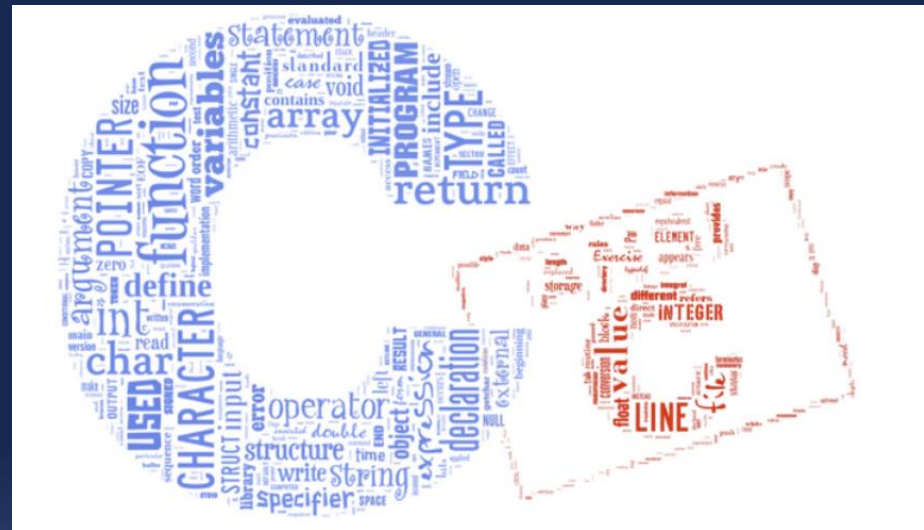




Programação Avançada e Estrutura de Dados

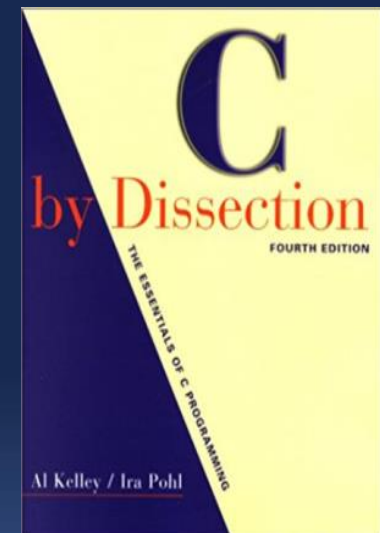
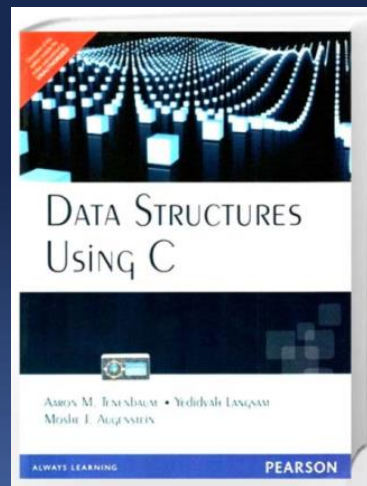
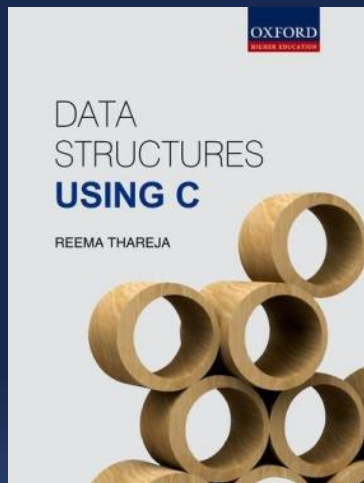
Unidade 6 – Modularidade e Recursão




Prof. Aparecido V. de Freitas
Doutor em Engenharia
da Computação pela EPUSP
aparecido.freitas@online.uscs.edu.br
aparecidovfreitas@gmail.com

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, Pohl - Third Edition - Addison Wesley



