Algoritmos e Programação B

Ponteiros

- Parte 3 –

Sumário

- Matrizes
- Alocação dinâmica de Memória
- Matrizes alocadas dinamicamente
  - Matrizes alocadas dinamicamente usando ponteiros para ponteiros

Matrizes

- Um elemento da matriz pode ser referenciado de duas formas:
  - por indexação: `a[0][2]`

    ou
  - por ponteiros: `*(a+2)`

- Isto porque uma matriz é alocada na memória de forma contígua.

Matrizes

- Por exemplo, a matriz

$$A = \begin{pmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{pmatrix}$$

na memória é alocada de forma contígua:

$$A = (a_{00}\ a_{01}\ a_{02}\ a_{10}\ a_{11}\ a_{12}\ a_{20}\ a_{21}\ a_{22})$$

onde 0 1 2 3 4 5 6 7 8

são os índices do vetor.

Matrizes

- Assim, para qualquer matriz bidimensional, referenciar o elemento

```
a[i][j]
```

é equivalente à

```
*(a + (i*Número_colunas+j))
```

Exemplo Matrizes e Funções: acesso por indexação

```
#include<stdio.h>
#define N 2
void lerMatriz(int a[N][N]){
     int i, j;

     for(i=0;i<N;i++)
        for(j=0;j<N;j++)
           scanf("%d", &a[i][j]);
     return;
}

int somatorio(int a[N][N]){
     int i, j, s=0;

     for(i=0;i<N;i++)
        for(j=0;j<N;j++)
           s +=a[i][j];
     return s;
}

void mult(int a[N][N], int s){
     int i, j;

     for(i=0;i<N;i++)
        for(j=0;j<N;j++)
           a[i][j] = s * a[i][j];
     return;
}
```

```
void mostrarMatriz(int a[N][N]){
     int i, j;

     for(i=0;i<N;i++){
        printf("\n");
        for(j=0;j<N;j++)
           printf("%d ", a[i][j]);
     }
     return;
}

int main(){
    int mat[N][N], x;

    lerMatriz(mat);
    x = somatorio(mat);
    mult(mat, x);
    mostrarMatriz(mat);

    system("pause");
    return;
}
```

Matrizes

- Em C, ponteiros e matrizes estão intimamente relacionados:
  - O nome de uma matriz sem o seu índice é um ponteiro para o primeiro elemento da matriz
  - Por exemplo, se for declarado `int a[3][3]`, então

`a` e **`&a[0][0]`**

são equivalentes, pois endereçam o primeiro elemento da matriz.

Funções e Matrizes

- Como estudado anteriormente, quando é necessário passar matrizes (vetor, matriz ou string) para uma função, estas **são sempre passadas por referência.**
- Há, basicamente, duas formas se ser passada uma matriz (de inteiros, float ou double) para uma função: o primeiro é usando indexação e o segundo usando ponteiros

```
//Situação 1
void leMatriz(int m[3][3]);

void main(void){
     int a[3][3];
     ...
     leMatriz(a);
     ...
}
```

```
//Situação 2
void leMatriz(int *m);

void main(void){
     int a[3][3];
     ...
     leMatriz(&a[0][0]);
     ...
}
```

Funções e Matrizes: uso de Indexação

```
#include<stdio.h>
void lerMatriz(int m[3][3]);
int maiorElemento(int m[3][3]);

int main(void){
     int a[3][3], i, j, maior;
     lerMatriz(a);
     maior=maiorElemento(a);
     printf("O maior elemento da matriz eh
%d\n", maior);
     return 0;
}

void  lerMatriz(int m[3][3]){
     int i, j;
     for(i=0;i<3;i++)
       for(j=0;j<3;j++){
           printf("a[%d][%d]: ", i, j);
           scanf("%d", &m[i][j]);
       }
}
```

```
int maiorElemento(int m[3][3]){
     int i, j, elementoMaior;
     elementoMaior=m[0][0];

     for(i=0;i<3;i++)
       for(j=0;j<3;j++){
           if (elementoMaior<m[i][j])
              elementoMaior=m[i][j];
       }

    return elementoMaior;
}
```

