

Universidade Federal de Viçosa
Campus Rio Paranaíba
Instituto de Ciências Exatas e Tecnológicas

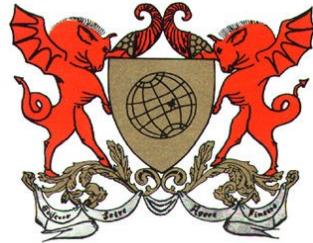
SIN 110

Programação

Curso de Sistemas de Informação

Prof. Rodrigo Smarzaro

smarzaro@ufv.br



Universidade Federal de Viçosa
Campus Rio Paranaíba
Instituto de Ciências Exatas e Tecnológicas

Aula de Hoje

Modularização (Funções) Parte 2

Passagem de Parâmetro por Valor

- Na passagem de parâmetro por valor a função chamada recebe uma **cópia dos valores** que são passados como argumentos.
- Dentro da função chamada são criadas outras variáveis **temporárias** para armazenar estes valores.



Exercício

Vamos definir uma função chamada **abs()** que aceita como parâmetro um valor x qualquer do tipo **float** e retorna o valor absoluto de x .

```
float abs( float x ) {  
    if (x < 0)  
        return -x;  
    else return x;  
}
```



Exercício

```
#include <stdio.h>
float abs( float x ) {
    if (x < 0)
        return -x;
    else return x;
}

int main() {
    float x1, x2;
    printf("Entre com o x1 e x2: ");
    scanf("%f %f", &x1, &x2);
    printf("|x1 - x2| = %f\n", abs( x1-x2 ) );
    printf("|x1| = %f\n", abs( x1 ) );
    printf("|x2| = %f\n", abs( x2 ) );
    return 0;
}
```



1. A expressão é avaliada (uma constante, uma variável ou uma expressão / fórmula);

```
#include <stdio.h>
float abs( float x ) {
    if (x < 0)
        return -x;
    else return x;
}

int main() {
    float x1, x2;
    printf("Entre com o x1 e x2: ");
    Scanf("%f %f", &x1, &x2);
    printf("|x1 - x2| = %f\n", abs( x1-x2 ) );
    printf("|x1| = %f\n", abs( x1 ) );
    printf("|x2| = %f\n", abs( x2 ) );
    return 0;
}
```



2. O valor-resultado é copiado para dentro do parâmetro da função, obedecendo a mesma ordem de chamada e de declaração;

```
#include <stdio.h>
float abs( float x ) {
    if (x < 0)
        return -x;
    else return x;
}

int main() {
    float x1, x2;
    printf("Entre com o x1 e x2: ");
    Scanf("%f %f", &x1, &x2);
    printf("|x1 - x2| = %f\n", abs( x1-x2 ) );
    printf("|x1| = %f\n", abs( x1 ) );
    printf("|x2| = %f\n", abs( x2 ) );
    return 0;
}
```

The diagram illustrates the flow of values between the main function and the abs function. A red arrow points from the parameter 'x' in the main function's call to the local variable 'x' in the abs function's definition. A red circle highlights the call to 'abs(x1-x2)' in the main function, with the number '1' next to it. The number '2' is placed near the abs function's definition.



3. A execução do programa é desviada para o início da função;

```
#include <stdio.h>
float abs( float x ) {
    if (x < 0)
        return -x;
    else return x;
}

int main() {
    float x1, x2;
    printf("Entre com o x1 e x2: ");
    Scanf("%f %f", &x1, &x2);
    printf("|x1 - x2| = %f\n", abs( x1-x2 ) );
    printf("|x1| = %f\n", abs( x1 ) );
    printf("|x2| = %f\n", abs( x2 ) );
    return 0;
}
```

3 → 2 → 1



4. Ao encontrar um comando **return**, o programa retorna a execução para o local de onde a função foi chamada, juntamente com o valor de retorno, se houver.

```
#include <stdio.h>
float abs( float x ) {
    if (x < 0)
        return -x;
    else return x;
}

int main() {
    float x1, x2;
    printf("Entre com o x1 e x2: ");
    Scanf("%f %f", &x1, &x2);
    printf("|x1 - x2| = %f\n", abs( x1-x2 ) );
    printf("|x1| = %f\n", abs( x1 ) );
    printf("|x2| = %f\n", abs( x2 ) );
    return 0;
}
```

