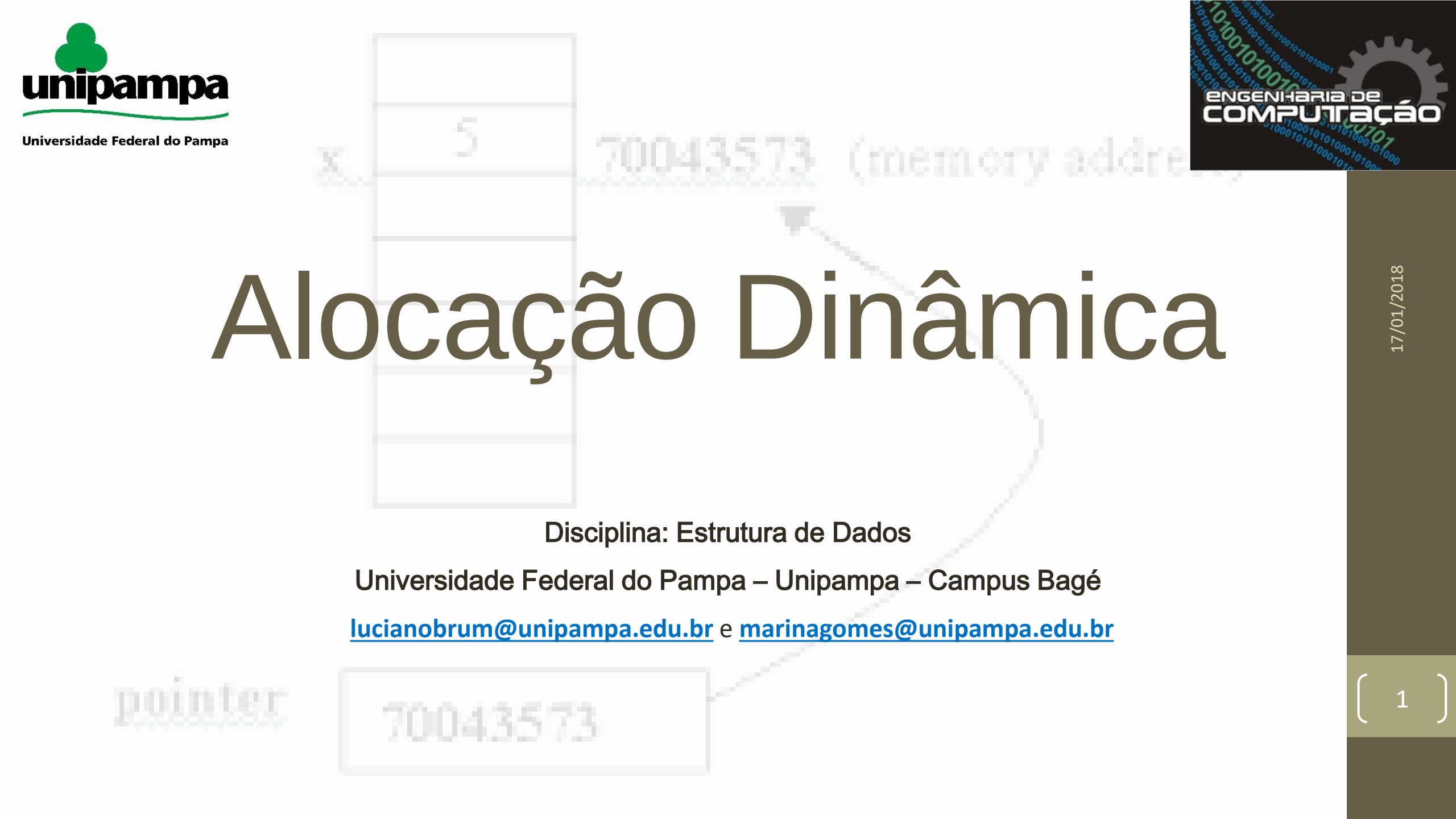


Alocação Dinâmica

Disciplina: Estrutura de Dados

Universidade Federal do Pampa – Unipampa – Campus Bagé

lucianobrum@unipampa.edu.br e marinagomes@unipampa.edu.br

Tópicos

- Alocação Dinâmica;
- Usos da memória;
- Funções da biblioteca padrão;
- Alocação Dinâmica de Vetores;
- Alocação Dinâmica de Matrizes;

Alocação Dinâmica

- Para declararmos um vetor, até o momento, precisamos sempre definir o seu tamanho. O mesmo para matrizes.

