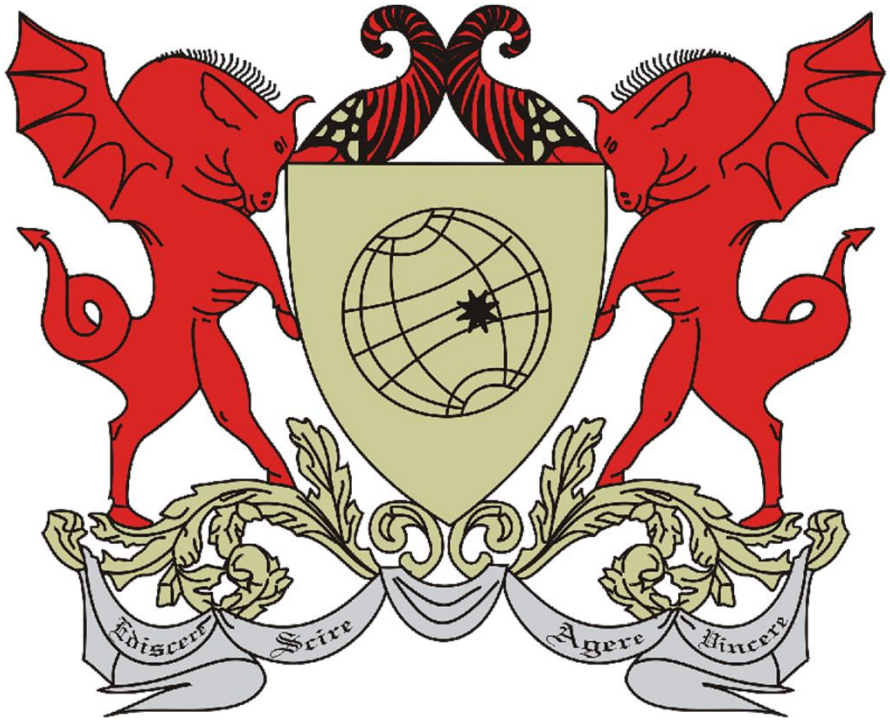


# CCF 110 – Programação

Aula 08 – Funções / Linguagem C

Prof. José Augusto Nacif – jnacif@ufv.br

# Funções

- Objetivo
  - Facilitar a solução de problemas complexos.

"A arte de programar consiste na arte de organizar e dominar a complexidade dos sistemas"

Dijkstra, 1972

# Funções

- Estratégia
  - Dividir para conquistar

- Divisão de um problema original em subproblemas (módulos) mais fáceis de resolver e transformáveis em trechos mais simples, com poucos comandos (subprogramas ou funções)

# Funções

- Trechos de código independentes
  - Estrutura semelhante àquela de programas
  - Executados somente quando chamados por outro(s) trecho(s) de código
- Devem executar uma tarefa específica
- Ativação de uma função
  - Fluxo de execução desloca-se do fluxo principal para a função
  - Execução da função é concluída
  - Fluxo de execução retorna ao ponto imediatamente após onde ocorreu a chamada da função.

# Vantagens de utilização de funções

- Maior controle sobre a complexidade.
- Estrutura lógica mais clara.
- Maior facilidade de depuração e teste, já que funções podem ser testados separadamente.
- Possibilidade de reutilização de código.

# Funções em Linguagem C

- Segmentos de programa que executam uma tarefa específica
- Essência da programação estruturada.
  - Ex: sqrt(), strlen(), etc.
- O programador também pode escrever suas próprias funções, chamadas de funções de usuário, que tem uma estrutura muito semelhante a um programa.

# Forma geral da declaração de uma função

```
tipo_da_funcao  nome_da_função (lista_de_parâmetros){
      //declarações locais
      //comandos
}
```

- tipo_da_funcao
  - Tipo de valor retornado pela função.
  - Se não especificado, é considerado como retornando um inteiro.
- nome_da_função
  - Nome da função conforme as regras do C
- lista_de_parâmetros
  - Tipo de cada parâmetro seguido de seu identificador, com vírgulas entre cada parâmetro.

# Exemplos de cabeçalhos de funções

- `soma_valores (int valor1, int valor2)`
- `void imprime_linhas(int num_lin)`
- `void  apresenta_menu(  )`
- `float conv_dolar_para_reais(float dolar)`

# Funções void

- Void é um termo que indica ausência.
- Em linguagem C é um tipo de dados.