The diagram illustrates the control flow in the program. Red arrows show the flow from the main loop back to the `abs` function at various points. Red numbers are placed near specific code elements to identify them:

- Number 1 is placed near the `abs` call in the `printf` statement: `printf("|x1 - x2| = %f\n", abs(x1-x2));`
- Number 2 is placed near the `abs` call in the first `printf` statement: `printf("|x1| = %f\n", abs(x1));`
- Number 3 is placed near the `if` condition in the `abs` function: `if (x < 0)`
- Number 4 is placed near the `else` branch of the `abs` function: `else return x;`



Outro exemplo clássico

```
1 #include <stdio.h>
2 #include <math.h>
3
4 void troca(int a, int b){
5     int c = a;
6     a = b;
7     b = c;
8 }
9
10 int main(){
11     int x = 5, y = 10;
12     troca(x, y);
13     printf("x:%d y:%d\n", x, y);
14     return 0;
15 }
```



O que o seguinte código faz?

```
1 #include <stdio.h>
2 #include <math.h>
3
4 int main()
5 {
6     int a, b;
7     a = 10;
8     b = 22;
9     printf("Valor de a: %d", a);
10    printf("\nValor de b: %d", b);
11    printf("\nEndereco da variavel a: %d", &a);
12    printf("\nEndereco da variavel b: %d", &b);
13    return 0;
14 }
```



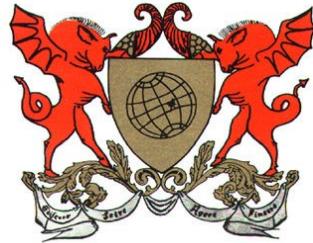
O que o seguinte código faz?

```
1 #include <stdio.h>
2 #include <math.h>
3
4 int main()
5 {
6     int a, b;
7     a = 10;
8     b = 22;
9     printf("Valor de a: %d", a);
10    printf("\nValor de b: %d", b);
11    printf("\nEndereço da variável a: %d", &a);
12    printf("\nEndereço da variável b: %d", &b);
13    return 0;
14 }
```

```
C:\Users\Rodrigo\OneDrive\Documentos\Untitled1.exe
Valor de a: 10
Valor de b: 22
Endereço da variável a: 6487580
Endereço da variável b: 6487576
```

Que valores
são esses?





Universidade Federal de Viçosa
Campus Rio Paranaíba
Instituto de Ciências Exatas e Tecnológicas

Ponteiros

Ponteiros

- Ponteiros são um dos recursos mais poderosos da linguagem C e de outras.
- Qualquer programa de utilidade prática escrito em C dificilmente dispensará o uso de ponteiros.
- A tentativa de evitá-los implicará quase sempre códigos maiores e de execução mais lenta.
- O conceito de ponteiro não é fácil; é preciso fazer algum esforço para dominá-lo.



Ponteiros

São muitas as aplicações de ponteiros:

- Acessar endereços de memória que o programa aloca em **tempo de execução**.
- Acessar variáveis que não são visíveis em uma função.
- Manipulação de arrays.
- Manipulação de strings.
- Passar o endereço de uma função para outra.
- Retornar mais de um valor para uma função.



Ponteiros Endereços

- A memória RAM de qualquer computador é uma sequência de **bytes**.
- Cada byte armazena um de 256 possíveis valores.
- Os bytes são numerados sequencialmente.
- O número de um byte é o seu endereço (= *address*).
- Cada objeto na memória do computador ocupa um certo número de bytes consecutivos.
- Um **char** ocupa 1 byte, um **int** ocupa 4 bytes (usualmente) e um **double** ocupa 8 bytes (usualmente).



Ponteiros Endereços

Cada objeto na memória tem um endereço. Na maioria dos computadores, o endereço de um objeto é o endereço do seu primeiro byte.