- Isto é um fator limitante:
  - Se dimensionarmos poucos elementos, o programa será muito limitado;
  - Se dimensionarmos muitos elementos, o programa vai consumir muita memória desnecessariamente;

- A linguagem C oferece mecanismos para requisitar em tempo de execução espaços de memória.

- Isso é a alocação dinâmica.

Usos da Memória

- Há 3 formas informais de reservar espaços de memória para armazenar informações:
  - Variáveis Globais e Estáticas: o espaço reservado existe enquanto o programa está em execução.
  - Variáveis Locais: o espaço reservado existe enquanto a função que declarou a variável está sendo executada.
  - Requisitar ao sistema, em tempo de execução, um espaço de determinado tamanho:
    - O espaço alocado é reservado até que seja explicitamente liberado pelo programa.
    - Com isso, podemos alocar espaço de memória em uma função e usar este mesmo espaço em outra.
    - Ao liberarmos este espaço, este será utilizado para outros fins e não poderá mais ser acessado. Quando o programa encerra, os espaços são automaticamente liberados.

Usos da Memória

- 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:
    - Usada 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.

Funções da Biblioteca Padrão

- A alocação dinâmica é gerenciada pelas funções **malloc**, **calloc**, **realloc** e **free** que estão na biblioteca **stdlib.**

- As mais utilizadas são as funções **malloc** e **free**. Além das funções mencionadas acima existem outras que não serão abordadas, pois não são funções padrão.

- As funções **malloc** e **calloc** são responsáveis por alocar memória, a **realloc** por realocar a memória e por último a **free** fica responsável por liberar a memória alocada.

- A alocação dinâmica é muito utilizada em casos em que usam muito volume de dados. Utilizado principalmente para variáveis indexadas (vetores e matrizes) e estruturas.

- **A variável deve ser do tipo ponteiro.**

Funções da Biblioteca Padrão: malloc

- A função *malloc* recebe como parâmetro o número de bytes que se deseja alocar e retorna o endereço inicial da área de memória alocada.

- Exemplificando o uso da função malloc:

  - Considere realizar a alocação dinâmica de um vetor de números inteiros com 20 elementos.

  - `int *v = malloc(20*4);`

  - V armazenará o endereço inicial de uma área contínua de memória suficiente para armazenar 20 inteiros.

  - V poderá então ser tratado como vetor declarado estaticamente.

Funções da Biblioteca Padrão : malloc

- `int *v = malloc(20*4);`

- Para ficar independente de compiladores e máquinas, usa-se o operador *sizeof*().

- `v = malloc(20*sizeof(int));`

- A função malloc retorna um ponteiro genérico (*void) para qualquer tipo de dado. Tal ponteiro pode ser convertido automaticamente pela linguagem para o tipo apropriado na atribuição.

- O mais comum é fazer explicitamente a conversão utilizando operador de cast.

