# Introdução à Ciência da Computação

## Modularização de Algoritmos e Funções em C

Prof. Ricardo J. G. B. Campello

## Agradecimentos


- Parte dos slides a seguir são adaptações dos originais:
  - de A. L. V. Forbellone e H. F. Eberspächer
  - do Prof. Rudinei Goularte


## Sumário

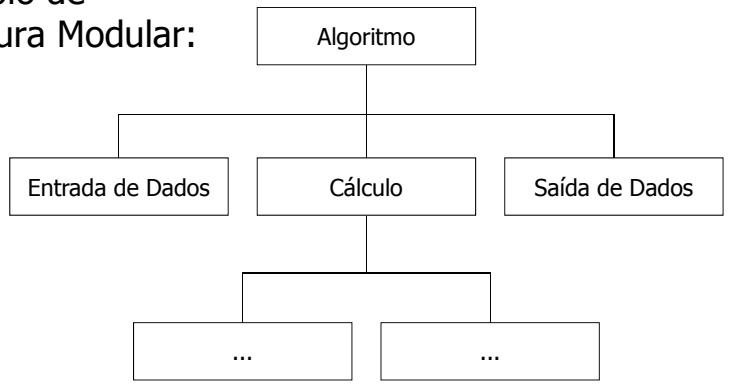
- Modularização de Algoritmos
- Módulos como Funções em C
  - Parâmetros e Retorno
  - Escopo de variáveis

## Modularização de Algoritmos

- Permite a construção de algoritmos e programas constituídos de **módulos**
- Consiste em dividir o algoritmo em sub-algoritmos (denominados **sub-rotinas**) que executam tarefas menores e bem definidas
  - possivelmente necessárias em diferentes programas ou partes de um mesmo programa
- Ajuda na elaboração de algoritmos e programas bem **estruturados**

# Modularização de Algoritmos

- Exemplo de Estrutura Modular:

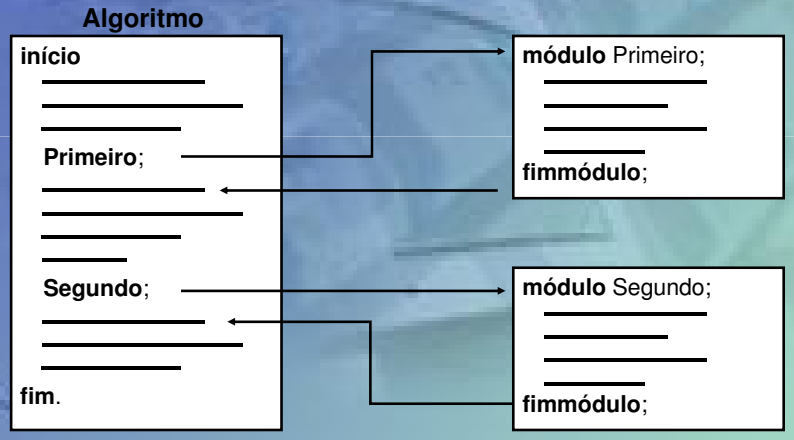



# Modularização de Algoritmos

- Manipulação:
  - Um módulo pode ser acionado de qualquer ponto do algoritmo principal ou de outro módulo
  - O acionamento de um módulo também é conhecido por **chamada** ou **ativação** do módulo
  - Quando ocorre uma chamada, o fluxo de execução passa para o módulo chamado
  - Quando se conclui a execução do módulo chamado, o fluxo retorna imediatamente após o ponto de chamada do algoritmo / módulo chamador

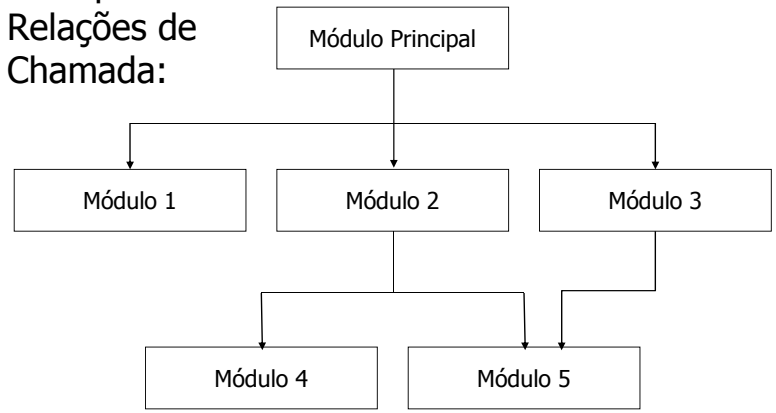
# Modularização de Algoritmos

- Esquema de Chamada :




