



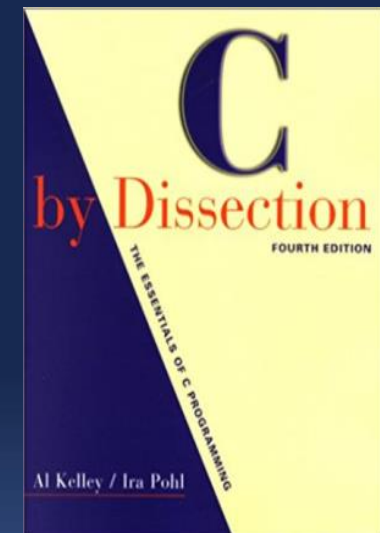
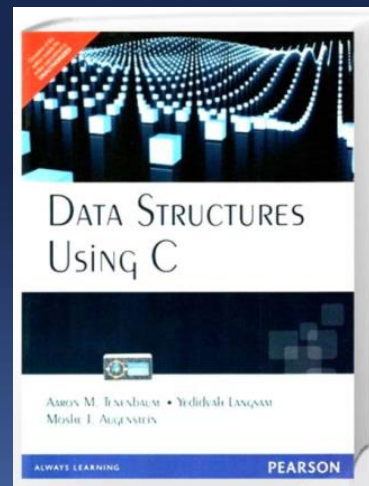
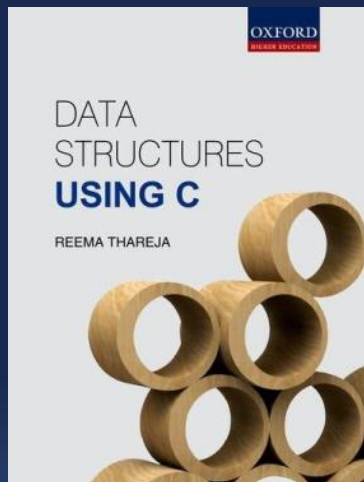
Algoritmos e Estrutura de Dados – II

Modularidade (Funções)


Agradecimento ao Prof. Aparecido V. de Freitas,
por compartilhar o material

Bibliografia

- ✓ Data Structures using C - Oxford University Press - 2014
- ✓ Data Structures Using C - A. Tenenbaum, M. Augensem, Y. Langsam, Pearson 1995
- ✓ C By Dissection - Kelley, Pohn - Third Edition - Addison Wesley



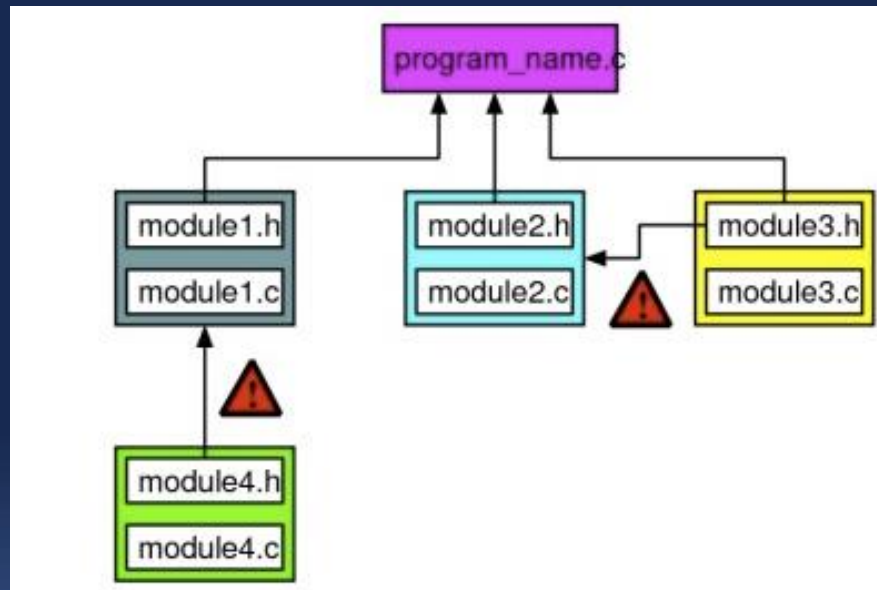
Introdução

- 
Modularidade significa dividir o código em diversas partes (**divisão** e **conquista**) no qual cada módulo tem uma **funcionalidade** muito bem definida.



