

Estruturas de Dados I

Funções e Procedimentos

UNICID - 2016

- Por que usar funções?
  - Evita escrita repetida de código (uma certa seqüência de comando deve ser repetida em vários lugares de um programa).
  - Economiza o tempo gasto com o trabalho de copiar estas seqüências;
  - Evitar a necessidade de mudar em múltiplos lugares caso deseje alterar o seu funcionamento;
  - Dividir grandes tarefas de computação em tarefas menores;
  - Facilita o gerenciamento de grandes sistemas e
  - Aumenta a confiabilidade dos mesmos.
- 
- Resumo: Principais motivações para uso das funções!
  - Evitar repetição de código
  - Modularização (divide em pequenos subprogramas ou subalgoritmos).

- Funções em "C"

Em "C", todo programa é composto por funções;

Já utilizamos muitas funções, mesmo sem saber que eram funções (printf, scanf, sqrt, e até a famosa função "main"...) main()  
{  
  
}

printf("TEXTO\_A\_IMP");  $\text{pow}(x, y) \Leftrightarrow x^y$   
#include <stdio.h>  
int main() void main(void) float  
double  
}  
}

int, char, float, ....

**Formato de declaração de funções :**

```

# include <stdio.h>
tipo_de_retorno nome_da_função (tipo1 param1, tipo2 param2, ..., tipoN paramN)
{
    /* corpo da função */
    return valor_de_retorno;
} /* fim da função */
  
```

regra p/ nomear variáveis

variáveis locais

```

int soma(int n1, int n2)
{
    return n1+n2;
}
  
```

- tipo\_de\_retorno especifica o tipo do valor que será retornado para quem chamou a função (int, float, double, char, void).

- Se o tipo\_de\_retorno for void significa que se trata de uma função que se comporta como uma subrotina; ou seja, a função não necessita retornar nenhum valor (exemplo: printf)

→ Função Principal

```

main()
{
    int x, y;
    scanf("%d", &x, &y);
    printf("%d + %d = %d", x, y, soma(x, y));
    getch();
}
  
```

variáveis globais

```

int soma(int n1, int n2)
{
    return n1+n2;
}
  
```

### Definição

Conjunto de comandos agrupados em um bloco que recebe um nome e através deste pode ser ativado.

### Por que usar funções ?

- Para permitir o reaproveitamento de código já construído (por você ou por outros programadores);
- Para evitar que um trecho de código que seja repetido várias vezes dentro de um mesmo programa;
- Para permitir a alteração de um trecho de código de uma forma mais rápida. Com o uso de uma função é preciso alterar apenas dentro da função que se deseja;
- Para que os blocos do programa não fiquem grandes demais e, por consequência, mais difíceis de entender;
- Para facilitar a leitura do programa-fonte de uma forma mais fácil;
- Para separar o programa em partes (blocos) que possam ser logicamente compreendidos de forma isolada.

A `Lista_de_Parametros`, também é chamada de `Lista_de_Argumentos`, é opcional.

ditional.

~~void~~ main(~~void~~)      main()

```
int main()  
{  
  
}
```

## Parâmetros

A fim de tornar mais amplo o uso de uma função, a linguagem C permite o uso de parâmetros.