- `v = (int*)malloc(20*sizeof(int));`

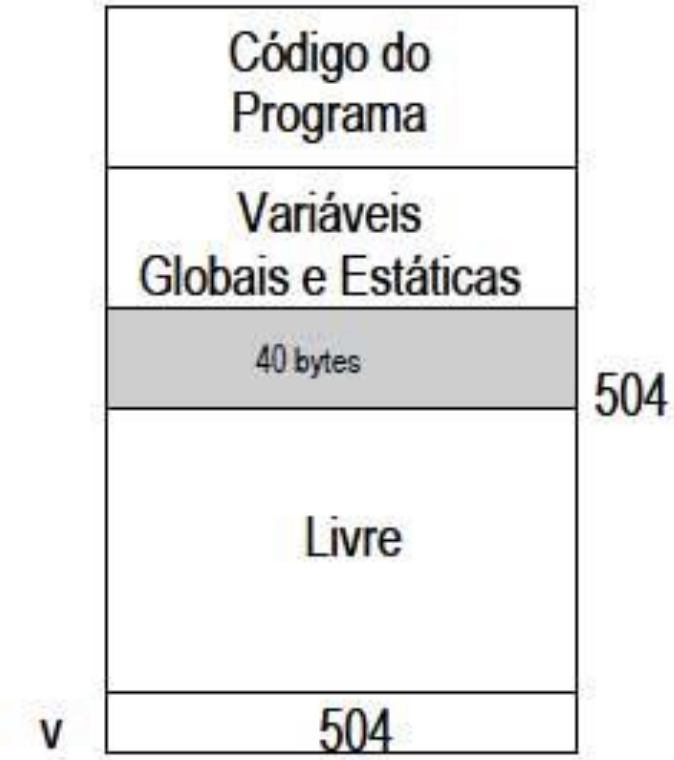
Funções da Biblioteca Padrão : malloc

1 - Declaração: int *v

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

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

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



Funções da Biblioteca Padrão : malloc

- Exemplo: Alocar memória para um vetor de caracteres de tamanho definido pelo usuário.

```
int n;
char *p;
scanf("%d",&n);
p = (char*)malloc(n*sizeof(char));
```

- Se não houver espaço livre para alocar a memória é retornado um endereço nulo (representado por NULL, definido na stdlib.h). É possível fazer tal teste após a tentativa de alocação.

```
p = (char*)malloc(n*sizeof(char));
if(p==NULL){
    printf("Memória insuficiente.\n");
    exit(1);
}
```

Funções da Biblioteca Padrão: realloc

- A sintaxe da função *malloc* é:
  - `void* malloc(size_t size);`

- Temos também a função *calloc (serve para alocar memória, como malloc)*:
  - `void* calloc(size_t num_itens, size_t tam_item);`

- Temos a função *realloc*, para realocar espaços de memória:

  - `void* realloc(void* ptr, size_t tam);`

  - O parâmetro `ptr` representa a região de memória que se deseja alterar. Já o parâmetro `tam` representa o novo tamanho da memória apontada por `ptr`. Este valor de `tam` pode ser maior ou menor que o original.

Funções da Biblioteca Padrão: realloc

```
#include<stdio.h>

int main(void){

    int* p;

    p=(int*)malloc(sizeof(int)*10);

    p=(int*)realloc(p,sizeof(int)*20);

    return 0;

}
```

Funções da Biblioteca Padrão: free

- Para liberar um espaço de memória alocado dinamicamente, utiliza-se a função free.
- Recebe como parâmetro ponteiro da memória a ser liberada.
- A sintaxe da função free é:
  - `void free(void* ponteiro);`
- Exemplo:
  - `int *p = (int*)malloc(10*sizeof(int));`
  - `free(p);`

Funções da Biblioteca Padrão

Exercício:

- Simule um vetor de n elementos, onde o usuário define quantos elementos deseja inserir. Faça uma alocação dinâmica de acordo com o número de elementos que o usuário escolheu. Preencha e mostre o vetor na tela e faça uma realocação de memória de acordo com valor solicitado novamente ao usuário. Mostre o vetor novamente e encerre o programa.

```
#include<stdio.h>
#include<stdlib.h>
int main(){
        int n, i;
        scanf("%d",&n);
        int *p = (int*)malloc(n*sizeof(int));
        for(i = 0; i < n; i++){
                scanf("%d",&p[i]);
        }
        for(i = 0; i < n; i++){
                printf("%d - ",p[i]);
        }
        scanf("%d",&n);
        p = (int*)realloc(p,n*sizeof(int));
        for(i = 0; i < n; i++){
                printf("%d - ",p[i]);
        }
        free(p);
        return 0;
}
```

Alocação Dinâmica de Vetores

- Vimos duas formas de alocar vetores:
  - Estática: usuário define tamanho máximo do vetor (no programa ou em tempo de execução) e aquele espaço de memória será utilizado até o fim da função.
  - Dinâmica: usuário define em tempo de execução o tamanho do vetor e pode redefini-lo. A área de memória ocupada pelo vetor permanece válida até que seja explicitamente liberada.
    - Vetor alocado dentro de uma função pode ser utilizado fora do corpo da função, enquanto estiver alocado.

