Alocação Dinâmica de Memória e Matrizes

Prof. Daniel Di Domenico
ddomenico@inf.ufsm.br

Introdução

- char c; int i; int v[10]; (alocação estática);
- Há casos em programação, em que precisamos lidar com dados dinâmicos:
  - Número de clientes numa **fila** aumenta e diminui durante o tempo de processo;
- A alocação dinâmica permite alocar memória em tempo de execução;
- A linguagem C disponibiliza quatro funções para utilizar este recurso.

Alocação Dinâmica

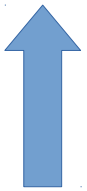
- **malloc:** aloca um bloco de bytes consecutivos e retorna um ponteiro para o primeiro byte do espaço alocado;
- **calloc:** aloca espaço de memória em bytes, inicializa-os com zeros e retorna um ponteiro para o bloco de memória alocado;
- **realloc:** modifica o tamanho do espaço de memória alocado previamente;
- **free:** libera um espaço de memória previamente alocado;

OBS: Estas funções estão disponíveis na biblioteca **<stdlib.h>**.

malloc ()

- Protótipo da função:

```
void * malloc ( unsigned int num);
```



Retorna um ponteiro do tipo void

Número de bytes a alocar

Um ponteiro nulo (NULL) é retornado se a memória não for alocada.

malloc ()

- Alocação de memória para um inteiro:

//aloca espaço para 1 inteiro

```
int * p = (int *)malloc(4); //4 bytes = tamanho int
```

//aloca espaço para 1 inteiro

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

//aloca espaço para um tipo struct estudante

```
struct estudante *e =
(struct estudante*)(malloc(sizeof(struct
estudante);
```

calloc ()

- Protótipo da função:

```
void * calloc(unsigned int num, unsigned int size);
```

Número de elementos a alocar

Tamanho de cada variável

Um ponteiro nulo (NULL) é retornado se a memória não for alocada.

calloc ()

- Alocação de memória para 10 inteiros;

  //aloca espaço para 10 inteiros (vetor)

```
int *p = (int *)calloc(10, sizeof(int));
if(!p){
   printf(“\nEspaço insuficiente”);
}
```

- Também funciona se fizer:

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

realloc ()

```
void * realloc(void *ptr, unsigned int num);
```

Quem será redimensionado

Quantidade de bytes a alocar

Um ponteiro nulo (NULL) é retornado se a memória não for redimensionada. Neste caso o bloco original é mantido.

realloc ()

- Realocação de memória: de 10 para 20 inteiros.

```
//Aloca espaço para 10 inteiros
int *p = (int *) malloc(10 * sizeof(int));
//Realoca espaço para 10 inteiros
int *pNew = realloc(p, 20 * sizeof(int));
if(pNew) {
  p = pNew;
} else {
  printf("\nMemória Insuficiente!\n");
  return 0;
}
```

free()

- Variáveis alocadas estaticamente desaparecem quando a função termina;
- Variáveis alocadas dinamicamente continuam a existir;
- Sempre que houver um **malloc()**, deve haver um **free()**;
- free(ptr);

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

//utiliza p no código...

free(p);
```

Exemplo alocação dinâmica


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

int main(){
   int *p; //cria ponteiro para um inteiro

   p = (int *)malloc(sizeof(int)); //aloca memória
   if(p){ //testa se memória foi alocada
      printf("Memória alocada com sucesso.\n");
   }else{
      printf("Não foi possivel alocar a memória.\n");
      return 0; //finaliza o programa
   }

   *p = 10; //atribui valor na memória alocada
   printf("Valor: %d\n\n", *p); //imprime o valor
   free(p); //libera a memória
}
```