Funções e Matrizes: uso de aritmética de ponteiros

```
#include<stdio.h>
void lerMatriz(int *m);
int maiorElemento(int *m);

int main(void){
     int a[3][3], i, j, maior;
     lerMatriz(&a[0][0]);
     maior=maiorElemento(&a[0][0]);
     printf("O maior elemento da matriz eh %d\n",
maior);
     return 1;
}

void  lerMatriz(int *m){
     int i, j, d;
     for(i=0;i<3;i++)
       for(j=0;j<3;j++){
           d=i*3+j;
           printf("a[%d][%d]: ", i, j);
           scanf("%d", m+d);
       }
}
```

```
int maiorElemento(int *m){
     int i, j, elementoMaior, d;
     elementoMaior=*m;

     for(i=0;i<3;i++)
       for(j=0;j<3;j++){
           d=i*3+j;
           if (elementoMaior<*(m+d))
                 elementoMaior=*(m+d);
       }
     return elementoMaior;
}
```

Exemplo 2: Matriz e uso de aritmética de ponteiros

```
#include<stdio.h>
#define N 2
void lerMatriz(int *a){
     int i, j, k;
     for(i=0;i<N;i++)
        for(j=0;j<N;j++){
           k = i*N+j;
           scanf("%d", (a+k));
        }
     return;
}
int somatorio(int *a){
     int i, j, k, s=0;

     for(i=0;i<N;i++)
        for(j=0;j<N;j++){
           k = i*N+j;
           s += *(a+k);
        }
     return s;
}
void mult(int *a, int s){
     int i, j, k;
     for(i=0;i<N;i++)
        for(j=0;j<N;j++){
           k = i*N+j;
           *(a+k) = s * *(a+k);
           }
     return;
}
```

```
void mostrarMatriz(int *a){
     int i, j, k;

     for(i=0;i<N;i++){
        printf("\n");
        for(j=0;j<N;j++){
           k = i*N+j;
           printf("%d ", *(a+k));
        }
     }
     return;
}

int main(){
    int mat[N][N], x;
    lerMatriz(&mat[0][0]);
    x = somatorio(&mat[0][0]);
    mult(&mat[0][0], x);
    mostrarMatriz(&mat[0][0]);
    return 0;
}
```

Alocação Dinâmica de Memória

Alocação Dinâmica de Memória

- Alocação dinâmica é o meio pelo qual um programa pode obter memória enquanto está em execução
- **Variáveis globais** têm armazenamento alocado em tempo de compilação
- **Variáveis locais** usam a pilha
- Porém, em tempo de execução nenhum dos dois tipos de variáveis pode ser acrescentada... a não ser por alocação dinâmica de memória

Alocação Dinâmica de Memória

- Alguns programas podem precisar capacidade de armazenamento variável
  - Processador de texto
  - Banco de dados
  - Simulador

- Alocação dinâmica em C consiste em duas funções principais (existem outras também)
  - `malloc()`
    Aloca memória
  - `free()`
    Libera memória