Alocação Dinâmica de Vetores

- Exemplos:

```
float* produto_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;
}
```

Alocação Dinâmica de Vetores

- Exemplos:

```
float* produto_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;
}
```

Alocação Dinâmica de Vetores

- Exemplos:

```
void produto_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];
}
```

Alocação Dinâmica de Matrizes

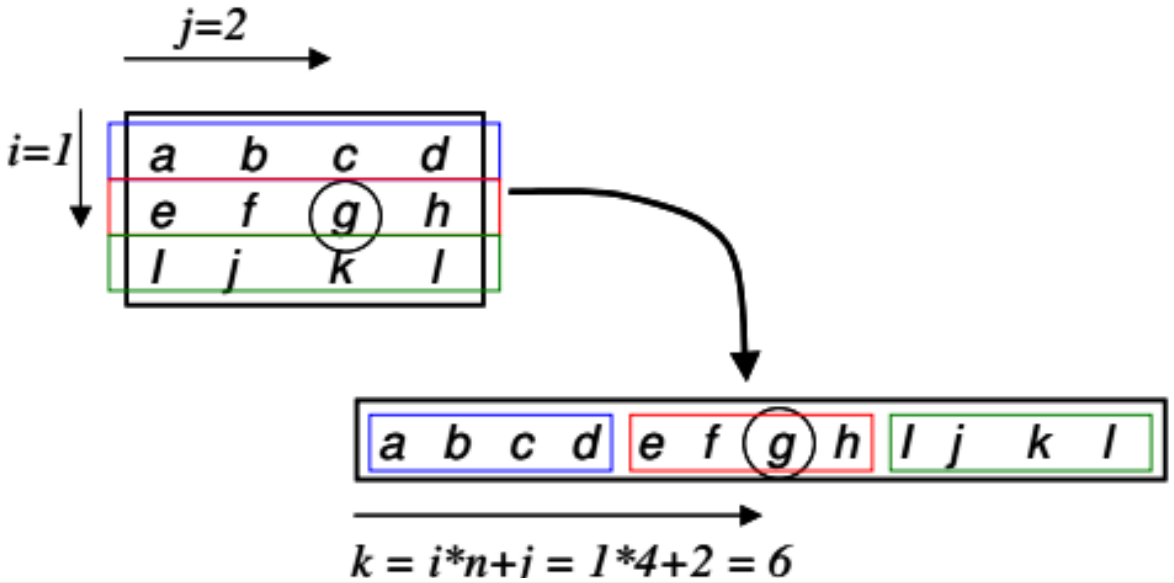
- Na alocação estática de uma matriz, assim como com vetores, é necessário saber de antemão as suas dimensões.
- O C permite somente alocação dinâmica de conjuntos unidimensionais.
- Como proceder?
  - É necessário criar abstrações conceituais com vetores para representar matrizes locadas dinamicamente. Utiliza-se a indireção múltipla (vetor de ponteiros)

Alocação Dinâmica de Matrizes

- Matriz representada por um vetor simples:
  - Estratégia:
    - Primeiras posições do vetor armazenam elementos da primeira linha seguidos dos elementos da segunda linha, e assim por diante;
- Exige disciplina para acessar os elementos da matriz.
- Conceitualmente, estamos trabalhando com uma matriz.
- Concretamente, estamos representando um vetor unidimensional.