# Modularização de Algoritmos

- Exemplo de Relações de Chamada:



# Modularização de Algoritmos

- Vantagens Principais da Modularização:
  - **Legibilidade**: clareza do algoritmo
  - **Independência**: construção e testes individualizados
  - **Manutenibilidade**: simplificação da manutenção
  - **Reusabilidade**: Reaproveitamento de código

# Parametrização de Módulos

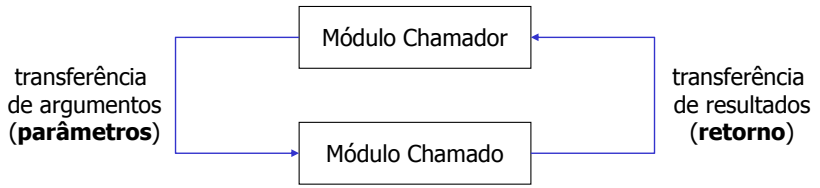
- É possível tornar um módulo mais genérico e portanto mais reutilizável
- Isso pode ser feito através de **parâmetros**
- Parâmetros são os valores que um módulo pode receber antes de iniciar sua execução
  - são também denominados *argumentos*
- Por exemplo:
  - um módulo que imprime a tabuada do 7 tem alguma aplicabilidade. Porém, se o módulo fosse capaz de imprimir qualquer tabuada, seria mais genérico
  - para isso, bastaria ativar o módulo indicando por **parâmetro** qual a tabuada desejada

# Retorno de Resultado

- Um módulo pode simplesmente realizar uma ou mais instruções, como imprimir na tela o resultado da tabuada do exemplo anterior
- Porém, assim como recebe valores como parâmetros, também pode **retornar** valores explicitamente como resultado

# Parâmetros e Retorno

- Vinculação entre módulos:
  - Transferência de argumentos:
    - módulo chamador → módulo chamado
  - Transferência de resultados:
    - módulo chamado → módulo chamador



# Módulos como Funções em C

- Um módulo em linguagem C é realizado por uma **função**
  - Funções recebem ou não valores (argumentos) como parâmetros e retornam ou não explicitamente um valor (resultado)
- Declaração:

```
tipo nome (lista de parâmetros) {
  instruções ;  /* corpo da função */
}
```

- Exemplo:

```
double quadrado (double x) {
    return x*x;
    }
```



# Funções em C

- O retorno de valor é feito através do comando **return**
  - **return** interrompe imediatamente o fluxo de execução
    - e retorna o valor especificado
- Funções que não retornam valores são do tipo **void**
  - **void** também é usado para indicar a inexistência de parâmetros
- Exemplo:

```
void hello(void){
      printf("Hello World!");
}
```



# Funções em C

- Nota 1:
  - `main` é uma função como qualquer outra em C
  - A diferença é que esta função é chamada quando o programa é inicialmente executado
    - não por uma outra função quando o programa já está em execução



# Funções em C

- Nota 2:
  - C exige que sejam declarados, no início do programa, **protótipos** de cada função definida pelo programador
  - Exemplo:

```
#include <stdio.h>
double quadrado(double x); /* protótipo */
void main(void){
   double y;
   y = quadrado(3.5);
   printf("%f", y);
}
double quadrado(double x){
   return x*x;
}
```

# Passagem de Parâmetros

- Existem dois tipos de passagem de parâmetros para uma função:
  - Passagem **por valor**
  - Passagem **por referência**
- Para entender a diferença entre esses tipos, é importante compreender o conceito de **escopo de variáveis**

# Escopo de Variáveis

- Variáveis **Globais**:
  - Declaradas fora de qualquer função (incluindo `main`)
    - no início do programa
  - São visíveis em qualquer parte do programa
- Variáveis **Locais**:
  - Declaradas dentro de funções
  - São visíveis apenas dentro do escopo a que pertencem
  - "Desaparecem" quando a função encerra sua execução
- **Conflito**:
  - Vale a variável local

# Exemplo de Variáveis Locais

- Vejamos novamente o exemplo da Nota 2:

```
#include <stdio.h>
double quadrado(double x);
void main(void){
    double y;                   /* y é variável local */
    y = quadrado(3.5);
    printf("%f", y);
}
double quadrado(double x){      /* x é variável local */
    return x*x;
}
```