Vetores e matrizes

Alocação dinâmica de vetores

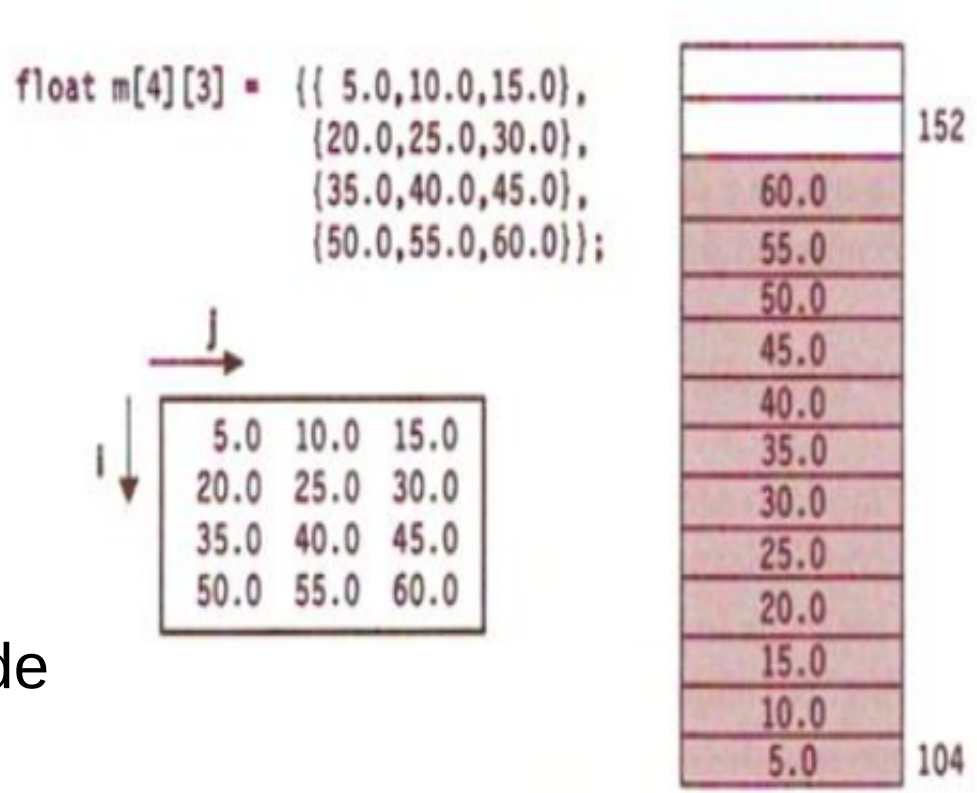

```
int tam = 10;
//Alocação do vetor
vetor = (int*) malloc(tam * sizeof(int));

//Uso do vetor (mesma forma do estático)
for(int i=0; i<tam; i++)
   vetor[i] = (i+1) * 10;

for(int i=0; i<tam; i++)
   printf("%d\n", vetor[i]);

free(vetor);
```

Vetores e matrizes

Matrizes: cada linha é um novo vetor

- A linguagem C permite a criação de vetores bidimensionais, declarados estaticamente;
  - Exemplo: float mat [4][3];

- Essa declaração reservar um espaço de memória para armazenar 12 elementos da matriz, que são armazenados de maneira contínua, organizados linha a linha.