Alocação Dinâmica de Matrizes

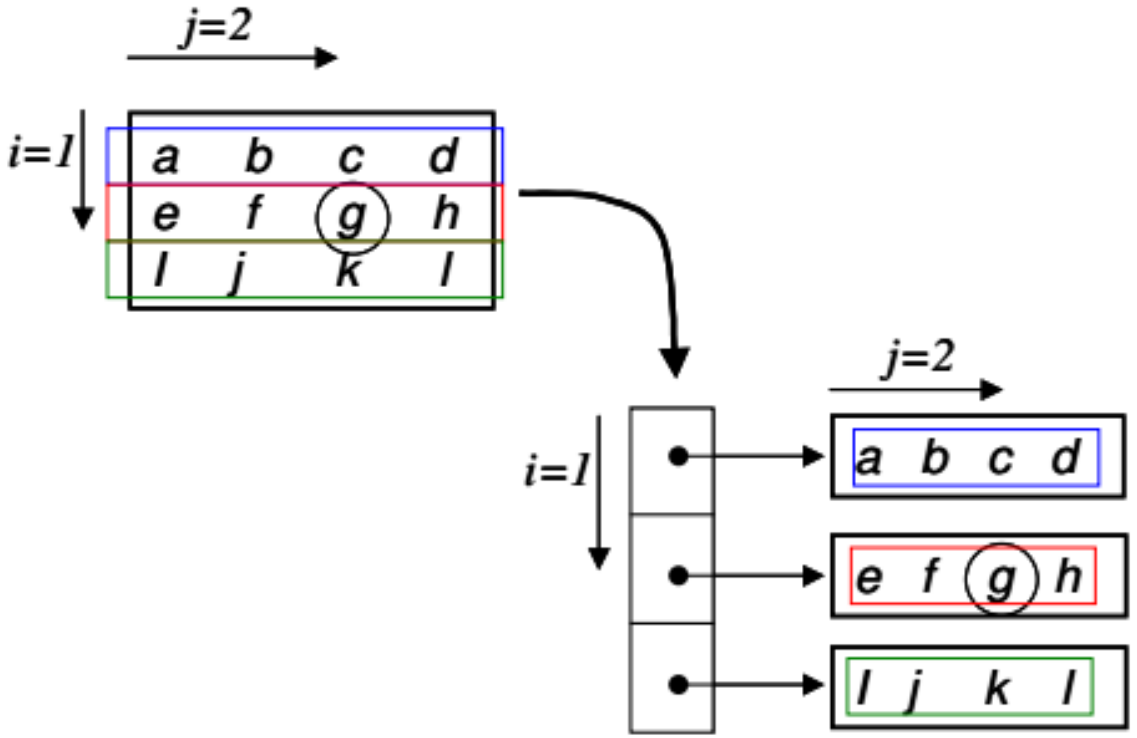
- Matriz representada por um vetor simples:
  - Matriz mat com n colunas representada no vetor v:
  - mat[i][j] mapeado em v[k] onde k=i*n+j.




```
float *mat;       /* matriz m x n representada por um vetor */
...
mat = (float*) malloc(m*n*sizeof(float));
```

Alocação Dinâmica de Matrizes

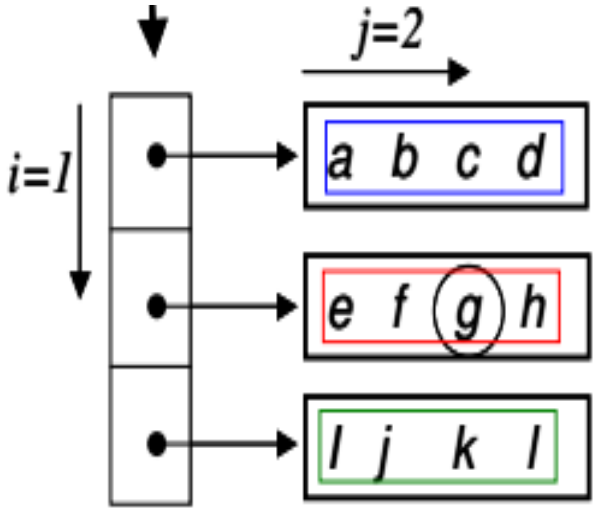
- Matriz representada por um vetor de ponteiros:
  - Cada elemento do vetor armazena o endereço do primeiro elemento de cada linha da matriz.
  - A matriz é representada por um vetor de vetores.