# Exemplo de função void – escreveint versão 1

```
//Escrita de numeros inteiros
#include<stdio.h>
#include <stdlib.h>
main( )
{  int i;
   for (i=1;i<20;i++) //apresentacao do cabecalho
        printf("*");
   printf("\n");
   printf("Numeros entre 1 e 5\n");
   for (i=1;i<20;i++)
        printf("*");
   printf("\n");
   for (i=1;i<=5;i++) //escrita dos numeros
        printf("%d\n",i);
   for (i=1;i<20;i++)
        printf("*");
   printf("\n");
   system("pause");
}
```

# Exemplo de função void – escreveint versão 1

- A repetição de trechos de código idênticos
  - Fácil e rápido.
  - Tende a produzir erros.
  - Manutenção/alteração mais trabalhosa e sujeita a erros.
    - Alterações de trechos iguais não são realizadas em todas as ocorrências
- A solução para esta questão são as funções.
- A seguir uma versão do programa `escreveint` com a função `apresente_linha`.

# Exemplo de função void – escreveint versão 2

```
//escrita de numeros inteiros
#include<stdio.h>
#include <stdlib.h>
void apresente_linha(void);
main( )
{  int i;
   apresente_linha( ); //apresentacao do cabecalho
   printf("Numeros entre 1 e 5\n");
   apresente_linha( ); // Escrita dos numeros
   for (i=1;i<=5;i++)
        printf("%d\n",i);
   apresente_linha( );
   system("pause");
}
```

```
void apresente_linha (void)
 {
  int i;
  for (i=1;i<20;i++)
       printf("*");
  printf("\n");
 }
```

# Cabeçalho da função apresente_linha

```
void apresente_linha (void)
```

Indica que a função não retorna valor no seu nome.

Indica que a função não tem parâmetros.

- A função `apresente_linha` realiza sua tarefa sem receber nenhum valor do mundo externo à função, via parâmetros, e sem retornar nenhum valor no seu nome.
  - Seu tipo é void e seus parâmetros são void.

# Execução de uma função

- Ao ser encontrada uma chamada de uma função
- Execução é desviada para o trecho de código da função.
- A função é ativada
- Itens locais à função são criados
- A função é executada
- Concluída a execução, todos os elementos locais são destruídos
- Execução retorna ao fluxo principal, ao ponto imediatamente seguinte àquele no qual ocorreu a chamada da função.

# Variáveis locais

- Os parâmetros que aparecem no cabeçalho das funções e as variáveis e constantes declaradas internamente a funções são locais à função.
  - Na função apresente_linha, o i é uma variável local a essa função.

```
void apresente_ linha (void)
 {
   int i;
   for (i=1;i<20;i++)
        printf("*");
   printf("\n");
 }
```

# Variáveis e constantes locais:

- Recomenda-se fazer todas as declarações de uma função no seu início.
- As variáveis e constantes declaradas em uma função são ditas locais à função porque:
  - Só podem ser referenciadas por comandos que estão dentro da função em que foram declaradas;
  - Existem apenas enquanto a função em que foram declaradas está sendo executada.
    - São criadas quando a função é ativada e são destruídas quando a função encerra.

# Funções tipo void

- São ativadas como se fossem comandos.
- Não ocorrem dentro de  expressões.
- Correspondem aos procedimentos de outras linguagens (Pascal, etc.).

# Funções com passagem de parâmetros

- sqrt: função pré-definida
  - A seguir um programa que extrai a raiz quadrada de um número indeterminado de valores informados.
  - Para extrair a raiz quadrada dos valores é usada a função pré-definida sqrt, da biblioteca math.h.
  - A função sqrt é do tipo double, isso significa que quando ela é chamada, no lugar de sua chamada retorna um valor double.
  - Para executar essa função é necessário fornecer um parâmetro, o valor para o qual se deseja que a raiz quadrada seja calculada.
  - No exemplo, está armazenado na variável valor.

# Exemplo sqrt: função pré-definida

```
//extrai a raiz quadrada de valores informados
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
main (  )
{
   int seguir;
   double valor;
   do
     {
       printf("\nValor para extrair raiz: ");
       scanf("%lf", &valor);
       printf ("\nRaiz quadrada de %6.2lf = %6.2lf\n",valor, sqrt(valor));
       printf("\nMais um valor, digite 1, para parar, digite 0: ");
       scanf("%d", &seguir);
     }
   while (seguir);
   system("pause");
}
```

# Exemplo calc_produto: função definida pelo usuário

- A seguir um programa que calcula o produto de um número indeterminado de pares de valores informados.
- Para calcular os produtos é usada a função definida pelo usuário calc_produto
- A função calc_produto é do tipo int, isso significa que quando ela é chamada, no lugar de sua chamada retorna um valor int.
- Para executar essa função é necessário fornecer dois parâmetros, os dois valores para cálculo do produto, oper1 e oper2.