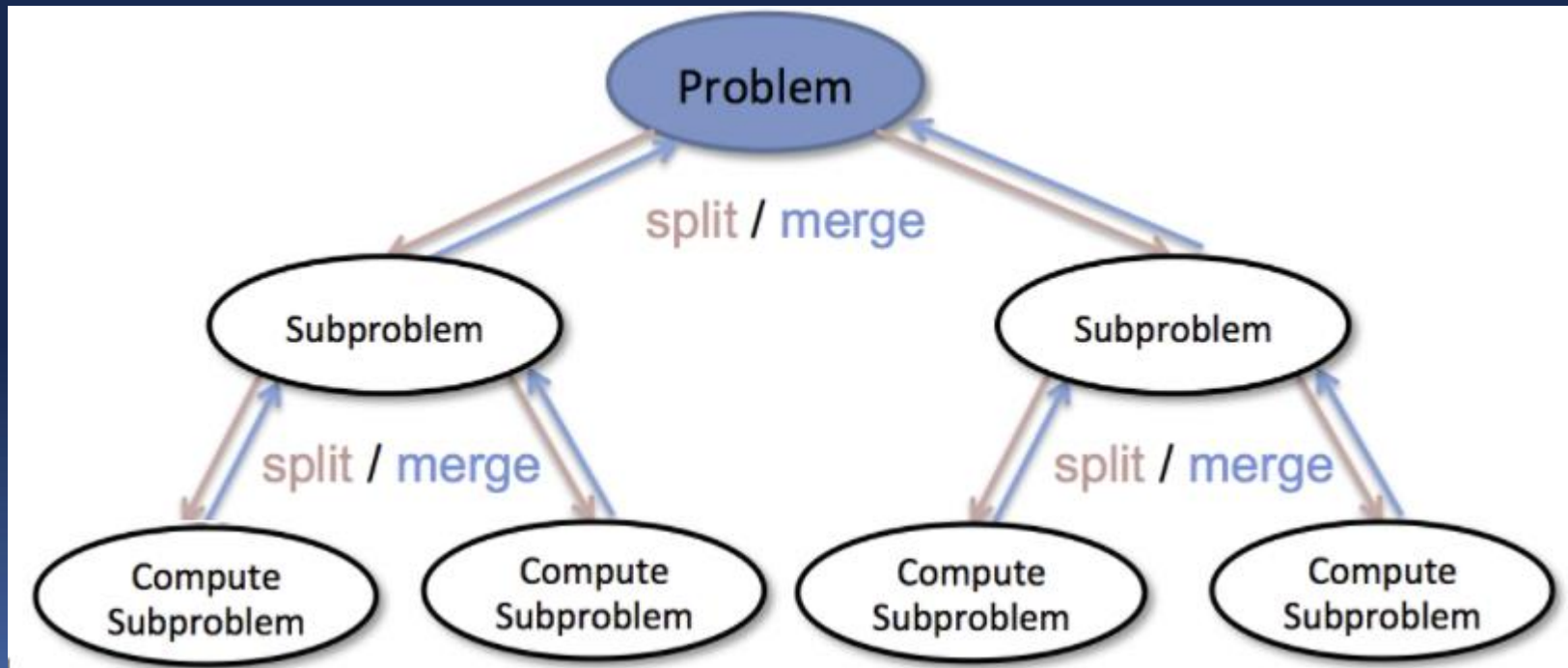
Modularidade

- ❖ Por meio desse conceito, pode-se aproveitar códigos de funções que já foram previamente testadas e utilizá-los em novos programas.



