



UFOP

# PROGRAMAÇÃO DE COMPUTADORES I



Instituto de Ciências Exatas e  
Aplicadas - Campus João Monlevade



DEPARTAMENTO DE  
COMPUTAÇÃO E SISTEMAS

## *Roteiro – Aula Teórica 6* *Funções*

# Funções Contexto

- Assistir a parte 1 seguinte lista de execução:
  - <https://www.youtube.com/playlist?list=PL1K9y5L0Vn9XomSwP0qiL0uLPIMH-INOS>

# Funções em Linguagens de Programação

- Frequentemente, dividimos um problema maior em problemas menores e resolvemos os problemas menores.
- Ao criarmos um programa para resolver um problema, utilizamos sub-rotinas para codificar trechos do programa que resolvem problemas menores.
- As sub-rotinas devem codificar a solução para um problema pequeno e específico.
- Sub-rotinas podem ser funções ou procedimentos (quando não retornam valores).



# Por que utilizar sub-rotinas?

- Evitar que os programas fiquem grandes demais e difíceis de serem lidos e compreendidos
- Separar o programa em partes que possam ser compreendidas de forma isolada (criação de módulos)
- Utilizar um código em diferentes partes do programa, sem que ele precise ser escrito em cada local em que se deseje utilizá-lo
- Permitir o reuso de código em outros programas (bibliotecas)

# Funções – Definição, Criação e Utilização

- Assistir as partes 2 e 3 seguinte lista de execução:
  - <https://www.youtube.com/playlist?list=PL1K9y5L0Vn9XomSwP0qiL0uLPIMH-INOS>

# Declaração de Funções em C

```
tipo_retorno nome ( tipo parâmetro1, ..., tipo parâmetroN)
{
    comandos;

    return variável_tipo_retorno;
}
```

- Uma função executa comandos e retorna algum resultado, cujo tipo é determinado por tipo\_retorno.
- Cada parâmetro é uma variável que assume o valor que for passado na chamada da função.
- O comando return fornece o resultado da execução desta função para quem a chamou.

# Exemplo de Declaração de Funções em C

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

```
int soma (int X, int Y) {  
    return (X+Y);  
}
```

```
int main( void ) {  
    int A, B, C;  
    scanf("%d %d", &A, &B);  
    C = soma(A, B);  
    printf("%d", C);  
    return 0;  
}
```



# Declaração de Procedimentos em C

```
void nome ( tipo parâmetro1, ..., tipo parâmetroN)
{
    comandos;
}
```

Um procedimento é um tipo especial de função que executa comandos e não retorna um resultado.

O resultado do procedimento são as alterações executadas no estado do programa

Cada parâmetro é uma variável que assume o valor que for passado na chamada do procedimento



# O tipo void

**void** é um tipo especial que indica “nada” ou “vazio”.

**void** é utilizado para indicar que:

- Uma função não retorna valor (ou seja, que a função é um procedimento);
- Uma função possui uma lista vazia de parâmetros.

# Exemplo de Procedimento

```
#include<stdio.h>

void imprimeMaior (int X, int Y) {
    if (X > Y)
        printf("%d", X);
    else
        printf("%d", Y);
}

int main() {
    int X, Y;
    scanf("%d %d", &X, &Y);
    imprimeMaior(X, Y);
    return 0;
}
```

# Regras para definições de sub-rotinas

- Sub-rotinas (funções ou procedimentos) só podem ser declaradas fora de outras funções;
- Sub-rotinas devem ser declaradas **antes** de serem usadas;
- O uso de **void** na lista de parâmetros na declaração de funções é opcional, mas indicado por clareza;
- O tipo de retorno de uma sub-rotina não declarado é assumido ser **int**.



# A função main

A função **main** é a primeira função executada no programa.

Ela possui tipo de retorno fixo (**int**) e é chamada automaticamente pelo sistema operacional quando o programa é executado.

O comando **return**, neste caso, indica ao sistema operacional se o programa funcionou corretamente ou não.

Por padrão, o valor **0** é interpretado como funcionamento correto e demais valores são interpretados como erros.



UFOP

Universidade Federal de Ouro Preto  
Campus João Monlevade  
Instituto de Ciências Exatas e Aplicadas



DEPARTAMENTO DE  
COMPUTAÇÃO E SISTEMAS

# Declarações comuns da função main

```
int main (void) {... return 0;}
```

Este programa não recebe parâmetros na linha de comando.

```
int main (int argc, char const * argv[])  
{... return 0;}
```

Este programa recebe parâmetros na linha de comando.

`argc` indica a quantidade de parâmetros; e  
`argv[]` permite recuperar os parâmetros



# Definição, declaração e chamada de Funções – Videoaula

- Assistir as partes 3 a 6 seguinte lista de execução:
  - <https://www.youtube.com/playlist?list=PL1K9y5L0Vn9XomSwP0qiL0uLPIMH-INOS>

# Variáveis globais e variáveis locais

- Variáveis declaradas fora de funções são chamadas globais e são visíveis a partir do ponto de declaração pelo restante do programa.
- Variáveis declaradas dentro de sub-rotinas são chamadas locais e só são visíveis dentro da sub-rotina em que foram declaradas.
- São variáveis locais:
  - variáveis declaradas dentro da sub-rotina;
  - os parâmetros declarados na definição da sub-rotina.

