Introdução a Programação

Ponteiros e Strings, Alocação Dinâmica

Tópicos da Aula

- Hoje aprenderemos a relação entre ponteiros e strings
  - Ponteiros para strings X Vetores de Caracteres
  - Vetores de ponteiros para strings
- Também veremos como alocar espaço de memória em tempo de execução
  - Conceito de Alocação Dinâmica
  - Importância
  - Funções em C para alocação dinâmica
  - Usando alocação dinâmica para criar vetores e matrizes dinâmicas

Ponteiros e Strings

- Uma string é um vetor de caracteres

```
char   s [10] ;
```

- Vimos que **s** representa o endereço inicial do vetor
- Portanto, podemos manipular strings com ponteiros

Ponteiros e Strings

ERRADO

```
char capital[7];
capital = "Recife";
```

Declaração da constante do tipo vetor de caracteres `capital`

Já foi atribuido um endereco à constante `capital` (endereço inicial do vetor)

Tentativa de atribuir endereço da constante "Recife" à constante `capital`

Ponteiros e Strings

Declaração da constante do tipo vetor de caracteres `capital`

Vetor de caracteres inicializado com as letras que fazem parte de `Recife`

CORRETO

```
char capital[7] = "Recife";
char* cidade;
cidade = "Recife";
```

Atribuição do endereço da constante `"Recife"` à variável `cidade`

Inicialização de Strings Através de Ponteiros

```
int main() {
 char *salute="saudacoes";
 puts(++salute);
 return 0;
}
```

```
int main() {
 char salute[] ="saudacoes";
 puts(++salute);
 return 0;
}
```

Estes dois códigos executariam da mesma forma?

Imprime audacoes (sem s): ponteiro variável

Erro de compilação: tentativa de modificar ponteiro constante

Passando Ponteiros para Strings para Funções

- Na biblioteca **string.h**, existem várias funções que têm como parâmetros ponteiros para strings
  - Exemplos:
    - `char* strcpy(char* dest, char* origem);`
    - `int strcmp(char* s1, char* s2);`
- Podemos passar como argumentos, tanto ponteiros para strings ou vetores de caracteres(strings)

```
int main() {
 char *salute="saudacoes";
 char saudacao[10];
 strcpy(saudacao,salute);
 return 0;
}
```

Vetores de Strings x Vetores de Ponteiros para Strings

- Quando queremos armazenar um conjunto de strings podemos:
  - Usar um vetor de strings
    - Vetor bidimensional (matriz) de caracteres
  - Usar um vetor de ponteiros para strings

Usando Vetor de Strings (Matriz de Caracteres)

Programa verifica se um nome entrado pelo usuário pertence a um vetor inicializado de strings

```
int main(){
  int cont;
  int entra=0;
  char nome[40];
  char list[4][7]= {"Carlos", "Ana",  "Pedro", "Andre"};

  printf ("Digite seu nome:");
  gets(nome);
  for (cont=0; cont < 4; cont++){
      if (strcmp(list[cont],nome) == 0)
           entra=1;
  }
  if (entra == 1)
       printf ("Você pode entrar.");
  else
       printf ("Você não pode entrar.");
  return 0;
}
```

Cuidado para que as strings não excedam as colunas da matriz!

Usando Vetores de Ponteiros para Strings

Programa verifica se um nome entrado pelo usuário pertence a um vetor inicializado de ponteiros para strings

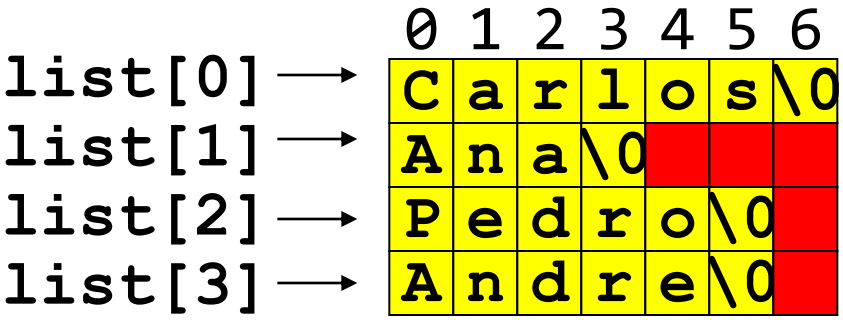
```
int main(){
  int cont;
  int entra=0;
  char nome[40];
  char* list[4]= {"Carlos", "Ana",  "Pedro", "Andre"};

  printf ("Digite seu nome:");
  gets(nome);
  for (cont=0; cont < 4; cont++){
      if (strcmp(list[cont],nome) == 0)
           entra=1;
  }
  if (entra == 1)
       printf ("Você pode entrar.");
  else
       printf ("Você não pode entrar.");
  return 0;
}
```