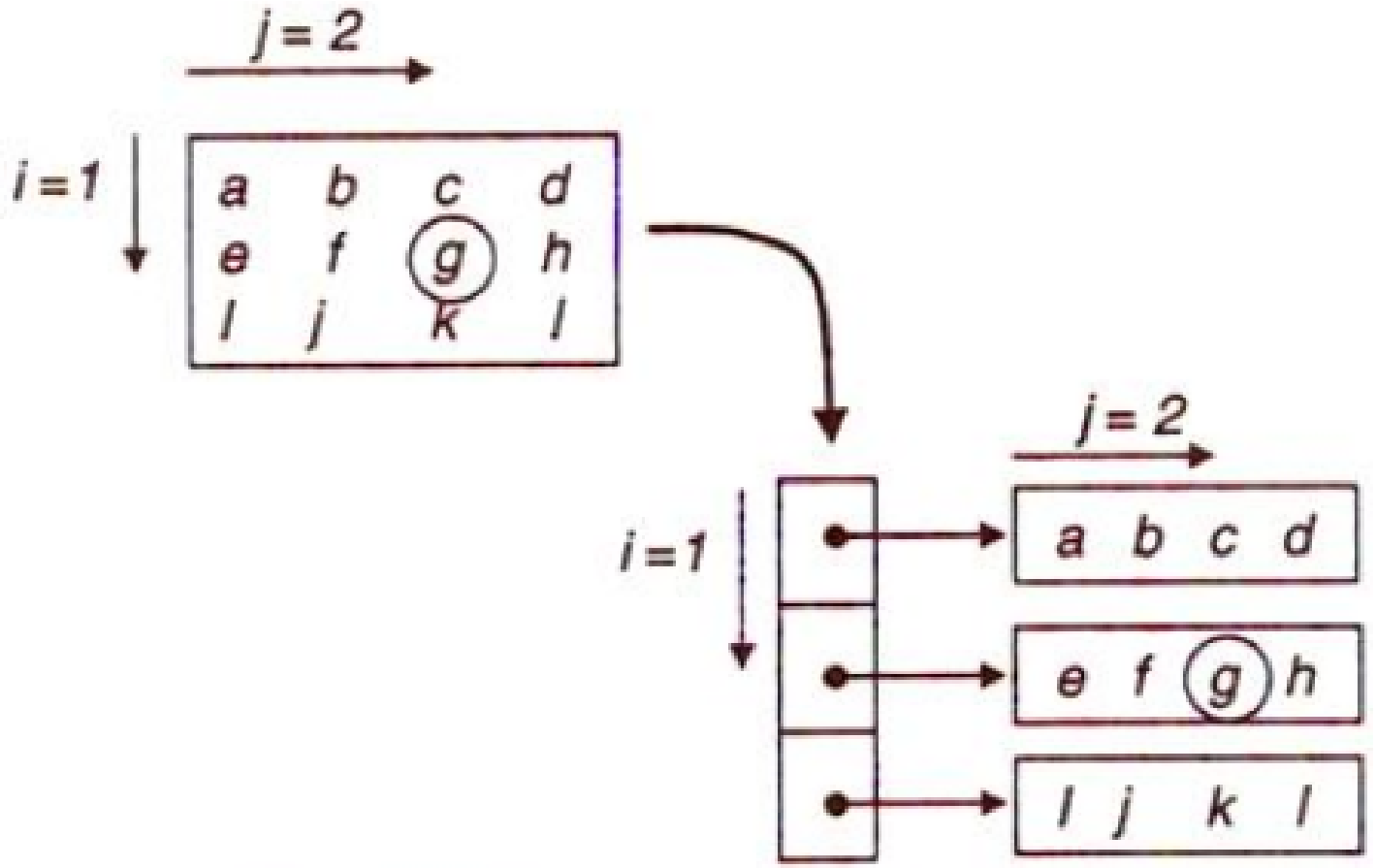
Vetores e matrizes

Matrizes: cada linha é um novo vetor


```
int lin = 4, col = 6;
int matriz[lin][col], *linha;

//Uso normal
for(int i=0; i<lin; i++)
   for(int j=0; j<col; j++)
      matriz[i][j] = ((i+1) * 10) + (j+1);

//Acesso às linhas como vetor
for(int i=0; i<lin; i++) {
   linha = matriz[i];
   for(int j=0; j<col; j++)
      printf("%d\t", linha[j]);

   printf("\n");
}
```

Vetores e matrizes

Matrizes alocadas dinamicamente (representação através de vetores de ponteiros)



Vetores e matrizes

Matrizes alocadas dinamicamente (representação através de vetores de ponteiros)

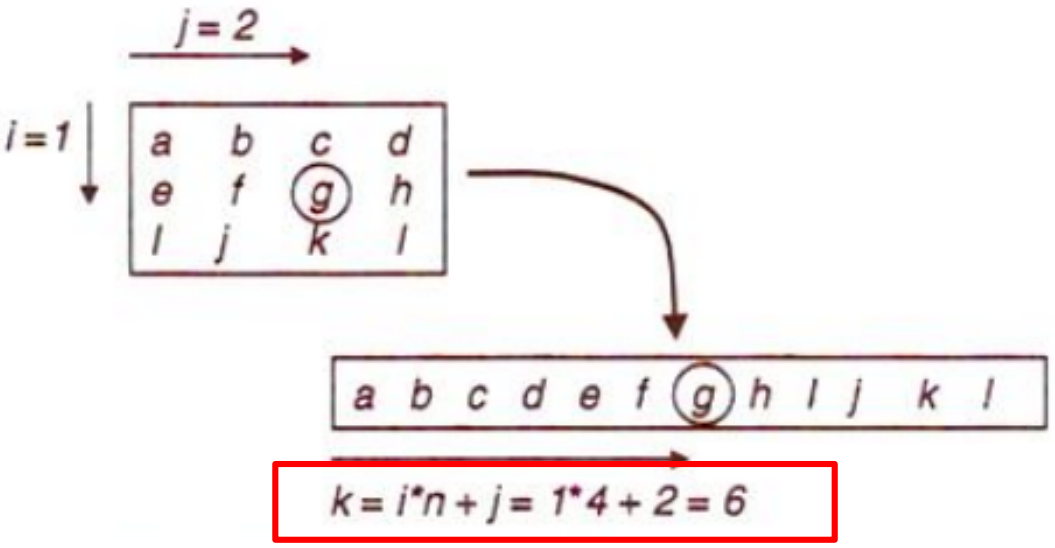

```
int lin = 4, col = 6, *linha;
int **matriz = (int**) malloc(lin * sizeof(int*));
for(int i=0; i<lin; i++)
   matriz[i] = (int*) malloc(col * sizeof(int));

for(int i=0; i<lin; i++) //Uso normal
   for(int j=0; j<col; j++)
      matriz[i][j] = ((i+1) * 10) + (j+1);

for(int i=0; i<lin; i++) { //Acesso às linhas como vetor
   linha = matriz[i];
   for(int j=0; j<col; j++)
      printf("%d\t", linha[j]);

   printf("\n");
}

for(int i=0; i<lin; i++)
   free(matriz[i]);
free(matriz);
```

Vetores e matrizes

Matrizes alocadas dinamicamente (representação através de vetor simples)

- Matrizes alocadas com um único vetor;
- Acesso através da expressão:

```
linha*qtd_colunas+coluna
```



Vetores e matrizes

Matrizes alocadas dinamicamente (representação através de vetor simples)


```
int lin = 4, col = 6;
int *matriz = (int*) malloc(lin * col * sizeof(int));

//Uso através da expressão = linha*qtd_colunas+coluna
for(int i=0; i<lin; i++)
   for(int j=0; j<col; j++)
      matriz[i*col+j] = ((i+1) * 10) + (j+1);

for(int i=0; i<lin; i++) {
   for(int j=0; j<col; j++)
      printf("%d\t", matriz[i*col+j]);

   printf("\n");
}

free(matriz);
```