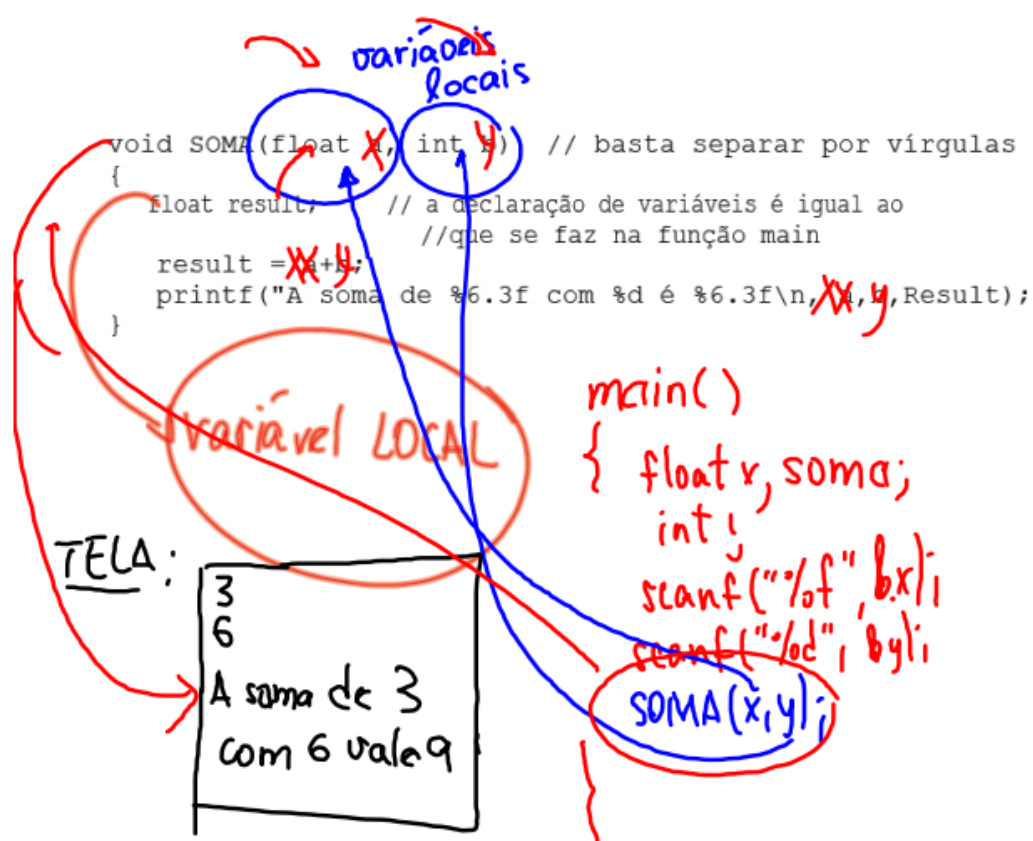
```
int funcao_louca(float a, char c, int d)
```

Estes parâmetros possibilitam que se defina sobre quais dados a função deve operar.

Para definir os parâmetros de uma função o programador deve explicitá-los como se estivesse declarando uma variável, entre os parênteses do cabeçalho da função. ~~Caso precise declarar mais de um parâmetro, basta separá-los por vírgulas.~~

No exemplo a seguir temos a função SOMA que possui dois parâmetros, sendo o primeiro um float e o segundo um int.

Os parâmetros da função na sua declaração são chamados **parâmetros formais**. Na chamada da função os parâmetros são chamados **parâmetros atuais**.





Os parâmetros são passados para uma função de acordo com a sua posição.

```
int subtrai(int x, int y)
{
    return x-y;
}
```

Ou seja, o primeiro parâmetro atual(da chamada) define o valor o primeiro parâmetro formal (na definição da função, o segundo parâmetro atual define o valor do segundo parâmetro formal e assim por diante.

Os nomes dos parâmetros na chamada não tem relação com os nomes dos parâmetros na definição da função.

```
main
scanf("%d %d", &a, &b);
subtrai(b, a);
return b-a;
```

No código a seguir, por exemplo, a função SOMA é chamada recebendo como parâmetros as variáveis "b" e "a", nesta ordem.

Funções sem Passagem de Parâmetros e sem retorno

São o tipo mais simples de função que não recebe nenhuma informação no momento de sua chamada e que também não recebe nenhum valor para quem a chamou:

```
#include <stdio.h>
void soma() {
    int a,b,soma;
    printf("\nDigite o 1o. numero: ");
    scanf("%d",&a);
    printf("\nDigite o 2o. numero: ");
    scanf("%d",&b);
    soma = a + b;
    printf("\n\n%d + %d = %d",a,b,soma);
    getch();
}
main()
{
    soma();
}
```

→ procedimentos

### Funções com Passagem de Parâmetros e sem retorno

É representado por aquelas que recebem valores no momento em que são chamadas (parâmetros), mas que, no final, não devolvem valor para quem as chamou (retorno)

```
#include <stdio.h>
void calcula_media(float numero1, float numero2){
    float media;
    media = (numero1+numero2)/2;
    printf("\n\nMedia = %.1f",media);
    getch();
}
main()
{
    float n1,n2;
    printf("\nDigite o 1o. numero: ");
    scanf("%f",&n1);
    printf("\nDigite o 2o. numero: ");
    scanf("%f",&n2);
    calcula_media(n1,n2);
}
```