Despreocupação com tamanho das strings

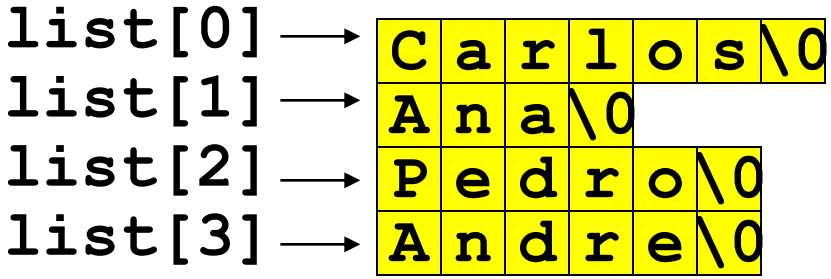
Inicialização de um Vetor de Strings x Vetor de ponteiros para Strings

Versão Vetor de Strings



Desperdício de Memória!

Versão Vetor de Ponteiros



Outro Exemplo Usando Vetor de Strings

```
int main(){
   int cont;
   int entra=0;
   char nome[40];
   char list[4][10];
   for (cont=0; cont < 4; cont++){
        printf("\nEntre com mais um convidado:\n");
        gets(list[cont]);
   }
   return 0;

}
```

Programa pede para o usuário preencher um vetor de strings

Cuidado para que as strings não excedam as colunas da matriz!

Outro Exemplo Usando Vetor de Ponteiro de Strings

```
int main(){
   int cont;
   int entra=0;
   char nome[40];
   char* list[4];
   for (cont=0; cont < 4; cont++){
       printf("\nEntre com mais um convidado:\n");
       gets(list[cont]);
   }

}
```

Programa pede para o usuário prencher um vetor de ponteiro para strings

ERRADO!

Tenta armazenar string em endereço (INVÁLIDO) apontado por `list[cont]`

Alocação de Memória

- Quando declaramos uma variável, o compilador reserva (aloca) um espaço na memória suficiente para armazenar valores do tipo da variável
  - Alocação estática (em tempo de compilação)

```
int var ;
```