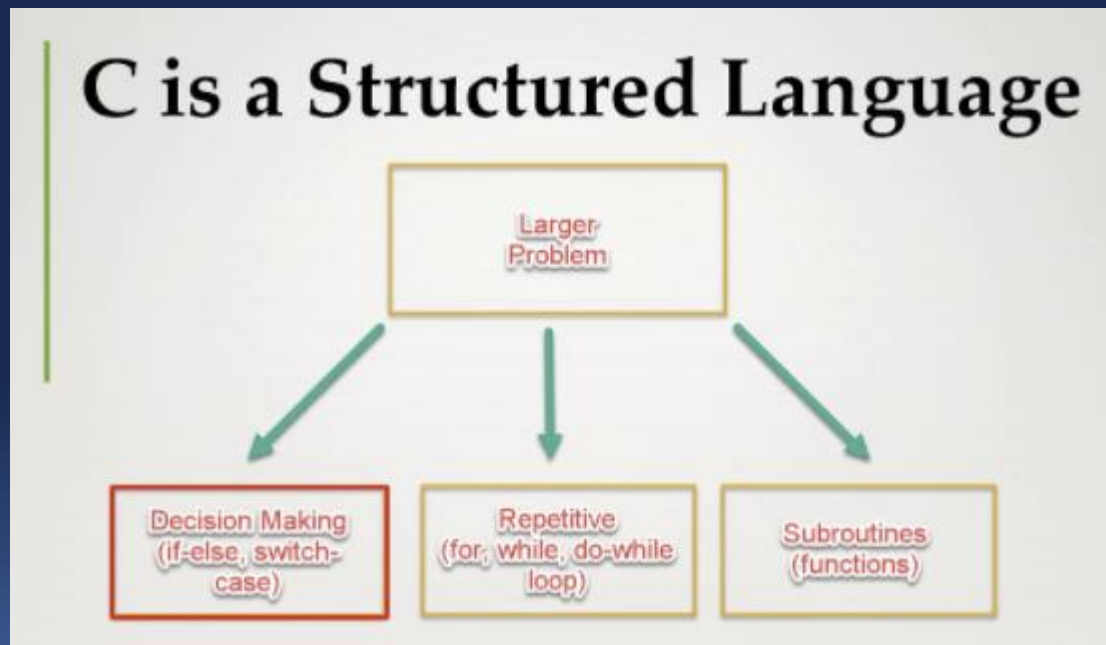
Divide and Conquer

- ❖ Corresponde à uma técnica de programação no qual um problema mais complexo pode ser **decomposto** em problemas menores com algoritmos mais simples;
- ❖ A solução completa do problema pode ser obtida **combinando-se** os sub-problemas desenvolvidos com módulos mais coesos.



Programação Estruturada

- O controle de fluxo em um programa deve ser o mais simples possível;
- A construção do código deve adotar uma estratégia top-down.



Refinamento sucessivo



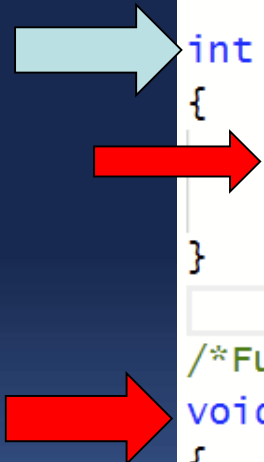
- O design **top-down**, também chamado **refinamento sucessivo**, consiste de repetidamente decompor o problema em sub-problemas;
- Essa técnica de design é implementada em **C** por meio de **funções**;

```
#include<stdio.h>

/*Function prototype Decleration*/
myfunction();

int main()
{
    myfunction(); /* Function Call*/
    return 0;
}

/*Function Definition*/
void myfunction()
{
    printf("Hello,I am a Function\n");
}
```