- Por exemplo, depois das declarações:

```
char c;
int i;
struct {
    int x, y;
} ponto;
int v[4];
```

os endereços
das variáveis
poderiam ser

c	89421
i	89422
ponto	89426
v[0]	89434
v[1]	89438
v[2]	89442



Ponteiros Endereços

- O endereço de uma variável é dado pelo operador &. **ATENÇÃO:** Não confunda esse uso de & com o operador lógico && (and).
- Se i é uma variável então &i é o seu endereço
No exemplo anterior, &i = 89422 e &v[3]=89446
- O segundo argumento da função scanf é o endereço da posição na memória onde devem ser depositados os objetos lidos do dispositivo padrão de entrada:

```
int i;  
scanf("%d", &i);
```



Ponteiros

- O que são ponteiros?

O ponteiro nada mais é do que uma variável que guarda o endereço de uma outra variável. (É um apontador para..)

- Como declarar um ponteiro? (usando o operador *****)

```
int *px; //px é um ponteiro para uma variável do tipo int
```

- Como atribuir um valor para um ponteiro?

```
int x = 5; //declara uma variável int chamada x
int *px; //declara um ponteiro para uma var. do tipo int
px = &x; //inicializa o ponteiro px com o endereço de x
*px = 10; //altera o valor da variável x
```



Ponteiros

- Depois que um ponteiro é declarado ele é sempre inicializado com valor **NULL**.
- **NULL** não é um endereço válido.
- Logo, você deve sempre inicializar um ponteiro com um endereço.



Ponteiros

- Há vários tipos de ponteiros:
 - Ponteiros para caracteres,
 - Ponteiros para inteiros,
 - Ponteiros para registros,
 - Até ponteiros para ponteiros!
- O computador faz questão de saber de que tipo de ponteiro você está falando.



Ponteiros

- Para declarar um ponteiro para inteiro:

```
int *pi;
```

- Para declarar um ponteiro p para um registro cel:

```
struct cel *pc;
```

- Para declarar um ponteiro para um ponteiro que aponta para um inteiro:

```
int **ppi;
```



Ponteiros

Suponha que `a`, `b` e `c` são variáveis inteiras. Eis um jeito bobo de fazer “`c = a+b`”:

```
int *p;          /* p é um ponteiro para um inteiro */
int *q;
p = &a;          /* o valor de p é o endereço de a */
q = &b;          /* q aponta para b */
c = *p + *q;
```

Outro exemplo bobo:

```
int *p;
int **r;          /* r é um ponteiro para um ponteiro para um inteiro */
p = &a;          /* p aponta para a */
r = &p;          /* r aponta para p e *r aponta para a */
c = **r + b;
```



Exercício

Suponha um programa com duas variáveis a e b com valores inteiros.

- 1) Declare dois ponteiros “pa” e “pb” do tipo inteiro
 - 2) Atribua para “pa” o endereço da variável a
 - 3) Atribua para “pb” o endereço da variável b
 - 4) Imprima os valores armazenados em “a” e “b” usando os ponteiros pa e pb
-



Exercício

Suponha um programa com duas variáveis a e b com valores inteiros.

```
int a = 10, b = 20;
```

1) Declare dois ponteiros “pa” e “pb” do tipo inteiro

```
int *pa, *pb;
```

2) Atribua para “pa” o endereço da variável a

```
pa = &a;
```

3) Atribua para “pb” o endereço da variável b

```
pb = &b;
```

4) Imprima os valores armazenados em “a” e “b” usando os ponteiros pa e pb

```
printf("a:%d e b:%d", *pa, *pb)
```



Passagem de parâmetro por Referência

- Na passagem por REFERÊNCIA é sempre passado para a função, o endereço onde o dado está armazenado na memória.
- Para isto os parâmetros devem ser declarados como PONTEIROS!
- Dessa forma, a função pode modificar o valor original do parâmetro independente de onde ele foi declarado.