Aloca espaço para 1 `int`

```
char s1 [10];
```

Aloca espaço para 10 `char`

```
char*  s2;
```

Aloca espaço para 1 endereço

Alocação Dinâmica

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

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

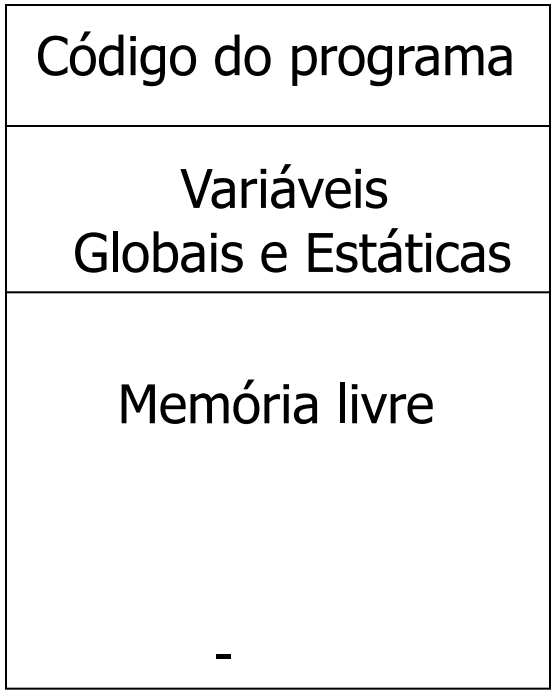
```
void free(void* endereco);
```

  - 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

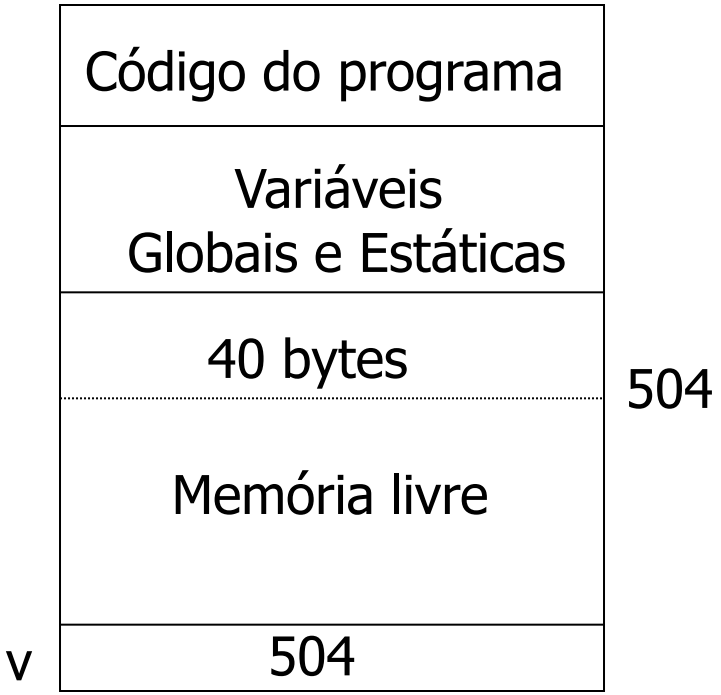
Memória e Alocação Dinâmica

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

```
#include <stdio.h>
int main() {
    int qtdNumeros,contador = 0;
    float numeros[2000];
    float media = 0.0;
    do{
        printf("Quantidade de numeros? (< 2000):\n");
        scanf("%d", &qtdNumeros);
    } while (qtdNumeros <= 0 || qtdNumeros > 2000);
    while (contador < qtdNumeros) {
        scanf("%f",&numeros[contador]);
        media = media + numeros[contador];
        contador++;
    }
    media = media/qtdNumeros;
    printf("\nA media eh %f\n");
    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() {
    int qtdNumeros,contador = 0;
    float* numeros;
    float media = 0.0;
    do{
        printf("Quantidade de numeros?:\n");
        scanf("%d", &qtdNumeros);
    } while (qtdNumeros <= 0);
    numeros = (float*) malloc(qtdNumeros*sizeof(float));
    if (numeros != NULL) {
        while (contador < qtdNumeros) {
                scanf("%f",&numeros[contador]);
                media = media + numeros[contador];
                contador++;
        }
    } /* continua */
}
```

Ponteiro recebe endereço de espaço alocado dinamicamente (vetor dinâmico)

Tamanho do vetor é determinado pelo usuário

Função Realloc

- Podemos mudar o espaço de memória alocado previamente de forma dinâmica
- Para isso, podemos utilizar a função **realloc**

```
void* realloc(void* ptr,unsigned qtdBytes);
```

  - Recebe endereço do bloco de memória alocado previamente
  - 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
    - Se endereço retornado for diferente do passado como parâmetro, conteúdo do bloco original é copiado para novo endereço

Usando Realloc

```
int main() {
    int qtdNumeros = 5,contador = 0;
    char resposta;
    float media = 0.0;
    float* nums;
    nums = (float*) malloc(qtdNumeros*sizeof(float));
    if (nums == NULL) {
        printf("Memoria insuficiente");exit(1);
    }
    printf("Programa calcula media de 5 numeros.");
    printf("Deseja mais/menos? (s/n)\n");
    scanf("%c",&resposta);
    if (resposta == 's') {
        do {
                printf("Quantidade de numeros?:\n");
                scanf("%d", &qtdNumeros);
        } while (qtdNumeros <= 0);
        nums = (float*) realloc(nums,qtdNumeros*sizeof(float));
    /* continua */
    }
```