Funções sem Passagem de Parâmetros e com retorno → return

Este tipo de função é representado por aquelas que não recebem valores no momento em que são chamadas (parâmetros), mas que, no final, devolvem um valor para quem as chamou (retorno).

```
#include <stdio.h>

float multiplica(){
    float a,b,produto;
    printf("\nDigite o 1o. numero: ");
    scanf("%f",&a);
    printf("\nDigite o 2o. numero: ");
    scanf("%f",&b);
    produto = a * b;
    return produto;
}

main()
{
    float resposta;
    resposta = multiplica();
    printf("\n\nO produto vale: %.1f",resposta);
    getch();
}
```

Funções com Passagem de Parâmetros e com retorno

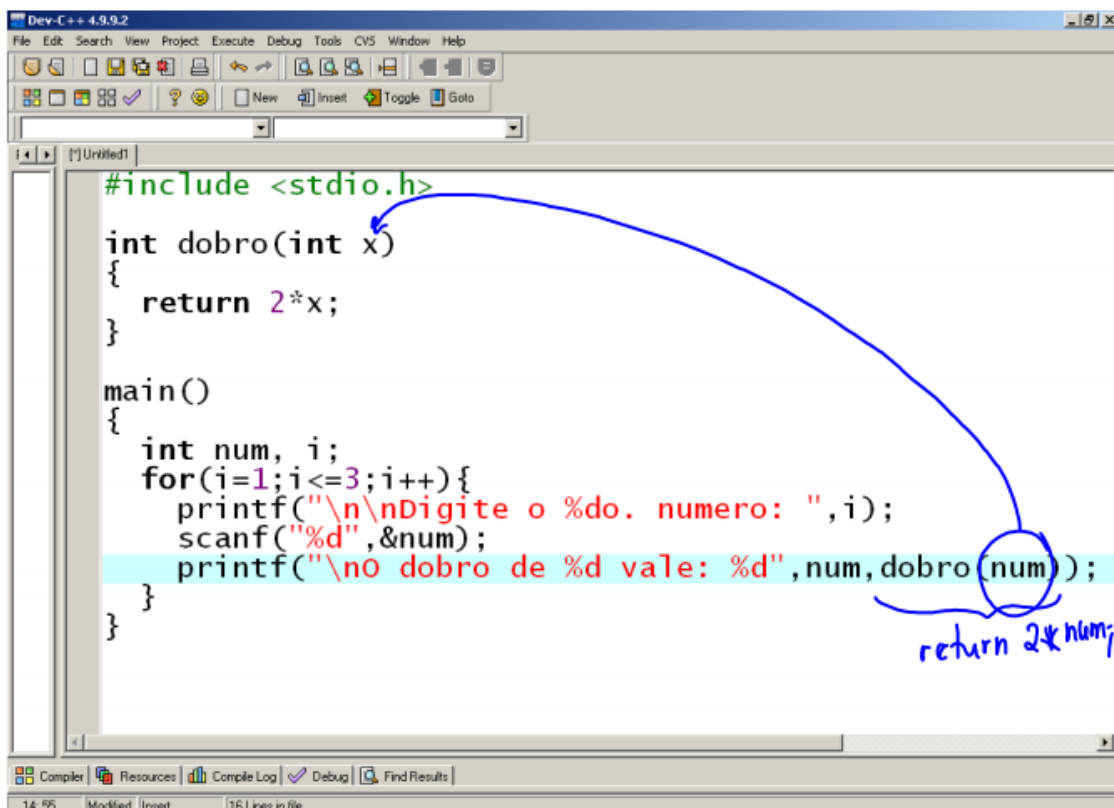
São aquelas funções que recebem valores no momento em que são chamadas (parâmetros) e que, no final, devolvem um valor para quem as chamou (retorno)

The diagram illustrates the flow of data between the `divisao` function and the `main` function. Red handwritten notes label the `float` type as "tipo" and the parameters `dividendo` and `divisor` as "parâmetros". Blue arrows show the return value `q` being passed from the function back to the caller. Green arrows show the arguments `n1` and `n2` being passed from the caller to the function. A black arrow shows the return value being assigned to the `resposta` variable in `main`.

```
#include <stdio.h>
float divisao(float dividendo, float divisor){
    float q;
    q = dividendo / divisor;
    return q;
}
void main()
{
    float n1, n2, resposta;
    printf("\nDigite o 1o. numero: ");
    scanf("%f", &n1);
    printf("\nDigite o 2o. numero: ");
    scanf("%f", &n2);
    resposta = divisao(n1, n2);
    printf("\nn1: %.1f / %.1f = %.2f", n1, n2, resposta);
    getch();
}
```