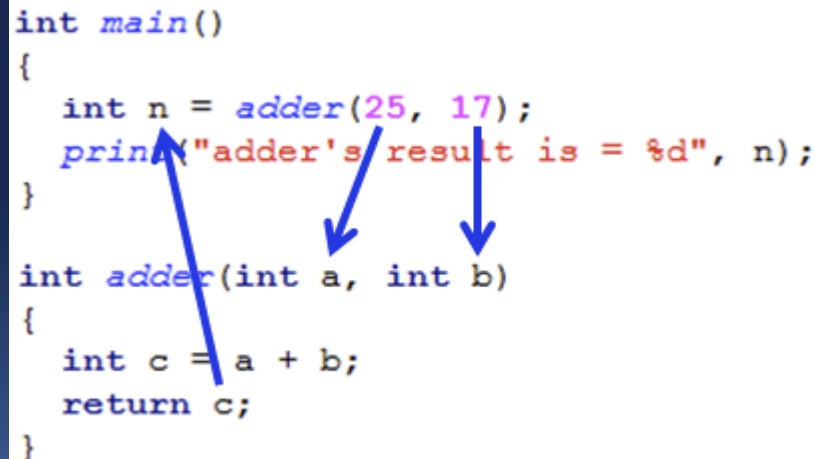


Invocação de Função

- Um programa em C é estruturado em funções, uma das quais é a função **main()**;
- Durante o fluxo de execução, ao se encontrar um nome de função seguido por parênteses, a função é chamada (ou **invocada**).

```
int main()
{
    int n = adder(25, 17);
    printf("adder's result is = %d", n);
}

int adder(int a, int b)
{
    int c = a + b;
    return c;
}
```



Invocação de Função

```
//Programa 01 - Unidade 8
#include <stdio.h>
#include <locale.h>

void printMsg(void);
//-----

int main() {
    setlocale(LC_ALL, "Portuguese");
    printf("\nInicio do Programa 01");

    printMsg();

    printf("\nFim do Programa 01");
    return 0;
}
//-----

void printMsg(void) {
    printf("\nHello World...");
}
//-----
```

Invocação de Função

```
E:\USCS\DISCIPLINAS_USCS\Disciplinas_1S_2021\Alg_Est_Dados_I\Fontes_C\Unidade_8\Pgm_01.exe
Inicio do Programa 01
Hello world...
Fim do Programa 01
-----
Process exited after 0.3846 se
Press any key to continue . .
```

Funções retornam valor



```
//Programa 02 - Unidade 8
#include <stdio.h>
#include <locale.h>

int printMsg(void);
//-----

int main() {
    setlocale(LC_ALL, "Portuguese");
    printf("\nInicio do Programa 02");
    int codRetorno;

    codRetorno = printMsg();
    printf("\ncodRetorno = %d", codRetorno);
    printf("\nFim do Programa 02");
    return 0;
}
//-----

int printMsg(void) {
    printf("\nHello World...");
    return 0;
}
//-----
```



Funções retornam valor



```
E:\USCS\DISCIPLINAS_USCS\Disciplinas_15_2021\Alg_Est_Dados_I\Fontes_C\Unidade_8\Pgm_01.exe
Inicio do Programa 02
Hello world...
codRetorno = 0
Fim do Programa 02
-----
Process exited after 0.0
Press any key to continu
```