# Exemplo de variáveis globais e variáveis locais

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

```
int contador_global=0;
```

```
void imprimeMaior (int X, int Y) {  
    if (X > Y) printf("%d", X);  
    else printf("%d", Y);  
}
```

```
int main() {  
    int X, Y;  
    scanf("%d %d", &X, &Y);  
    imprimeMaior(X, Y);  
    return 0;  
}
```

// Variável global

// X e Y são  
variáveis locais ao  
procedimento

// X e Y são  
variáveis locais à  
função, diferentes  
das variáveis do  
procedimento  
anterior



UFOP

Universidade Federal de Ouro Preto  
Campus João Monlevade  
Instituto de Ciências Exatas e Aplicadas



DEPARTAMENTO DE  
COMPUTAÇÃO E SISTEMAS



# Escopo de variáveis

- O escopo de uma variável determina em quais partes do código ela pode ser acessada.
- Uma variável só pode ser acessada após o ponto em que é declarada.
- As regras de escopo de variáveis em C são:
  - As variáveis globais são visíveis por todas as funções.
  - As variáveis locais são visíveis apenas na função onde foram declaradas.

# Escopo de variáveis

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

```
int contador_global=0;
```

```
void imprimeMaior (int X, int Y) {  
    if (X > Y) printf("%d", X);  
    else printf("%d", Y);  
}
```

```
int main() {  
    int A, B;  
    scanf("%d %d", &A, &B);  
    imprimeMaior(A, B);  
    return 0;  
}
```

// pode ser acessada  
em qualquer ponto do  
programa

// X e Y são visíveis  
apenas neste  
procedimento

// A e B são visíveis  
apenas na função  
main



UFOP

Universidade Federal de Ouro Preto  
Campus João Monlevade  
Instituto de Ciências Exatas e Aplicadas



DEPARTAMENTO DE  
COMPUTAÇÃO E SISTEMAS

# Passagem de parâmetros por valor

- Quando invocamos uma sub-rotina devemos fornecer, para cada um dos seus parâmetros, um valor de mesmo tipo do parâmetro, respeitando a ordem e a quantidade de parâmetros declarados.
- Ao invocarmos uma sub-rotina passando variáveis no lugar dos parâmetros, os valores das variáveis são copiados para os parâmetros da função.
- Alterações (dentro da sub-rotina) no valor dos parâmetros não afetam as variáveis usadas na chamada da função.
- Isto é chamado Passagem de Parâmetros por Valor



# Protótipos (ou Assinaturas) de sub-rotinas (I)

- Para podermos implementar sub-rotinas em partes distintas do arquivo-fonte e podermos implementar sub-rotinas depois de utilizá-las, utilizamos protótipos (ou assinaturas) de sub-rotinas.
- Protótipos correspondem à primeira linha da definição de uma função.
- O protótipo de uma sub-rotinas deve aparecer antes do uso desta sub-rotina.
- Em geral, colocam-se os protótipos no início do arquivo-fonte.

# Protótipos (ou Assinaturas) de sub-rotinas (II)

**tipo\_retorno** nome ( **tipo** parâmetro1, ..., **tipo** parâmetroN) **Assinatura  
(Protótipo)  
da função**

```
{  
comandos;  
return variável_tipo_retorno;  
}
```

Corpo (definição) da função

**void** nome ( **tipo** parâmetro1, ..., **tipo** parâmetroN) **Assinatura  
(Protótipo) do  
procedimento**

```
{  
comandos;  
}
```

Corpo (definição) do procedimento



UFOP

Universidade Federal de Ouro Preto  
Campus João Monlevade  
Instituto de Ciências Exatas e Aplicadas



DEPARTAMENTO DE  
COMPUTAÇÃO E SISTEMAS

# Definindo função após o uso através de protótipo

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

```
int soma (int X, int Y) ; // Assinatura
```

```
int main( void ) {  
    int A, B, C;  
    scanf("%d %d", &A, &B);  
    C = soma(A, B); // Chamada da função  
    printf("%d", C);  
    return 0;  
}
```

```
int soma (int X, int Y) { // Definição da função  
    return (X+Y);  
}
```



# Escopo, chamadas e parâmetros – Videoaula

- Assistir as partes 7 a 9 seguinte lista de execução:
  - <https://www.youtube.com/playlist?list=PL1K9y5L0Vn9XomSwP0qiL0uLPIMH-INOS>

# Referências

- Material de aula do Prof. Ricardo Anido, da UNICAMP:  
<http://www.ic.unicamp.br/~ranido/mc102/>
- Material de aula da Profa. Virgínia F. Mota:  
<https://sites.google.com/site/virginiaferm/home/disciplinas>
- DEITEL, P; DEITEL, H. *C How to Program*. 6a Ed. Pearson, 2010.



# Agradecimentos

- Professores do Departamento de Ciência da Computação da UFJF que gentilmente permitiram a utilização das videoaulas elaboradas por eles.