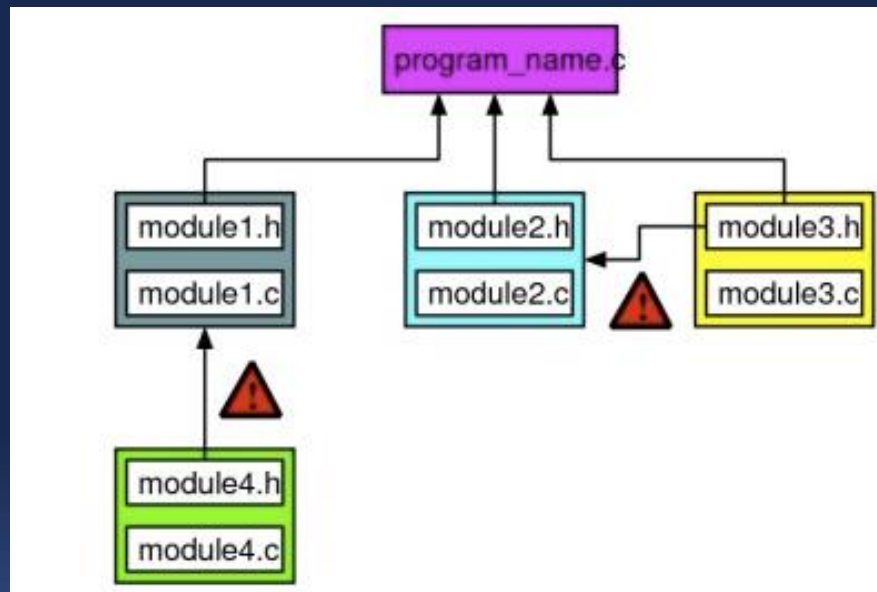
Introdução

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



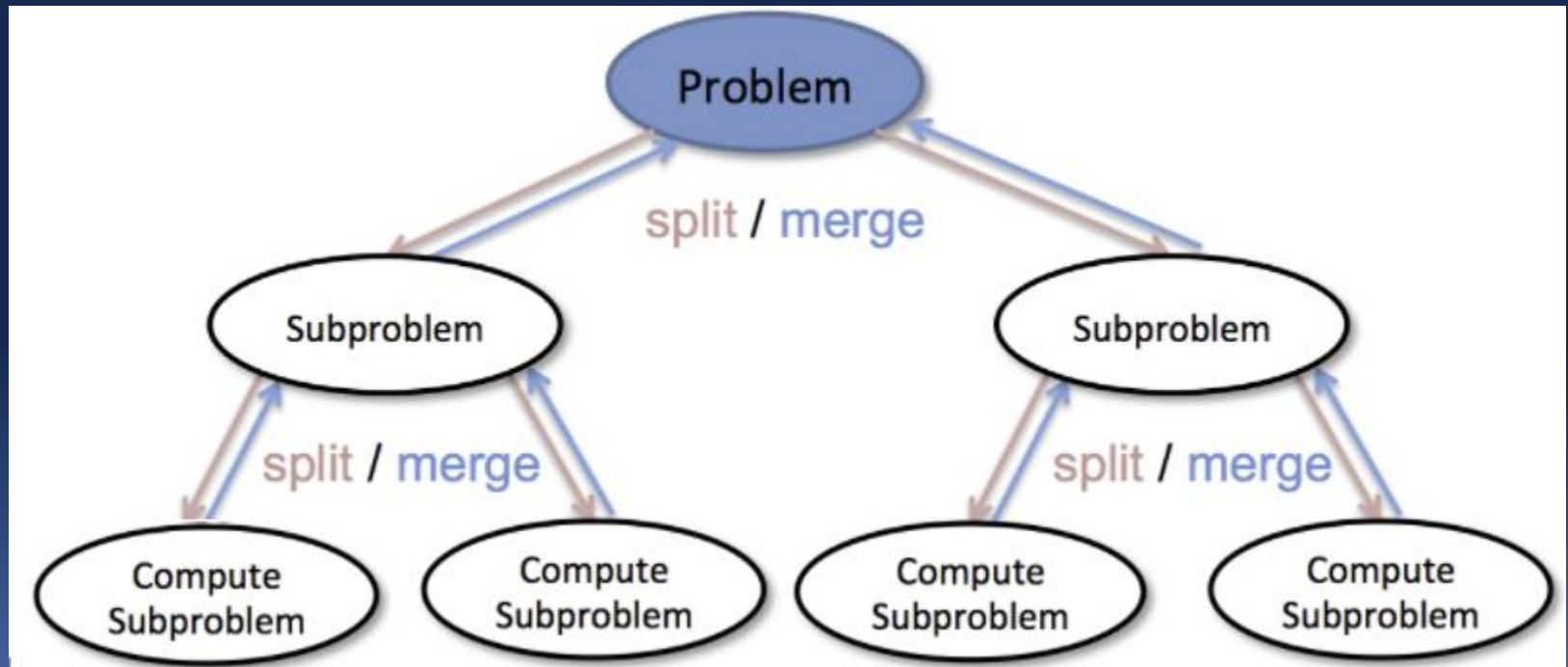
Modularidade

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



Divide and Conquer

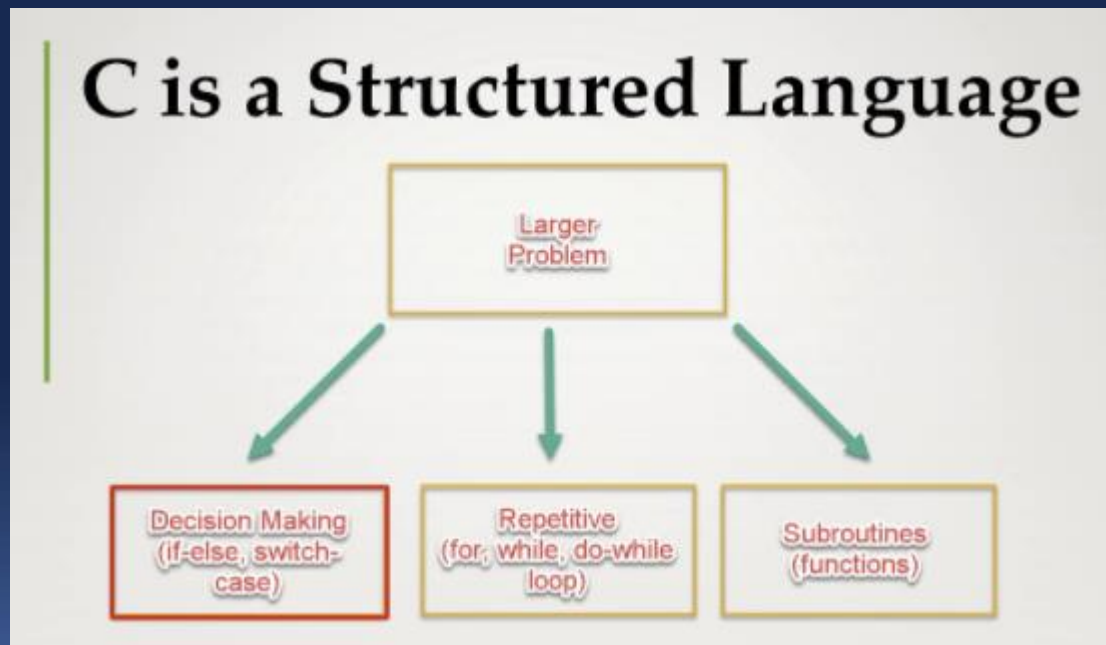
- ❖ Corresponde à uma técnica de programação no qual um problema mais complexo pode ser **decomposto** em problemas menores com algoritmos mais simples;
- ❖ A solução completa do problema pode ser obtida **combinando-se** os subproblemas desenvolvidos com módulos mais coesos.