Passagem de parâmetro por Referência

```
void soma_mais_um_valor (int n){  
    n = n + 1;  
} //valor
```



```
void soma_mais_um_referencia (int *n){  
    *n = *n + 1;  
} //referencia
```

por valor

por referência



Passagem de parâmetro por Referência

```
int main() {
    int x = 5;
    soma_mais_um_valor(x);
    printf("Depois da funcao: x = %d\n", x);
    int y = 5;
    soma_mais_um_referencia(&y);
    printf("Depois da funcao: y = %d\n", y);
    system("pause");
    return 0;
}
```



Passagem de parâmetro por Referência

- Como uma função pode alterar o valor das variáveis da função chamadora?
 - **Primeiro** : o programa chamador, em vez de passar **valores** para a função, passa seus **endereços**
 - **Segundo** : a função chamada deve declarar **ponteiros** para receber os endereços enviados por parâmetro.
 - **Terceiro** : deve-se utilizar o **operador &** na chamada da função.



Exemplo clássico

- Como fazer para a função trocar os valores das variáveis de fato?

```
void troca(int a, int b){  
    int c = a;  
    a = b;  
    b = c;  
}  
int main(){  
    int x = 5, y = 10;  
    troca(x, y);  
    printf("x:%d y:%d\n", x, y);  
    return 0;  
}
```



Exemplo clássico

Passagem por valor:

```
#include <stdio.h>
void swap(int a,int b) {
    int temp;
    temp = b;
    b = a;
    a = temp;
}
int main(void){
    int a,b;
    printf("Entre c/ valores:");
    scanf("%d %d",&a,&b);
    printf("a=%d b=%d \n",a,b);
    swap(a,b);
    printf("a=%d b=%d \n",a,b);
    return 0;
}
```

Digite dois valores: 5 6
a=5 e b=6
a=5 e b=6

Passagem por referência:

```
#include <stdio.h>
void swap(int *a,int *b) {
    int temp;
    temp = *b;
    *b = *a;
    *a = temp;
}
int main(void){
    int a,b;
    printf("Entre c/ valores:");
    scanf("%d %d",&a,&b);
    printf("a=%d b=%d \n",a,b);
    swap(&a,&b);
    printf("a=%d b=%d \n",a,b);
    return 0;
}
```

Digite dois valores: 5 6
a=5 e b=6
a=6 e b=5



Array como parâmetro de função

- Arrays são sempre passados “por referência” para uma função.
- Isso é feito pois evita a **cópia desnecessária** de grandes quantidades de dados para outras áreas de memória durante a chamada da função.
- E isso provavelmente afetaria o desempenho do programa.



Vetor como parâmetro de função

- Suponha que queiramos uma função para preencher o vetor notas. O vetor notas deve ser **criado na main e passado como parâmetro da função**.

```
int main()
{
    float notas[5];
    int i;
    for(i = 0; i < 5; i++)
    {
        printf("Digite a nota do aluno %d: ", i);
        scanf("%f", &notas[i]);
    }
    return 0;
}
```



Vetor como parâmetro de função

```
void preencheVetor(float *vet)
{
    int i;
    for(i = 0; i < 5; i++)
    {
        printf("Digite a nota : ");
        scanf("%f", &vet[i]);
    }
}
int main()
{
    float notas[5];
    preencheVetor(notas);
    return 0;
}
```



Vetor como parâmetro de função

```
void preencheVetor(float vet[])
{
    int i;
    for(i = 0; i < 5; i++)
    {
        printf("Digite a nota : ");
        scanf("%f", &vet[i]);
    }
}
int main()
{
    float notas[5];
    preencheVetor(notas);
    return 0;
}
```

PASSAGEM POR REFERÊNCIA

Logo, quando trabalhamos com vetores podemos usar tanto **float vet[]** quanto **float *vet**



Vetor como parâmetro de função

```
void preencheVetor(float *vet)
{
    int i;
    for(i = 0; i < 5; i++)
    {
        printf("Digite a nota : ");
        scanf("%f", &vet[i]);
    }
}
int main()
{
    float notas[5];
    preencheVetor(notas);
    return 0;
}
```

Faça uma função para imprimir o conteúdo do vetor notas.