Alocação Dinâmica de Matrizes

- Matriz representada por um vetor de ponteiros:
  - Precisamos alocar memória para o vetor de ponteiros e atribuir o endereço das linhas da matriz.

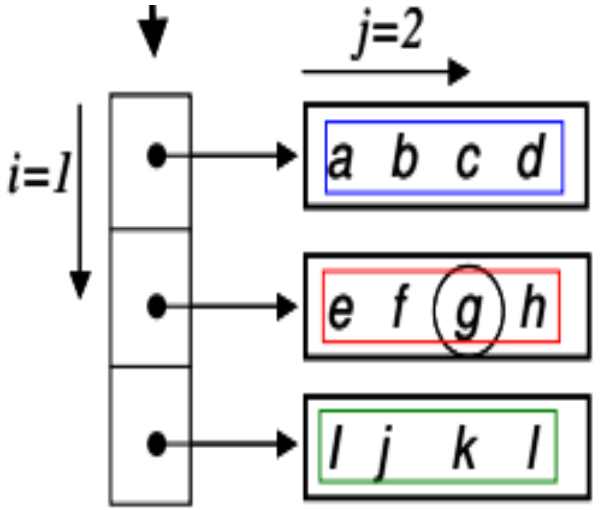
```
int i;
float** mat;/*vetor de ponteiros*/
...
mat=(float**)malloc(m*sizeof(float*));
for(i=0;i<m;i++){
mat[i]=(float*)malloc(n*sizeof(float));
}
```



Alocação Dinâmica de Matrizes

- Como liberar o espaço alocado para a matriz?

```
int i;
float** mat;/*vetor de ponteiros*/
...
mat=(float**)malloc(m*sizeof(float*));
for(i=0;i<m;i++){
    mat[i]=(float*)malloc(n*sizeof(float));
}
for (i=0; i<m; i++){
    free(mat[i]);
}
free(mat);
```



Alocação Dinâmica de Matrizes

- Exemplo: função transposta.
  - entrada: mat matriz de dimensão mxn.
  - saída: trp transposta de mat, alocada dinamicamente.
    - Q é a *matriz transposta* de M se e somente se *Qij=Mji.*
  - Solução 1: matriz alocada como vetor simples.
  - Solução 2: matriz alocada como vetor de ponteiros.

Alocação Dinâmica de Matrizes

```
/* Solução 1: matriz alocada como vetor simples */
float* transposta (int m, int n, float* mat)
{
   int i, j;
   float* trp;

   /* aloca matriz transposta com n linhas e m colunas */
   trp = (float*) malloc(n*m*sizeof(float));

   /* preenche matriz */
   for (i=0; i<m; i++)
      for (j=0; j<n; j++)
         trp[ j*m+i ] = mat[ i*n+j ];

   return trp;
}
```

Alocação Dinâmica de Matrizes

```
/* Solução 2: matriz alocada como vetor de ponteiros */
float** transposta (int m, int n, float** mat)
{
  int i, j;
  float** trp;

  /* aloca matriz transposta com n linhas e m colunas */
  trp = (float**) malloc(n*sizeof(float*));
  for (i=0; i<n; i++)
    trp[i] = (float*) malloc(m*sizeof(float));

  /* preenche matriz */
  for (i=0; i<m; i++)
    for (j=0; j<n; j++)
      trp[ j ] [ i ] = mat[ i ][ j ];

  return trp;
}
```