### EXERCÍCIOS:

- ① Crie uma função em C, que possa entrar com 3 números <sup>inteiros</sup> e, para cada um, imprimir o dobro de cada um deles.
- ② Crie uma função em C, que leia três (3) notas de um aluno e calcule a sua média aritmética.



The screenshot shows the Dev-C++ 4.9.9.2 IDE with a C program. The code is as follows:

```
#include <stdio.h>

int dobro(int x)
{
    return 2*x;
}

main()
{
    int num, i;
    for(i=1; i<=3; i++){
        printf("\n\nDigite o %do. numero: ", i);
        scanf("%d", &num);
        printf("\no dobro de %d vale: %d", num, dobro(num));
    }
}
```

Handwritten annotations in blue ink include:

- A curved arrow pointing from the `dobro(num)` argument in the `printf` statement to the `return 2*x;` line in the `dobro` function.
- A circle around the `dobro(num)` argument, with the handwritten text `return 2*num;` written below it.

The status bar at the bottom indicates the file has 16 lines and the cursor is at line 14, column 55.

```
#include <stdio.h>
#include <conio.h>
main()
{
    int num, cont;
    printf("Funcao que dobra 3 numeros\n\n");
    for(cont=1;cont<=3;cont++){
        printf("\n\nDigite um numero inteiro: ");
        scanf("%d",&num);
        printf("\nO dobro do numero vale: %d",dobro(num));
    }
    getch();
}

int dobro(int x)//x é variável local
{
    return (2*x);
}
```



```
Dev-C++ 4.9.9.2
File Edit Search View Project Execute Debug Tools CVS Window Help

float media(float nota1, float nota2, float nota3)
{
    return (nota1 + nota2 + nota3)/3;
}

main()
{
    float n1,n2,n3;
    printf("MEDIA DE 3 NOTAS:\n\n");
    printf("\nDigite a 1a nota: ");
    scanf("%f",&n1);
    printf("\nDigite a 2a nota: ");
    scanf("%f",&n2);
    printf("\nDigite a 3a nota: ");
    scanf("%f",&n3);
    system("CLS");
    printf("Media das 3 notas: %.1f",media(n1,n2,n3));
    getch();
}
```

- ③ Crie uma função que imprima o maior número, entre dois números inteiros.
- ④ Crie uma função que retorne 1 se o valor digitado for positivo ou 0 se for negativo.
- ⑤ Crie uma função em C que converta um ângulo em graus, para radianos.

$$\pi \text{ rad} = 180^\circ \Rightarrow \text{Arad} = (\text{Agraus} * \pi) / 180;$$

$\text{Arad} = \text{Agraus}$  #define pi 3.14

```
#include <stdio.h>
int maior(int num1,int num2){
    if(num1 >= num2)
        return num1;
    else
        return num2;
}
main()
{
    int n1,n2;
    printf("Digite o 1o. numero: ");
    scanf("%d",&n1);
    printf("\n\nDigite o 2o. numero: ");
    scanf("%d",&n2);
    printf("\n\nMaior numero = %d",maior(n1,n2));
    getch();
}
```

```
#include <stdio.h>
int posneg(int num){
    if(num >= 0)
        return 1;
    else
        return 0;
}
main()
{
    int n;
    printf("POSITIVO OU NEGATIVO \n\n");
    printf("Digite um no. inteiro: ");
    scanf("%i",&n);
    printf("\n\n%d ",posneg(n));
    printf("\n1- Positivo | 0 - Negativo");
    getch();
}
```

```
#include <stdio.h>
#define pi 3.14
float conv_a_rad(float graus){
    return (graus * pi)/180;
}
main()
{
    float g;
    printf("CONVERSAO DE GRAUS A RADIADOS\n\n");
    printf("Digite um angulo, em graus: ");
    scanf("%f",&g);
    printf("\n\n%.1f graus = %.3f rad",g,conv_a_rad(g));
    getch();
}
```