Vetor como parâmetro de função

```
void imprimeVetor(float *vet)
{
    int i;
    for(i = 0; i < 5; i++)
        printf("%f ", vet[i]);
    printf("\n");
}
int main()
{
    float notas[5];
    preencheVetor(notas);
    imprimeVetor(notas);
    return 0;
}
```



Estruturas com Ponteiros

Quando criamos um ponteiro para uma estrutura,
surge um novo operador na linguagem C.

Operador ->



Estrutura com Ponteiro

```
typedef struct Shora{  
    int h;  
    int m;  
    int s;  
}Hora;  
int main(){  
    Hora agora;  
    Hora *pHora;  
    pHora = &agora;  
    pHora->h=20;//(*pHora).h = 20  
    pHora->m=35;//(*pHora).m = 35  
    pHora->s=45;//(*pHora).s = 45  
    printf("%d: %d: %d\n", agora.h, agora.m, agora.s);  
    system("pause");  
    return 0;  
}
```



Estrutura com Ponteiro

```
typedef struct Shora{  
    int h;  
    int m;  
    int s;  
}Hora;  
int main(){  
    Hora agora;  
    Hora *pHora;  
    pHora = &agora;  
    pHora->h=20;//(*pHora).h = 20  
    pHora->m=35;//(*pHora).m = 35  
    pHora->s=45;//(*pHora).s = 45  
    printf("%d: %d: %d\n", agora.h, agora.m, agora.s);  
    system("pause");  
    return 0;  
}
```



Estuturas com Ponteiros

- O operador **"."** (ponto) é utilizado para acessar membros em estruturas cuja variável não seja um ponteiro.
- O operador **"->"** (seta) é utilizado para acessar membros em estruturas cuja variável seja um ponteiro.



Estuturas com Ponteiros

- Quando trabalhamos com estruturas podemos passar para a função:
 - Um campo
 - Toda estrutura
- “por valor”
- “por referencia”



Estrutura como parâmetro de função (por Valor)

```
typedef struct sData
{
    int dia, mes, ano;
} Data;

void cadastrar(Data vNasc)
{
    printf("Digite o dia");
    scanf("%d", &vNasc.dia);

    printf("Digite o mês");
    scanf("%d", &vNasc.mes);

    printf("Digite o ano: ");
    scanf("%d", &vNasc.ano);
}
```

```
int main()
{
    Data nasc;
    cadastrar(nasc);
    return 0;
}
```

ATENÇÃO:

Na passagem por valor, os dados não são inseridos na variável **nasc** (**main**), e sim numa cópia dela;



Estrutura como parâmetro de função (por Referência)

```
typedef struct sData
{
    int dia, mes, ano;
} Data;

void cadastrar(Data *pNasc)
{
    printf("Digite o dia");
    scanf("%d", &pNasc->dia);

    printf("Digite o mês");
    scanf("%d", &pNasc->mes);

    printf("Digite o ano: ");
    scanf("%d", &pNasc->ano);
}
```

```
int main()
{
    Data nasc;
    cadastrar(&nasc);
    return 0;
}
```



Vetor do tipo estrutura como parâmetro de função (por Referência)

```
typedef struct sData
{
    int dia, mes, ano;
} Data;
void cadastrar(Data *pNasc)
{
    int i;
    for( i = 0; i < 3; i++)
    {
        printf("Digite o dia");
        scanf("%d", &pNasc[i].dia);
        printf("Digite o mês");
        scanf("%d", &pNasc[i].mes);
        printf("Digite o ano: ");
        scanf("%d", &pNasc[i].ano);
    }
}
```

```
int main()
{
    Data nasc[3];
    cadastrar(nasc);
    return 0;
}
```



Passando Matrizes para Função

- Quando passamos matrizes como parâmetros para função devemos pelo menos especificar o tamanho da segunda dimensão da matriz.

void funcao(int vetor[]) { ... }

void funcao(int vetor[5]) { ... }

void funcao(int *vetor) { ... }

void funcao(int matriz[][]){ ... } ERRADO!

void funcao(int matriz[][5]){ ... } CERTO!

void funcao(int **matriz) { ... } ERRADO!