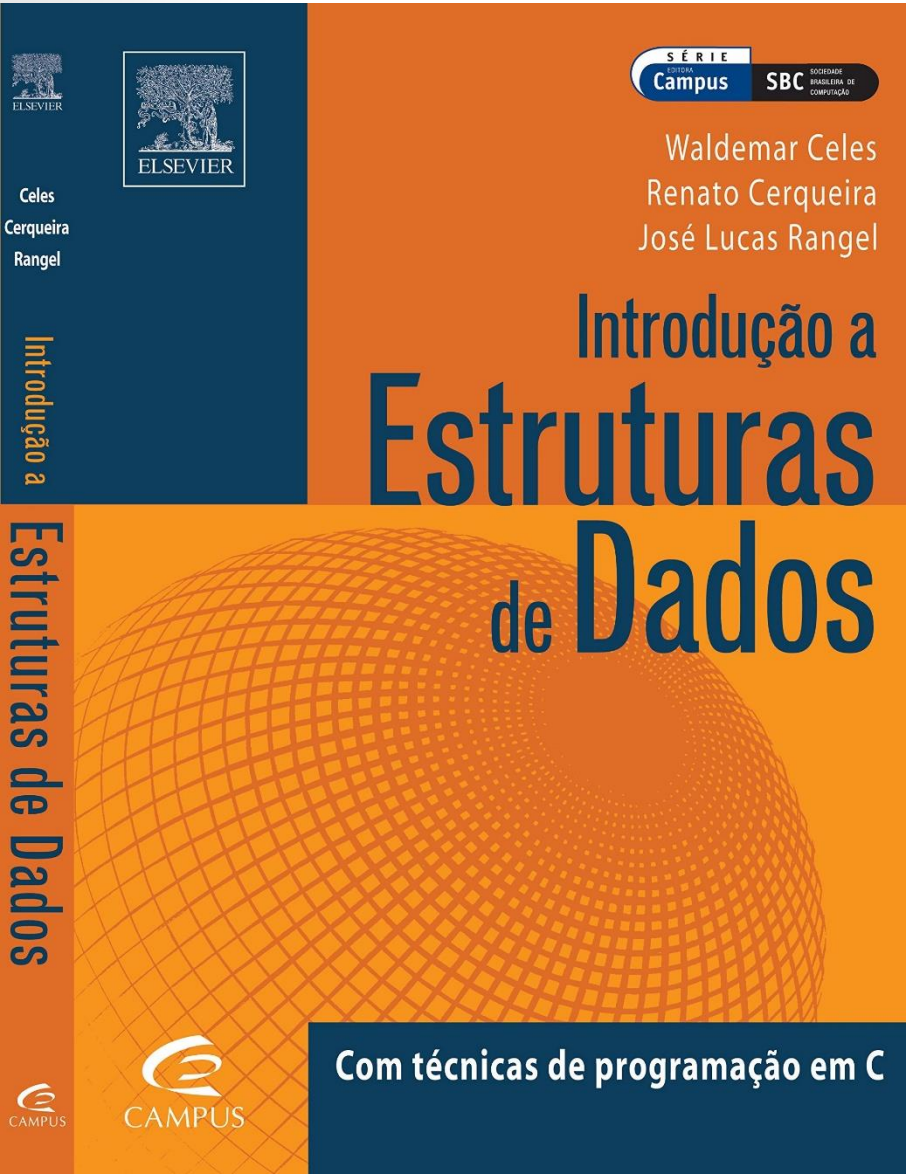
Resumo da Aula

- Vimos hoje:
  - Usos de memória em C;
  - Alocação dinâmica de memória;
  - Vetores e matrizes estáticas e dinâmicas;
  - Exemplos de códigos;

Complementando o Aprendizado

- **Leitura do livro Introdução a Estruturas de Dados (Celes, W):**
  - **Capítulo 5 (vetores).**
  - **Capítulo 6 (matrizes).**

Referências



- CELES, Waldemar; CERQUEIRA, Renato; RANGEL, José Lucas. Introdução a Estruturas de Dados com técnicas de programação em C. Rio de Janeiro: Elsevier (Campus), 2004. 4ª Reimpressão. 294 p.

Dúvidas ?