# produto: função definida pelo usuário

```
//calcula produtos de pares de valores informados
#include <stdio.h>
#include <stdlib.h>
int calc_produto(int, int);
main ( )
{
    int seguir;
    int oper1, oper2, produto;
    do
     {
       printf("\nOperando 1: ");
       scanf("%d", &oper1);
       printf("\nOperando 2: ");
       scanf("%d", &oper2);
       printf ("\nProduto = %d\n", calc_produto(oper1, oper2));
       printf("\nPara continuar, digite 1, para parar, digite 0: ");
       scanf("%d", &seguir);
     }
    while (seguir);
    system("pause");
}
```

```
int calc_produto(int valor1, int valor2)
 {
    return valor1 * valor2;
 }
```

# Comando return(): retorno de valor e fim lógico da função

- O comando return atribui valor a função.
- Ao ser executado, encerra a execução da função.
- Se uma função é declarada com tipo diferente de void (int, char, float, etc.) significa que ela pretende explorar a possibilidade de retorno de um valor em seu nome, e então pode ser usada em expressões.

# Perguntas

- Quando uma função encerra sua execução?
  - Uma função encerra sua execução quando:
    - O fim do seu código é atingido;

      ou

    - Um comando return é encontrado e executado.
- Vários comandos return podem existir em uma função?
  - Sim, embora não seja recomendável.
  - Princípios da programação estruturada
    - Cada função tem um único ponto de entrada e um único ponto de saída.
  - Se vários returns existirem em uma função, tem-se múltiplos pontos de saída possíveis.
  - Mas a função só conclui quando o primeiro return é ativado.

# Observações

- As funções devem ser declaradas de modo a serem o mais independentes possível do mundo externo a elas.

- Passagem de parâmetro por valor: os parâmetros de chamada e os parâmetros formais (da declaração da função) só se conectam no momento da chamada da função e então o que há é apenas a transferência de valores entre os parâmetros respectivos.

# Exemplo 1 de passagem de parâmetro por valor

```
#include <stdio.h>
#include <stdlib.h>
void soma_dez_a_valor(int);
main (  ) {
    int valor;
    printf("\nValor inteiro: ");
    scanf("%d", &valor);
    printf("\nNa Main: valor antes da chamada da funcao: %d\n",       valor);
    soma_dez_a_valor(valor);
    printf("\nNa Main: valor apos chamada da funcao: %d\n", valor);
    system("pause");
}
void soma_dez_a_valor(int valor) {
    valor = valor + 10;
    printf("\nNa Funcao: valor dentro da funcao: %d\n", valor);
 }
```

# Exemplo 2 de passagem de parâmetro por valor

```
#include <stdio.h>
#include <stdlib.h>
void soma_dez_a_valor(int);
main (  ) {
    int valor;
    printf("\nValor inteiro: ");
    scanf("%d", &valor);
    printf("\nNa Main: valor antes da chamada da funcao: %d\n",       valor);
    soma_dez_a_valor(valor);
    printf("\nNa Main: valor apos chamada da funcao: %d\n", valor);
    system("pause");
}
void soma_dez_a_valor(int num) {
    num = num + 10;
    printf("\nNa Funcao: valor dentro da funcao: %d\n", num);
 }
```

# Parâmetros de funções

- Reforçando
  - Os nomes das variáveis declaradas no cabeçalho de uma função são independentes dos nomes das variáveis usadas para chamar a mesma função.
  - As declarações de uma função são locais a essa função. Os parâmetros declarados no cabeçalho de uma função existem somente dentro da função onde estão declarados.

# Passagem de Parâmetros

- Declaração da função
  - `int calc_produto(int valor1, int valor2)`
- Chamada da função
  - `calc_produto(oper1, oper2);`
- Atenção:
  - `valor1` e `valor2` existem na função `calc_produto`.
  - `oper1` e `oper2` existem na função `main`.
- Quaisquer modificações de `valor1` e `valor2` que aconteçam a partir da chamada de `calc_produto` só são conhecidas e percebidas dentro da função `calc_produto`.

# Passagem de Parâmetros

- Ao ser ativada a função `calc_produto`, `valor1` e `valor2` são criadas.
- E os valores existentes nesse momento em `oper1` e `oper2` são transferidos para `valor1` e `valor2`.
- A conexão entre `oper1` e `valor1` e `oper2` e `valor2` só existe no momento que a função é ativada.
- Fora o momento da ativação as funções `calc_produto` e `main` são mundos independentes.