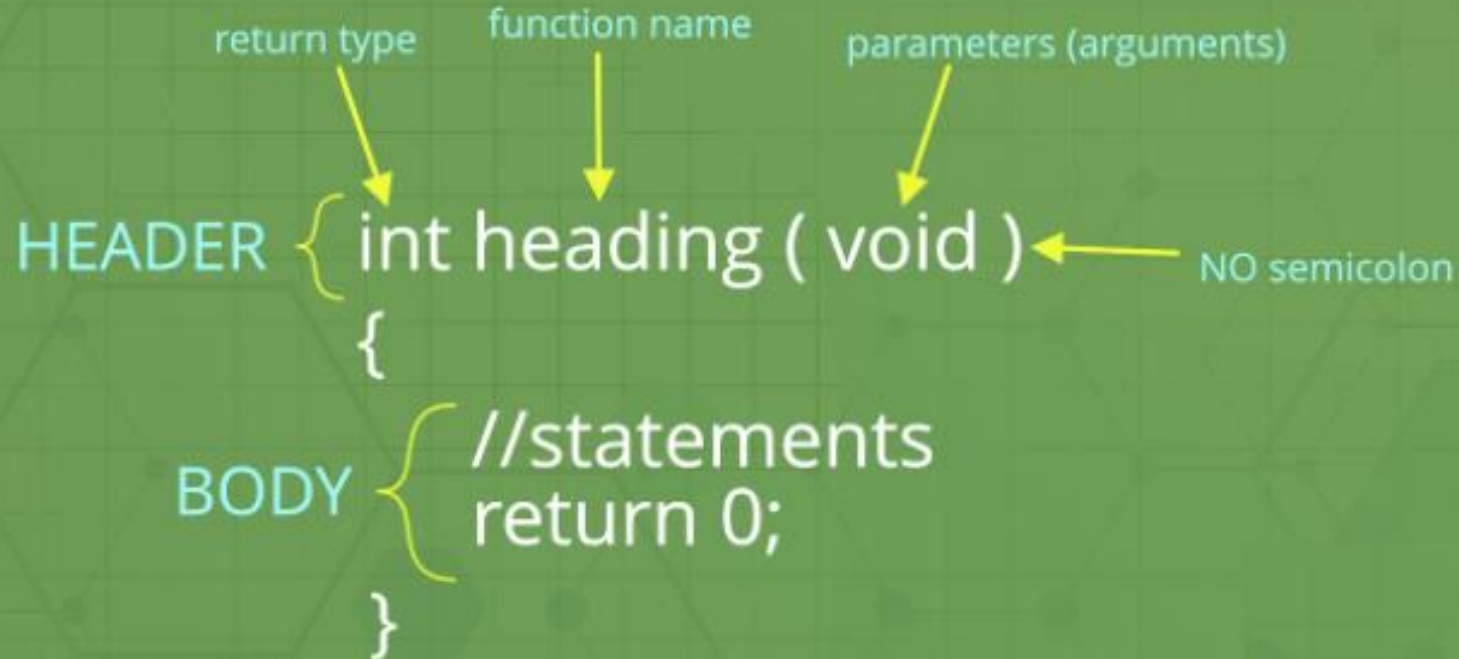
Definição de Função

Function Prototype

return type function name parameters (arguments)

HEADER { int heading (void) ← NO semicolon

BODY { //statements
return 0;
}



Funções recebem argumentos



- ❖ Em tempo de execução de uma função, dados (**argumentos**) podem ser recebidos;
- ❖ Em tempo de definição da função esses dados são identificados por **parâmetros**.

```
1 void print(String someString){
2     System.out.println(someString);
3
4 }
5
6 print("Hi Jim");
```

parameter
* general placeholders

Argument
* actual value
"Hi Jim"
passed to the
method/function



Funções recebem argumentos



```
//Programa 03 - Unidade 8
#include <stdio.h>
#include <locale.h>

int printMsg(int n);
//-----

int main() {
    setlocale(LC_ALL, "Portuguese");
    printf("\nInicio do Programa 03");
    int codRetorno, n;
    printf("\nEntre com um valor inteiro: ");
    scanf("%d", &n);
    codRetorno = printMsg(n);
    printf("\ncodRetorno = %d", codRetorno);
    printf("\nFim do Programa 03");
    return 0;
}
//-----

int printMsg(int n) {
    int i;
    for(i=0 ; i < n ; i++)
        printf("\nHello World...");
    printf("\nTexto impresso %d vezes...", n);
    return 0;
}
//-----
```