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");
}
```

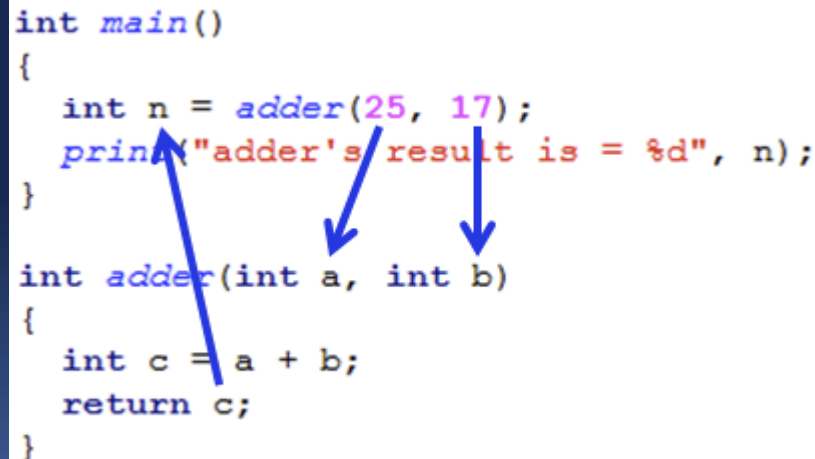


Invocação de Função

- 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);
//-----

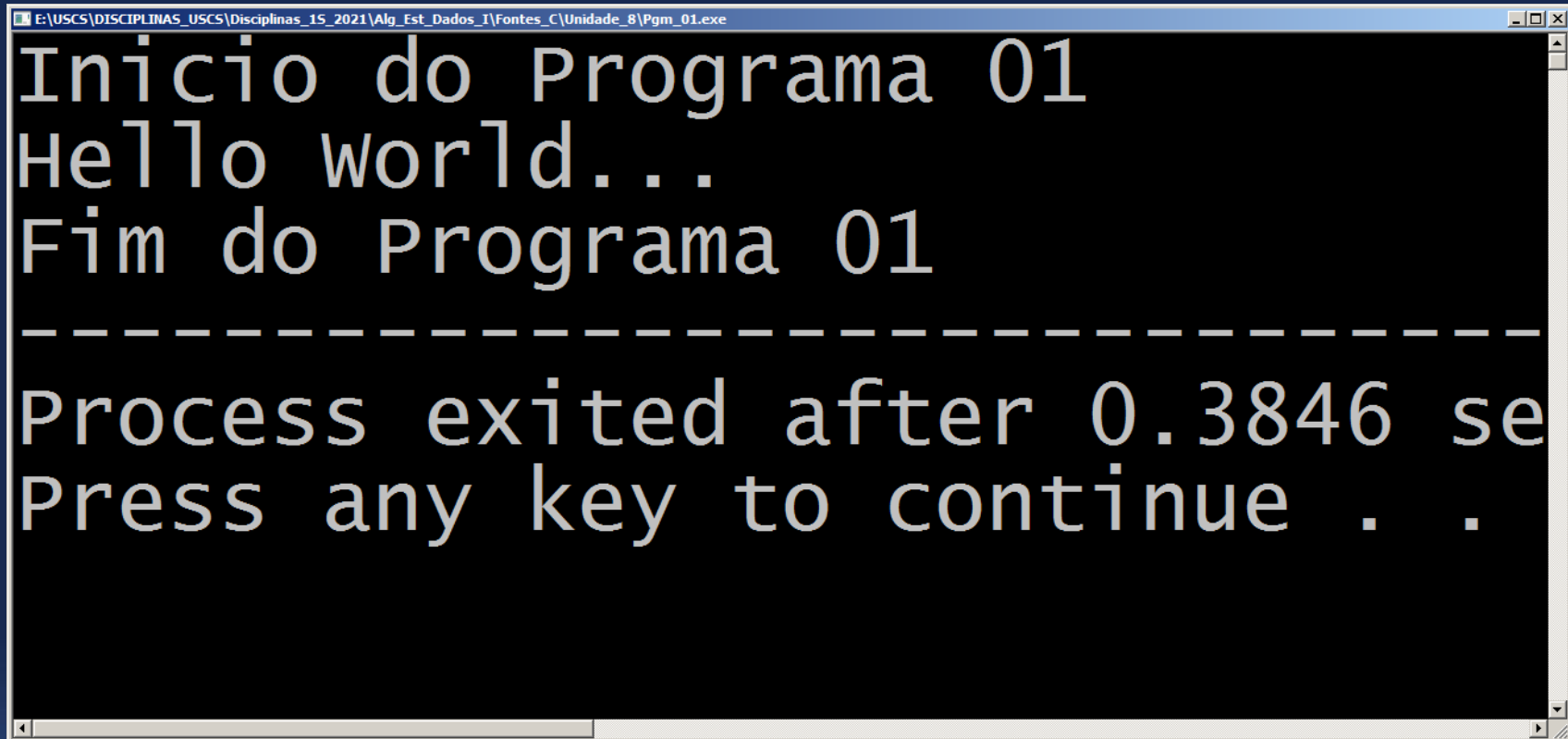
int main() {
    setlocale(LC_ALL, "Portuguese");
    printf("\nInicio do Programa 01");

    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
Início 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
```