Passando Matrizes para Função

```
int main(){

    int a[5][5];
    int i, j;

    for(i=0; i<5; i++)
        for(j=0; j<5; j++)
            a[i][j]=1;

    imprime(a, 5);

    return 0;
}
```



Passando Matrizes para Função

```
void imprime(int a[][] , int n) {  
    int i, j;  
    for(i=0; i<n; i++) {  
        for(j=0; j<n; j++)  
            printf("%d ", a[i][j]);  
        printf("\n");  
    }  
}
```

ATENÇÃO:

Na passagem de matrizes por referência, o número de colunas deve estar especificado no parâmetro;

```
int main(){  
  
    int a[5][5];  
    int i, j;  
    for(i=0; i<5; i++)  
        for(j=0; j<5; j++)  
            a[i][j]=1;  
    imprime(a, 5);  
    return 0;  
}
```



Exercício

- 1) Implemente uma função que receba como parâmetro um array de números reais (VET) de tamanho N e retorne quantos números negativos existem nesse array. Essa função deve obedecer ao cabeçalho: **int negativos(float *vet, int N);**
- 2) Implemente um programa que preenche dois vetores diferentes com número aleatórios (rand()) e que contenha 3 funções: (a) Retorna o maior elemento do vetor; (b) Retorna o menor elemento do vetor e (c) Retorna a média dos elementos do vetor.



Exercício 1

```
1 #include <stdio.h>
2
3 //Retorna a quantidade de numeros negativos
4 //em um vetor de N posicoes
5 int negativos(float vet*, int n){
6     int i, qtd = 0;
7     for(i=0; i<n; i++){
8         if(vet[i] < 0){
9             qtd++;
10        }
11    }
12    return qtd;
13}
14
15 int main(){
16
17     float vetor[100];
18     int i, n, neg;
19     printf("Qual o tamanho do vetor?\n");
20     scanf("%d", &n);
21
22     printf("Digite os elementos do vetor:\n");
23     for(i=0; i<n; i++){
24         scanf("%f", &vetor[i]);
25     }
26
27     neg = negativos(vetor, n);
28
29     printf("Existem %d numeros negativos no vetor.\n", neg);
30
31     system("pause");
32     return 0;
33}
34
```

Exercício 2

```
47 int main(){
48
49     int vetorA[100], vetorB[100];
50     int i, ta, tb; //Tamanho de A e B
51
52     printf("Qual o tamanho do vetor A?\n");
53     scanf("%d", &ta);
54     //Preenchendo o Vetor A com numeros aleatorios de 1 a 100
55     for(i=0; i<ta; i++){
56         vetorA[i] = rand()%100 + 1;
57     }
58     printf("Vetor A:\n");
59     imprimeVetor(vetorA, ta);
60
61     printf("Qual o tamanho do vetor B?\n");
62     scanf("%d", &tb);
63     //Preenchendo o Vetor B com numeros aleatorios de 1 a 1000
64     for(i=0; i<tb; i++){
65         vetorB[i] = rand()%1000 + 1;
66     }
67     printf("Vetor B:\n");
68     imprimeVetor(vetorB, tb);
69 }
```

```
70
71
72
73
74     int maiorA, menorA, maiorB, menorB;
75     float mediaA, mediaB;
76
77     printf("Informacoes do A:\n");
78     maiorA = maiorElemento(vetorA, ta);
79     menorA = menorElemento(vetorA, ta);
80     mediaA = mediaVetor(vetorA, ta);
81     printf("Maior elemento: %d\n", maiorA);
82     printf("Menor elemento: %d\n", menorA);
83     printf("Media dos elementos: %.2f\n", mediaA);
84
85
86
87
88
89
90
91 }
```

Como podem ser implementadas as funções destacadas?