Funções recebem argumentos



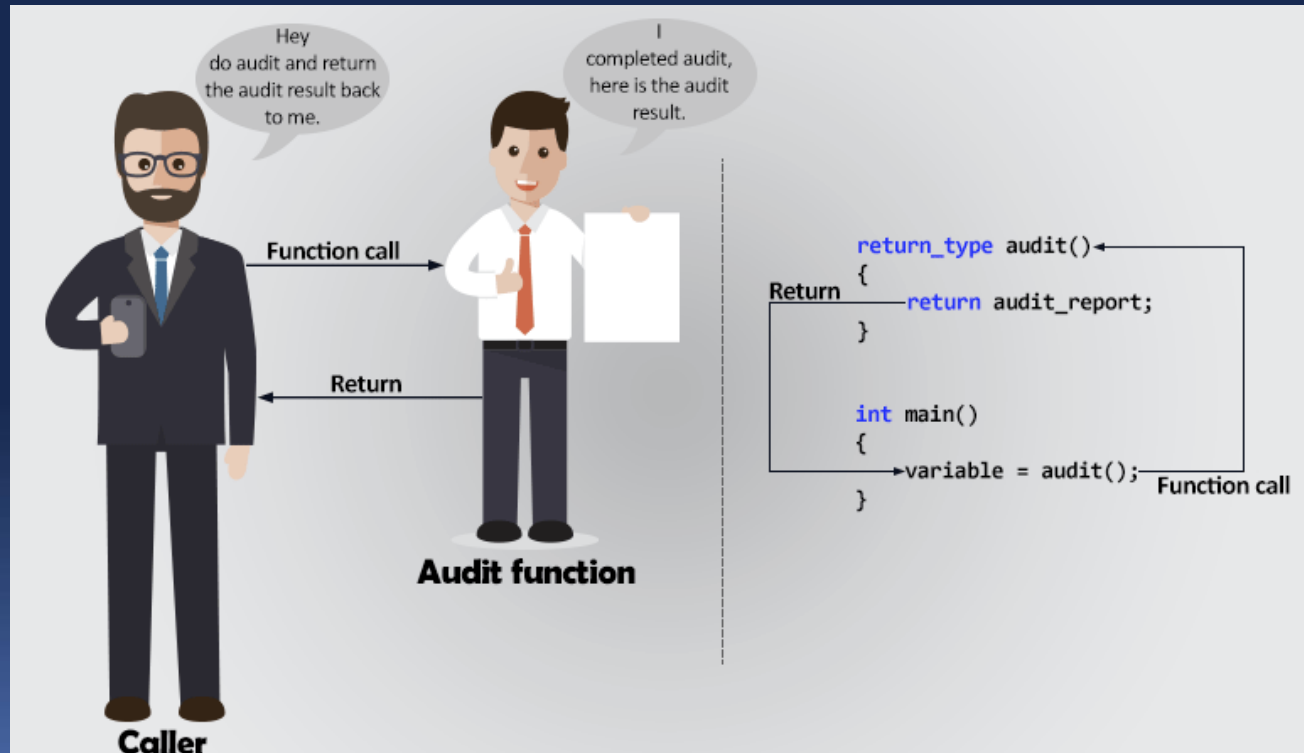
```
E:\USCS\DISCIPLINAS_USCS\Disciplinas_1S_2021\Alg_Est_Dados_1\Fontes_C\Unidade_8\Pgm_03.exe
Inicio do Programa 03
Entre com um valor inteiro: 5

Hello world...
Hello world...
Hello world...
Hello world...
Hello world...
Texto impresso 5 vezes...
codRetorno = 0
Fim do Programa 03
-----
Process exited after 5.244 secon
Press any key to continue . . .
```



O comando return

- ❖ Quando executado, o fluxo de controle do programa é imediatamente passado para o ambiente chamador;
- ❖ Se uma expressão seguir o comando **return**, ela será avaliada e o valor da expressão será retornado.