Deve ser usado o cabeçalho **stdlib.h**

malloc

Protótipo

```
void *malloc(size_t número_de_bytes)
```

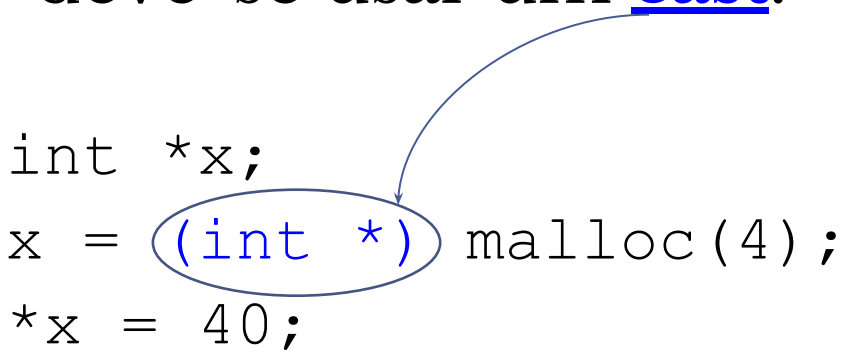
`número_de_bytes` é o número de bytes de memória que deseja-se alocar

Devolve um ponteiro do tipo `void` (pode ser atribuído a qualquer outro tipo de ponteiro)

- Se a alocação foi bem sucedida, malloc devolve um **ponteiro para o primeiro byte da região alocada**, caso contrário, retornará **NULL**

malloc

- O ponteiro retornado pela função `malloc` é do tipo void. Para assegurar-se da conversão do ponteiro ao tipo correto, deve-se usar um **cast**.



```
int *x;
x = (int *) malloc(4);
*x = 40;
```

malloc - exemplos

```
//Alocação de 1000 bytes
char *p;
p = malloc(1000)
```

```
//Alocação de 50 inteiros
int *x;
x = malloc(50*sizeof(int));
```

A memória alocada por essas funções é obtida no **heap**.

Exemplo de verificação de sucesso de alocação com malloc

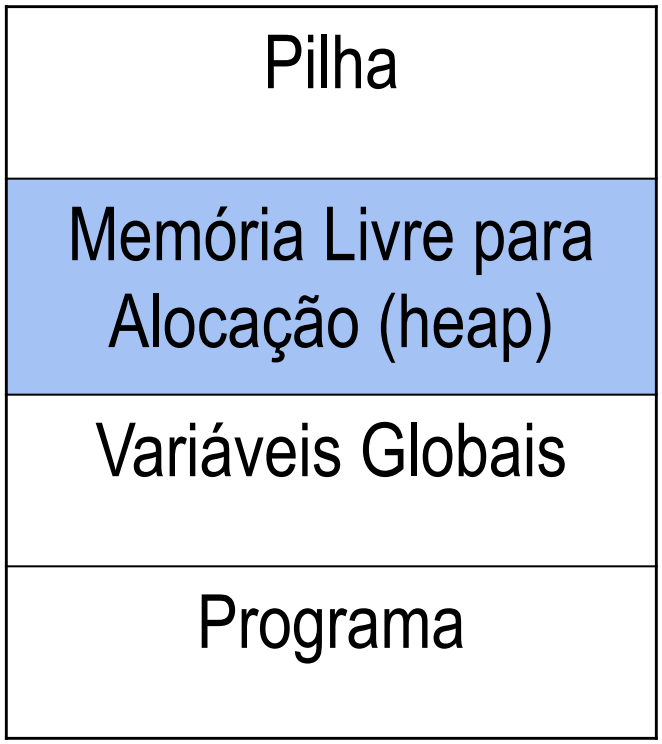
```
//Verificação do sucesso da alocação

int *y;

y = malloc(50 * sizeof(int));

if (!y)
{
   printf("Erro de alocação de memória ");
   exit(1);
}
```

Memória do Sistema



free()

- Devolve (libera) memória previamente alocada.

Protótipo

```
void free(void *p)
```

em que `p` é o ponteiro para a memória previamente alocada por `malloc()`

Nunca usar `free` com um argumento inválido, pois pode destruir a lista de memória livre.