Exercício

3) Implemente uma função que calcule as raízes de uma equação do segundo grau do tipo $Ax^2 + Bx + C = 0$. Lembrando que:

$$\frac{-B \pm \sqrt{\Delta}}{2A}$$

A variável A tem que ser diferente de zero.

- Se $\Delta < 0$ não existe real.
- Se $\Delta = 0$ existe uma raiz real.
- Se $\Delta \geq 0$ existem duas raízes reais.

$$\Delta = B^2 - 4AC$$

Essa função deve obedecer ao seguinte protótipo:

int raizes(float A,float B,float C,float * X1,float * X2);

Essa função deve ter como retorno o número de raízes reais e distintas da equação. Se existirem raízes reais, seus valores devem ser armazenados nas variáveis apontadas por X1 e X2.



Exercício 3

```
30 int main(){
31
32     float a, b, c;
33     printf("Digite os coeficientes A, B e C da equacao de 2o grau:\n");
34     scanf("%f %f %f", &a, &b, &c);
35
36     float x1, x2;//raizes
37     int num;//numero de raizes
38
39     num = raizes(a, b, c, &x1, &x2);
40
41     switch(num){
42         case -1: printf("Nao eh uma eq. de 2o grau!\n");
43                     break;
44         case 0:  printf("A eq. nao possui raizes reais!\n");
45                     break;
46         case 1:  printf("A raiz da equacao eh: %f\n", x1);
47                     break;
48         case 2:  printf("As raizes da equacao sao: %f e %f\n", x1, x2);
49                     break;
50     }
51
52
53     system("pause");
54     return 0;
55 }
```

Implementar a função raizes



Exercício 3

```
1 #include <stdio.h>
2 #include <math.h>
3 //Retorna a quantidade de raizes reais
4 //e se for > 0, volta por referencia
5 //na variaveis X1 e X2
6 int raizes(float A, float B, float C, float *X1, float *X2){
7     if(A == 0){//Nao e equacao de 2o grau
8         return -1;
9     }else{
10         float delta;
11         delta = B*B - 4*A*C;
12
13         if(delta < 0){           //Nao possui raizes reais
14             return 0;           //0 raizes
15         }else if(delta == 0){   //Possui uma raiz real
16             *X1 = (-B) / (2*A);
17             return 1;           // 1 raiz
18         }else{                 //Possui duas raizes reais
19             *X1 = ((-B) + sqrt(delta)) / (2*A);
20             *X2 = ((-B) - sqrt(delta)) / (2*A);
21             return 2;           //2 raizes
22     }
23 }
24 }
```

Exercício

- 4) Faça um programa com 3 funções (com matrizes como parâmetros): (Lembre-se que sempre serão por referência)
- Uma função para ler os dados de uma matriz
 - Uma função para imprimir os dados de uma matriz
 - Uma função para somar duas matrizes e gravar em uma terceira.



Exercício 4

```
36 int main(){
37
38     int mA[100][100], mB[100][100], mC[100][100], lin, col;
39
40     printf("Digite as dimensões das matrizes:\n");
41     scanf("%d %d", &lin, &col);
42
43     printf("Matriz A:\n");
44     lendoMatriz(mA, lin, col);
45
46     printf("Matriz B:\n");
47     lendoMatriz(mB, lin, col);
48
49     somaMatrizes(mA, mB, mC, lin, col);
50
51     printf("\n\nImprimindo...\n");
52     printf("Matriz A:\n");
53     imprimeMatriz(mA, lin, col);
54
55     printf("Matriz B:\n");
56     imprimeMatriz(mB, lin, col);
57
58     printf("Matriz C:\n");
59     imprimeMatriz(mC, lin, col);
60
61     return 0;
62 }
```