- Temos 2 variáveis locais neste exemplo, uma (y) restrita ao escopo da função `main` e a outra (x) restrita ao escopo da função `quadrado`

# Exemplo de Variável Global

```
#include <stdio.h>
int teste;              /* variável global */
void leia(void);        /* protótipo */
void escreva(void);     /* protótipo */
void main(void){
   leia();              /* chamada (função sem parâmetros) */
   escreva();           /* chamada (função sem parâmetros) */
}
void leia(void){
   scanf("%d", &teste);
}
void escreva(void){
   printf("%d", teste);
}
```

# Passagem de Parâmetros

- Geralmente, a passagem de parâmetros em C é realizada **por valor**:
  - Alterações feitas nos parâmetros recebidos não se refletem nos valores passados como argumentos
  - O valor do argumento é *copiado* durante a chamada da função
  - Para ilustrar, façamos uma pequena modificação no exemplo anterior da Nota 2...

# Passagem de Parâmetros

- Exemplo:

```
#include <stdio.h>
double quadrado(double x);
void main(void){
   double y, z = 3.5;
   y = quadrado(z);
   printf("%f %f", z, y);
}
double quadrado(double z){
   z = z*z;
   return z;
}
```

Saída:

```
3.5  12.25
```

# Passagem de Parâmetros

- Passagem **por referência**
  - Alterações feitas nos parâmetros recebidos são refletidas nos valores passados como argumentos
  - Para entender este processo em linguagem C, é preciso antes compreender o conceito de ponteiro
    - tópico avançado no curso...

# Funções Pré-Definidas

- As linguagens de programação têm à sua disposição várias funções pré-definidas
  - Em C, essas funções são organizadas em bibliotecas
- Exemplo (Biblio. Matemática C ANSI – #include <math.h>):
  - pow(b,e) base *b* elevada ao expoente *e*
  - sqrt(x) raiz quadrada de x

*Parâmetro*

*Identificador (nome)*

# Ativação de Funções

- Ativação e captura do resultado de uma função pode ser:
  - Por atribuição direta do retorno a uma variável do mesmo tipo
  - Por uso do retorno dentro de uma expressão
- Exemplo (H é **real**, A e Y são **inteiros** ou **reais**):

```
H = sqrt(A + pow(Y,2));
```

atribuição direta — uso em expressão



# Exemplo

- Dados dois números *N* e *K*, calcular a Combinação:

$$C = \frac{N!}{K!(N-K)!}$$

- Se existisse uma função **fat**(X) que calculasse o fatorial de um dado X, o cálculo acima ficaria:

C = **fat**(N) / (**fat**(K) * **fat**(N-K))

- Se não existe, podemos defini-la



# Exemplo

```
função FAT(inteiro: X): retorna inteiro;
início
    inteiro: P, I;
    P ← 1;
    para I de 1 até X faça P ← P * I;
    retorne P;
fim
```

```
long int FAT(long int X){
      long int I, P;
      P = 1;
      for(I=1; I<=X; I++) P=P*I;
      return P;
}
```



# Exercícios

- Faça um programa em C que receba do usuário dois inteiros, N e K, e utilize a função definida anteriormente para calcular (e depois exibir) a combinação:

C = **fat**(N) / (**fat**(K) * **fat**(N-K))

- Modifique o exercício anterior para que, ao invés de uma única combinação, todas as combinações de N e K entre 1 e 10 sejam calculadas e armazenadas em uma matriz 11 × 11 com elementos C[N][K]
  - Nota: ignore (não utilize) as células C[0][K] e C[N][0]

## Exercícios

- Faça uma função em C que receba como parâmetros dois valores reais e então calcule e retorne a **média aritmética** desses valores
- Refaça o exercício anterior para o cálculo da **média harmônica** (pesquise!) ao invés da média aritmética
- Refaça o exercício anterior para que a função retorne o menor (**mínimo**) dentre os dois valores recebidos como parâmetros
- Refaça o exercício anterior para retornar o **máximo**



## Exercícios

- Faça uma função em C que receba três parâmetros **double**, a, b e c, associados aos coeficientes de um polinômio de 2º grau $ax^2 + bx + c$, e então calcule e apresente na tela as raízes reais desse polinômio
  - Se as raizes não forem reais, forem complexas, isso deve ser escrito na tela ao invés das raizes



## Bibliografia


- Schildt, H. "C Completo e Total", 3a. Edição, Pearson, 1997.
- Damas, L. "Linguagem C", 10a. Edição, LTC, 2007