]);
         printf("\n");
       }
   }
   /* continua */
```

Alocando vetores-linha

Alocando vetor de ponteiros

Liberando uma Matriz Dinâmica

```
void liberaMatriz (int** mat, int linhas) {
     int i,j;
   for(i=0; i< linhas; i++) {
      free(mat[i]);
   }

   free(mat);

}
```

Libera primeiro cada vetor linha

Libera depois vetor de ponteiros

Passando Matrizes Dinâmicas como Argumentos

```
void imprimeMatriz(int linhas, int** mat) {
   int i,j;
   for (i=0; i < linhas; i++){
       for (j=0;j < 2; j++){
             printf("%d ",mat[i][j]);
       }
       printf("\n");
   }
}
```

Parâmetro do tipo ponteiro para ponteiro de `int`

Acesso usando notação de ponteiro:

`*(*(mat+i)+j)`

Cuidado na Assinatura da Função

```
void imprimeMatriz(int linhas, int** mat) {
   int i,j;
   for (i=0; i < linhas; i++){
       for (j=0;j < 2; j++){
             printf("%d ",mat[i][j]);
       }
       printf("\n");
   }
}
int main() {
   int matriz[][2] = {{6,10},{7,11}};
   imprimeMatriz(2,matriz);
   return 0;
}
```

Parâmetro do tipo ponteiro de ponteiro de `int`

Endereço da matriz estática passada como argumento

Errado!