Vetor é alocado dinamicamente

Tamanho do vetor muda dinamicamente

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
  - Forma Correta:

```
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

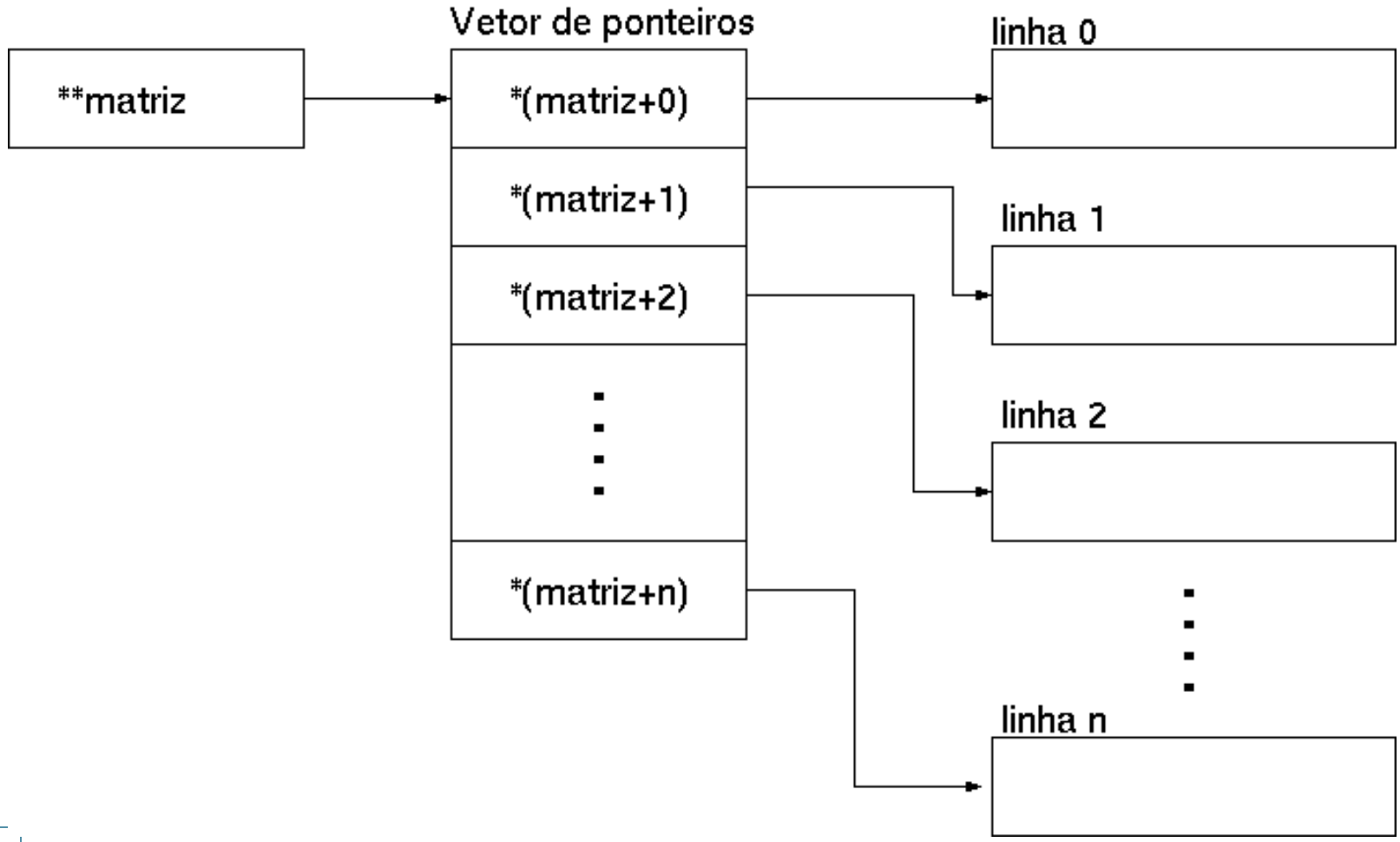
Criando Matrizes Dinâmicas

- Deve-se usar um ponteiro para ponteiro