Criar as funções correspondentes

Exercício 4

```
1 #include <stdio.h>
2
3 //Passando matriz como parâmetro
4 //deve-se especificar sempre o numero
5 //maximo de colunas
6 void lendoMatriz(int mat[][100], int l, int c){
7     int i, j;
8     printf("Digite os elementos da matriz:\n");
9     for(i=0; i<l; i++){
10         for(j=0; j<c; j++){
11             printf("%d, %d): ", i, j);
12             scanf("%d", &mat[i][j]);
13         }
14     }
15 }
16
17 void imprimeMatriz(int mat[][100], int l, int c){
18     int i, j;
19     for(i=0; i<l; i++){
20         for(j=0; j<c; j++){
21             printf("%d ", mat[i][j]);
22         }
23         printf("\n");
24     }
25 }
26
27 void somaMatrizes(int A[][100], int B[][100], int C[][100], int l, int c){
28     int i, j;
29     for(i=0; i<l; i++){
30         for(j=0; j<c; j++){
31             C[i][j] = A[i][j] + B[i][j];
32         }
33     }
34 }
```

Exercício

5) Implemente uma função que receba um vetor de alunos (Nome, matrícula e Nota) e imprimir todas as informações do struct do aluno que possui a maior nota e todas as informações do aluno que possui a menor nota.



Exercício 5

```
1 #include <stdio.h>
2
3 typedef struct sAluno{
4     char nome[30];
5     int matricula;
6     float nota;
7 }Aluno;
```

struct Aluno

```
57 int main(){
58
59     Aluno vetor[100];
60     int n;
61     printf("Digite a quantidade de alunos:\n");
62     scanf("%d", &n);
63
64     cadastrAlunos(vetor, n);
65
66     informacoes(vetor, n);
67
68
69     system("pause");
70     return 0;
71 }
```

Função main

Exercício 5

```
9  void cadastraAlunos(Aluno *vet, int tam){  
10 }  
11  int i;  
12  for(i=0; i<tam; i++){  
13      printf("Informacoes do aluno %d:\n", i+1);  
14      printf("Nome: ");  
15      fflush(stdin);  
16      gets(vet[i].nome);  
17      printf("Matricula: ");  
18      scanf("%d", &vet[i].matricula);  
19      printf("Nota: ");  
20      scanf("%f", &vet[i].nota);  
21 }
```

exercício 5

```
29 //Funcao que descobre e imprime as informacoes
30 //do aluno com a maior nota e do aluno com a
31 //menor nota
32 void informacoes(Aluno *vet, int tam){
33     int i, i_maior, i_menor;
34     float maior, menor;
35     maior = vet[0].nota;
36     menor = vet[0].nota;
37     i_maior = 0;
38     i_menor = 0;
39     for(i=0; i<tam; i++){
40         if(vet[i].nota > maior){
41             maior = vet[i].nota;
42             i_maior = i;
43         }
44         if(vet[i].nota < menor){
45             menor = vet[i].nota;
46             i_menor = i;
47         }
48     }
49     printf("Maior nota:\n");
50     imprimeDados(vet, i_maior);
51     printf("\nMenor nota:\n");
52     imprimeDados(vet, i_menor);
53
54
55 }
```



exercício 5

```
29 //Funcao que descobre e imprime as informacoes
30 //do aluno com a maior nota e do aluno com a
31 //menor nota
32 void informacoes(Aluno *vet, int tam){
33     int i, i_maior, i_menor;
34     float maior, menor;
35     maior = vet[0].nota;
36     menor = vet[0].nota;
37     i_maior = 0;
38     i_menor = 0;
39     for(i=0; i<tam; i++){
40         if(vet[i].nota > maior){
41             maior = vet[i].nota;
42             i_maior = i;
43         }
44         if(vet[i].nota < menor){
45             menor = vet[i].nota;
46             i_menor = i;
47         }
48     }
49     printf("Maior nota:\n");
50     imprimeDados(vet, i_maior);
51     printf("\nMenor nota:\n");
52     imprimeDados(vet, i_menor);
53
54
55 }
```

```
23 void imprimeDados(Aluno *vet, int indice){
24     printf("Nome: %s\n", vet[indice].nome);
25     printf("Matricula: %d\n", vet[indice].matricula);
26     printf("Nota: %.2f\n", vet[indice].nota);
27 }
```