Matrizes/vetores alocados dinamicamente

Matriz/vetor Dinâmico

- Usar `malloc` para alocar a quantidade de memória desejada
- A matriz é alocada de forma contígua na memória
- Dessa forma, deve-se trabalhar na memória como se ela fosse uma matriz
  - Qualquer ponteiro pode ser indexado como uma matriz unidimensional

Vetor alocado dinamicamente(exemplo)

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
main() {
   char *p;
   register int i;

   p = malloc(80);
   if (!p)  {
      printf("Erro alocação");
      exit(1);
      }

   gets(p);

   //inverte a ordem do string
   for(i=strlen(p)-1;i>=0;i--)
       putchar(*(p+i)); // ou putchar(p[i]);

   free(p);
}
```

Matriz Dinâmica Bidimensional
(exemplo)

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

void leMatriz(float *a, int l, int c);
void mostraMatriz(float *a, int l, int c);
int maiorElementoMatriz(float *a, int l, int c, int *posL, int *posC);
int main(void){
    int d, lin, col, pL, pC;
    float *m, maior;
    printf("Informe o número de linhas da matriz:");
    scanf("%d", &lin);
    printf("Informe o número de colunas da matriz:");
    scanf("%d", &col);
    m = (float *) malloc(lin*col*sizeof(float));
    if (!m){
       printf("Erro na alocacao dinamica de memória\n");
       exit(0);
    }
    leMatriz(m, lin, col);
    mostraMatriz(m, lin, col);
    maior=maiorElementoMatriz(m, lin, col, &pL, &pC);
    printf("\nO maior elemento da matriz eh %f e esta na linha %d e coluna %d\n", maior, pL, pC);
    free(m);
    return 1;
}
```

Matriz Dinâmica Bidimensional

(continuação exemplo)

```
void leMatriz(float *a, int l, int c){
     int i, j, d;
     for(i=0;i<l;i++)
      for(j=0;j<c;j++){
         d = i*c+j;
         printf("\na[%d][%d]:", i, j);
         scanf("%f", a+d);
      }
}

void mostraMatriz(float *a, int l, int c){
     int i, j, d;
     for(i=0;i<l;i++){
       printf("\n");
       for(j=0;j<c;j++){
          d = i*c+j;
          printf("%f  ", *(a+d));
       }
     }
}
```

Matriz Dinâmica Bidimensional

(continuação exemplo)

```
int maiorElementoMatriz(float *a, int l, int c, int *posL, int *posC){
   int i, j, d;
   float maior;

   maior = *a;

   for(i=0;i<l;i++)
      for(j=0;j<c;j++){
            d = i*c+j;
            if (maior<*(a+d)){
                maior=*(a+d);
                *posL=i;
                *posC=j;
                }
      }
    return maior;
}
```

Matrizes alocadas dinamicamente usando **ponteiros para ponteiros**

Matrizes alocadas dinamicamente

- Uma matriz pode ser alocada dinamicamente, utilizando **ponteiro para ponteiros**

- Assim, seus elementos podem ser facilmente referenciados por
  - indexação:

    ```
    a[i][j]
    ```

Matrizes alocadas dinamicamente

(exemplo)

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

int **MATRIXint (int r, int c, int val);

int main(void){
    int **a, i, j, lin=15, col=15;

    a = MATRIXint(lin, col, 23);

    for(i=0;i<lin;i++){
      printf("\n");
      for(j=0;j<col;j++)
         printf("%d   ", a[i][j]);
    }

    free(a);
    system("pause");
    return 1;
}
```

```
/* Funcao aloca uma matriz com linhas 0..r-1 e
colunas 0..c-1. Cada elemento da matriz recebe
valor val. */
int **MATRIXint (int r, int c, int val) {
   int i, j;
   int **m;

   m = malloc(r * sizeof(int *));

   if (!m){
     printf("Erro de alocação de memória!");
     exit(0);
   }

   for (i = 0; i < r; i++)
       m[i] = malloc(c * sizeof(int));

   for (i = 0; i < r; i++)
     for (j = 0; j < c; j++){
        m[i][j] = val;
        }
  return m;
}
```

Exercícios

Lista 7 – Ponteiros e Alocação Dinâmica de Memória