```
int** matriz;
```

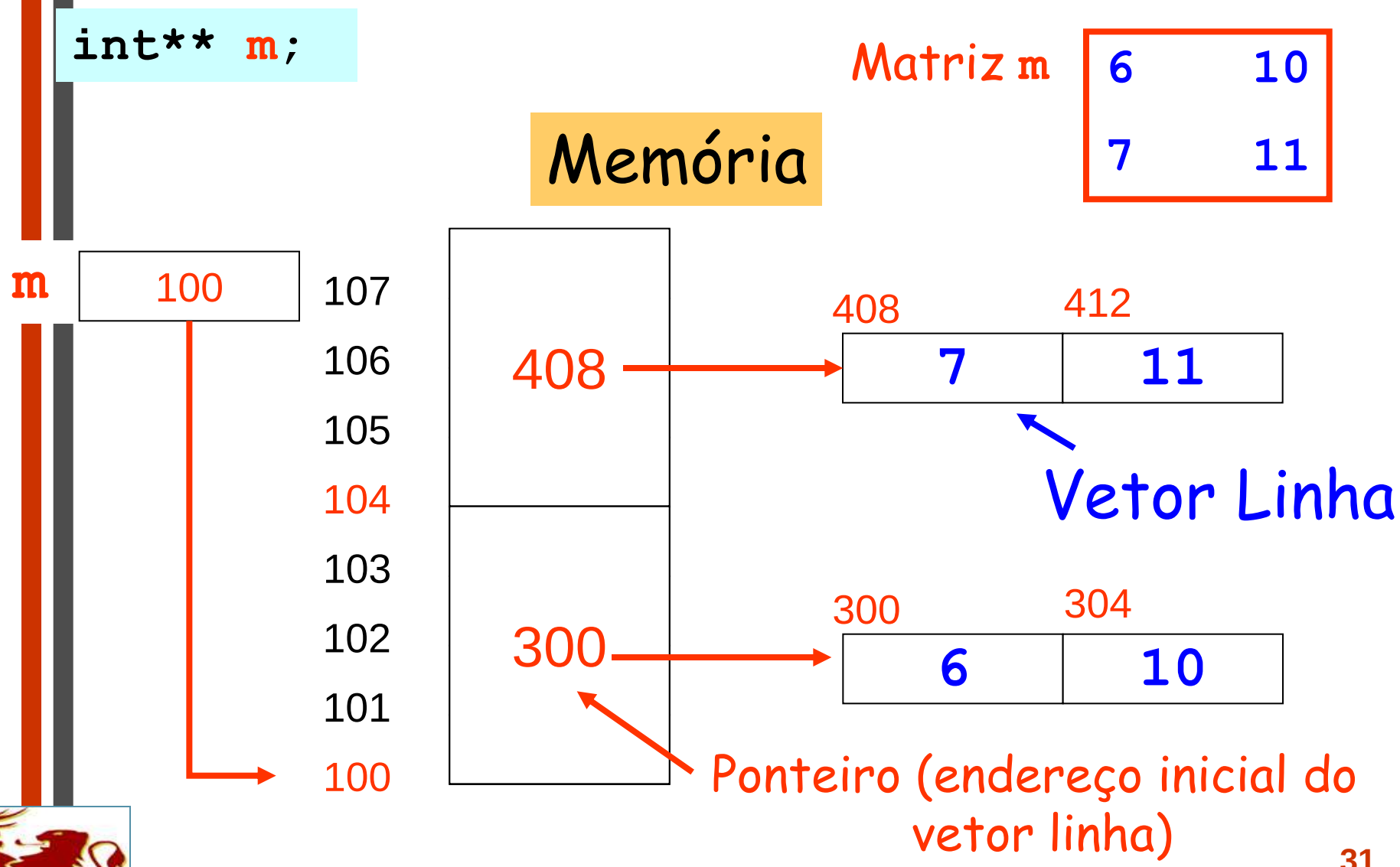
- Pode-se pensar em uma matriz dinâmica como um vetor de ponteiros
  - Cada elemento do vetor contém o endereço inicial de uma linha da matriz (vetor-linha)
  - Para alocar uma matriz com malloc, é preciso fazer a alocação do vetor de ponteiros e, em seguida, de cada vetor-linha
  - Da mesma forma, a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

Erro de Execução!

Cuidado na Assinatura da 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 tamanho 2 de int

Parâmetro deve indicar número de colunas

Certo!

Resumindo ...

- Relação entre ponteiros e strings
  - Ponteiros para strings X Vetores de Caracteres
  - Vetores de ponteiros para strings
- Alocação Dinâmica
  - Uso
  - Funções em C para alocação dinâmica
  - Usando alocação dinâmica para criar vetores e matrizes dinâmicas