O comando return



```
//Programa 04 - Unidade 8
```

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

```
#include <locale.h>
```

```
int minValor(int n1, int n2);
```

```
//-----
```

```
int main() {
```

```
    setlocale(LC_ALL, "Portuguese");
```

```
    printf("\nInicio do Programa 04");
```

```
    int n1, n2, valorMinimo;
```

```
    printf("\nEntre com um valor inteiro: ");
```

```
    scanf("%d", &n1);
```

```
    printf("\nEntre com um valor inteiro: ");
```

```
    scanf("%d", &n2);
```

```
    valorMinimo = minValor(n1,n2);
```

```
    printf("\nValor Mínimo: %d ", valorMinimo);
```

```
    printf("\nFim do Programa 04");
```

```
    return 0;
```

```
}
```

```
//-----
```

```
int minValor(int n1, int n2) {
```

```
    if (n1 < n2)
```

```
        return n1;
```

```
    return n2;
```

```
}
```

```
//-----
```



O comando return

```
E:\USCS\DISCIPLINAS_USCS\Disciplinas_15_2021\Alg_Est_Dados_1\Fontes_C\Unidade_8\Pgm_04.exe
Inicio do Programa 04
Entre com um valor inteiro: 2

Entre com um valor inteiro: 3

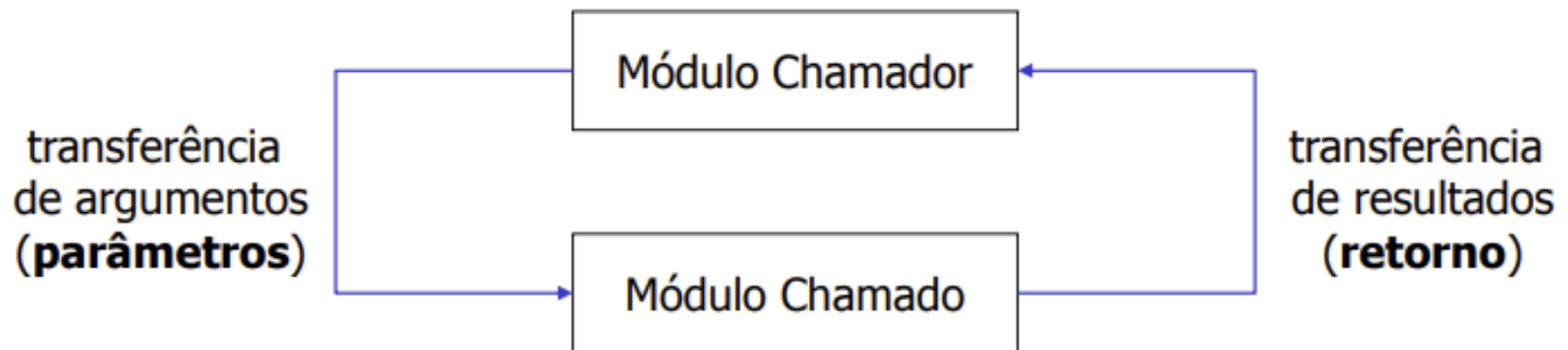
Valor Mínimo: 2
Fim do Programa 04
-----
Process exited after 4.25 second
Press any key to continue . . .
```

Retorno de Resultados

- 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

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`

Variáveis Globais



```
#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

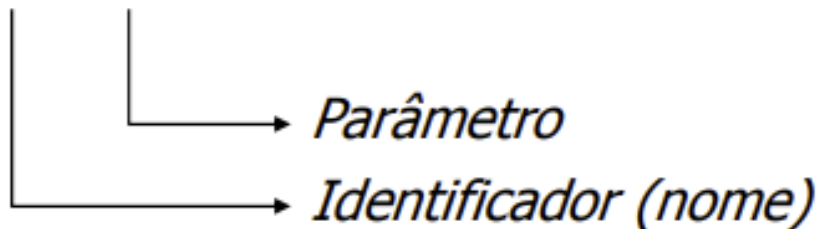
Passagem de Parâmetros

■ 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**

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



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**):

$$H = \underbrace{\text{sqrt}(A + \text{pow}(Y, 2))}_{\text{atribuição direta}};$$

$$\underbrace{\text{pow}(Y, 2)}_{\text{uso em expressão}}$$

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:

$$C = \mathbf{fat}(N) / (\mathbf{fat}(K) * \mathbf{fat}(N-K))$$

- 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 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