# O quê é necessário para usar-se uma função em Linguagem C?

- A declaração da função.
  - Cabeçalho e corpo da função, com o código que produz o(s) resultado(s) esperado(s).
  - Se for função com tipo diferente de void, deve ter pelo menos um return, para atribuir valor à função.
- A chamada da função.
  - No ponto onde se deseja que a função seja executada, deve ser escrito o nome da função seguido de um par de parênteses, tendo no interior o nome dos parâmetros, se houver.
- Dependendo do caso, o protótipo da função.
  - As funções têm que ser declaradas antes de serem usadas. Para deixar a função main em destaque, é melhor declarar as funções definidas pelo usuário após a main. Então, para funções, o sistema aceita que primeiro só se indique o tipo, nome da função e tipos dos parâmetros, se houver , ou seja, o protótipo da função, e depois em algum ponto do código adiante, se declare a função de forma completa.

# Exemplo

```
//calcula produtos de pares de valores informados
#include <stdio.h>
#include <stdlib.h>
int calc_produto(int, int);
main (  )
{
    int seguir;
    int oper1, oper2, produto;
    do
      {
        printf("\nOperando 1: ");
        scanf("%d", &oper1);
        printf("\nOperando 2: ");
        scanf("%d", &oper2);

        printf ("\nProduto = %d\n", calc_produto(oper1, oper2));
        printf("\nMais um valor, digite 1, para parar, digite 0. ");
        scanf("%d", &seguir);
      }
    while (seguir);
    system("pause");
}
int calc_produto(int valor1, int valor2)
 {
    return valor1 * valor2;
 }
```

**Protótipo**

**Chamada da função**

**Declaração da função**

# Forma geral de declaração de um protótipo

- tipo_da_funcao  nome_da_função (lista de tipos dos parâmetros);
- tipo_da_funcao: o tipo de valor retornado pela função.
- nome_da_função: nome da função conforme as regras do C.
- lista de tipos dos parâmetros: tipo de cada parâmetro, separados entre si por vírgulas.

# Cuidados no uso de funções com parâmetros

- Em funções com parâmetros, cuidar que o número e o tipo dos parâmetros sejam coincidentes no protótipo (se usado), na declaração e na chamada.
- Em C, os parâmetros independentemente de seus nomes são emparelhados na declaração e chamada por ordem de declaração, da esquerda para a direita.
- Exemplo
  - int calc_produto(int, int);
  - int calc_produto(int valor1, int valor2)
  - calc_produto(oper1, oper2)
- Aninhamento de funções é possível?
  - Em C, é possível chamar uma função de dentro de outra função, mas não é possível declarar uma função dentro de outra função!

# Exercício1

- Escreva o código de uma função que calcule o fatorial de um número informado  como parâmetro

- Escreva um programa que use esta função

# Exercício1 - Solução

```
#include<stdio.h>
#include <stdlib.h>
double fatorial(int);
main(){
	int N;
	printf ("Informe o numero: ");
	scanf("%d",&N);
	printf("fatorial: %lf\n",fatorial(N));
	system("pause");
}
// declaracao da funcao fatorial
double fatorial(int n){
	int I;
	double fat=1.0;
	for (I=1;I<=n;I++)
		fat=fat*I;
	return(fat);
}
```

# Exercício 2

- Escreva o código de uma função que calcule a média aritmética  de dois valores informados como parâmetros

- Escreva um programa que use esta função

# Exercício 2

```
#include<stdio.h>
#include <stdlib.h>
float media(float, float);
main(){

        float v1,v2,m;
        printf ("Informe os numeros: ");
        scanf("%f %f",&v1,&v2);
        m=media(v1,v2);
        printf("a media dos numeros e': %.4f\n",m);
        system("pause");
}
// declaracao da funcao media
float media(float n1, float n2){
        return((n1+n2)/2);
}
```

# Exercício 2 – Solução

```
#include<stdio.h>
#include <stdlib.h>
float media(float, float);
main(){

        float v1,v2,m;
        printf ("Informe os numeros: ");
        scanf("%f %f",&v1,&v2);
        m=media(v1,v2);
        printf("a media dos numeros e': %.4f\n",m);
        system("pause");
}
// declaracao da funcao media
float media(float n1, float n2){
        return((n1+n2)/2);
}
```

# Exercício 3

- Escreva o código de uma função que conte quantas ocorrências de um determinado caractere existem em um string. Ela recebe como entrada 2 parâmetros:
  - um string de caracteres e
  - o caractere a ser pesquisado.

- Escreva um programa que use esta função

# Exercício 3 - Solução

```
#include<stdio.h>
#include <stdlib.h>
#include <string.h>
int contaChar(char[],char);
main(){
       char texto[100],c;
       printf ("\n Informe uma string: ");
       gets (texto);
       printf ("\nInforme o caractere a ser contado: \n");
       scanf("%c",&c);
       printf("o caractere %c aparece %d vezes no texto\n",c,contaChar(texto,c));
       system("pause");
}
int contaChar(char s[], char ch) {
    int i,cont=0;
    for (i=0;i<strlen(s);i++)
        if (s[i]==ch) cont=cont+1;
    return cont;
}
```