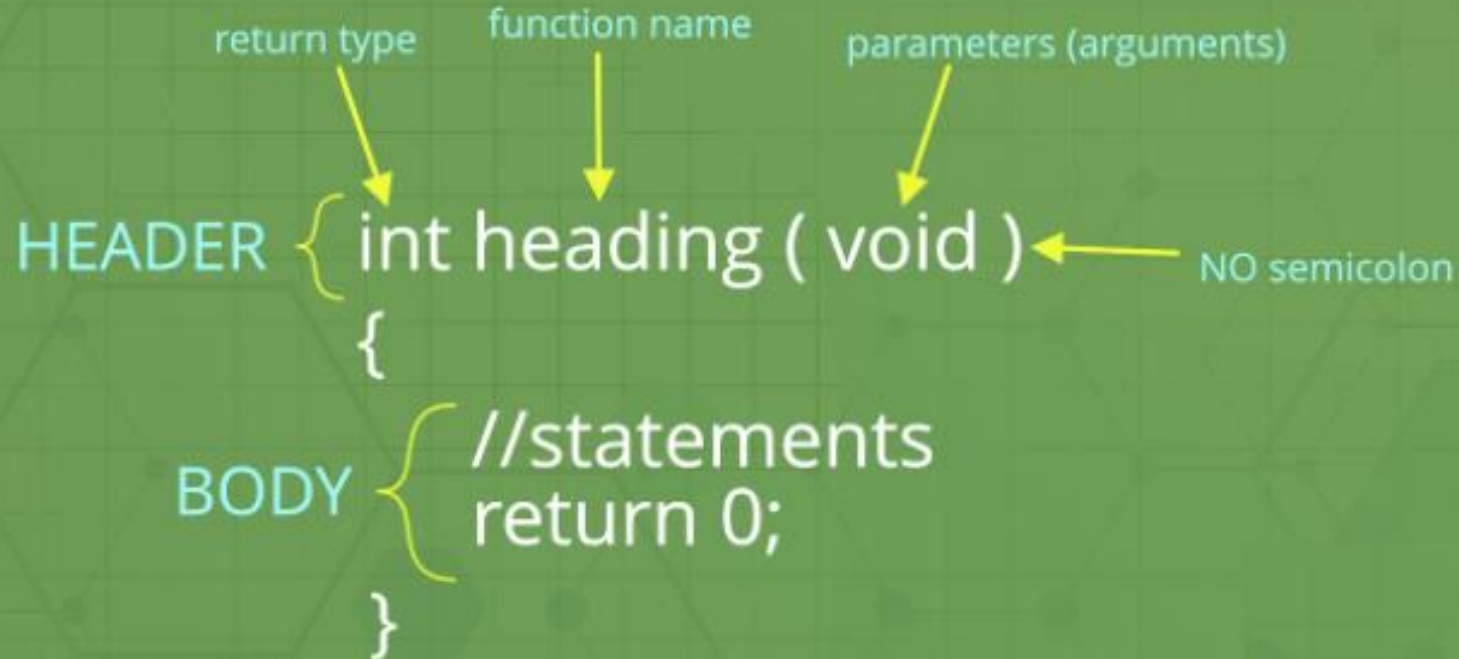
Definição de Função

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



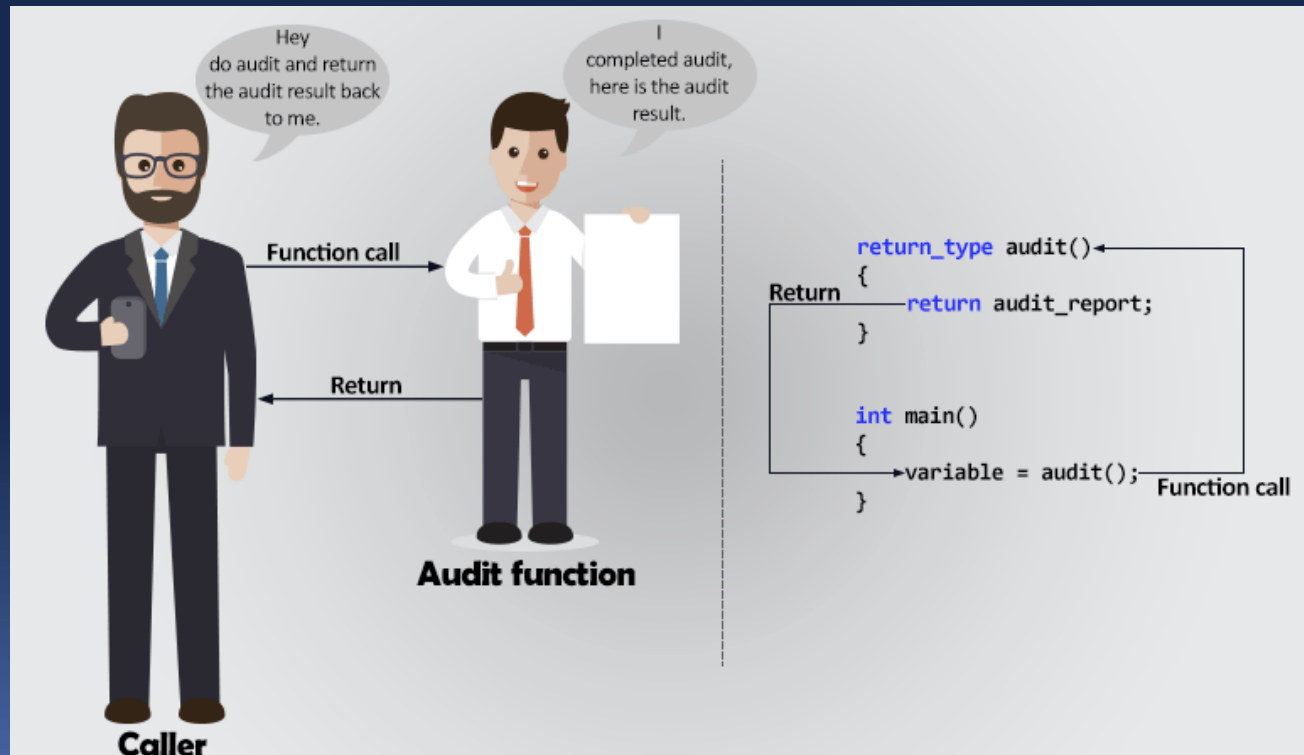
```
E:\USCS\DISCIPLINAS_USCS\Disciplinas_1S_2021\Alg_Est_Dados_1\Fontes_C\Unidade_8\Pgm_03.exe
Inicio do Programa 03
Entre com um valor inteiro: 5

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;  
}  
//-----
```



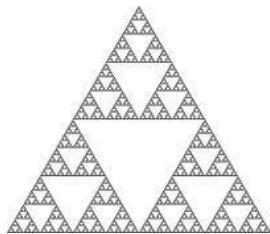
O comando return

```
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 . . .
```

Recursividade



Introdução

- Repetição de instruções pode ser obtida por meio iterações;
- Outra forma de se implementar repetições é por meio de Recursão;
- Recursão ocorre quando uma função faz chamada de si própria;
- Entretanto, a fim de gerar uma resposta, uma condição de término deve ocorrer;
- Algoritmos recursivos são representados por Recorrências;
- Uma Recorrência é uma expressão que fornece o valor de uma função em termos dos valores “anteriores” da mesma função.



Exemplo – Fatorial

▣ O fatorial de um inteiro positivo n , denotado por $n!$, é definido por:

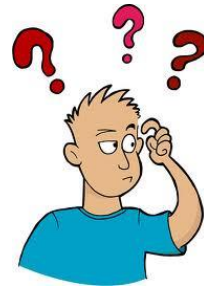
$$n! = \begin{cases} 1 & \text{se } n=0 \\ n.(n-1).(n-2)... 3.2.1 & \text{se } n \geq 1 \end{cases}$$



Função Fatorial

▣ Exemplo: $5! = 5.4.3.2.1 = 120$

Será que a função fatorial pode ser definida de forma recursiva ?



Função Fatorial

```
#include <stdio.h>

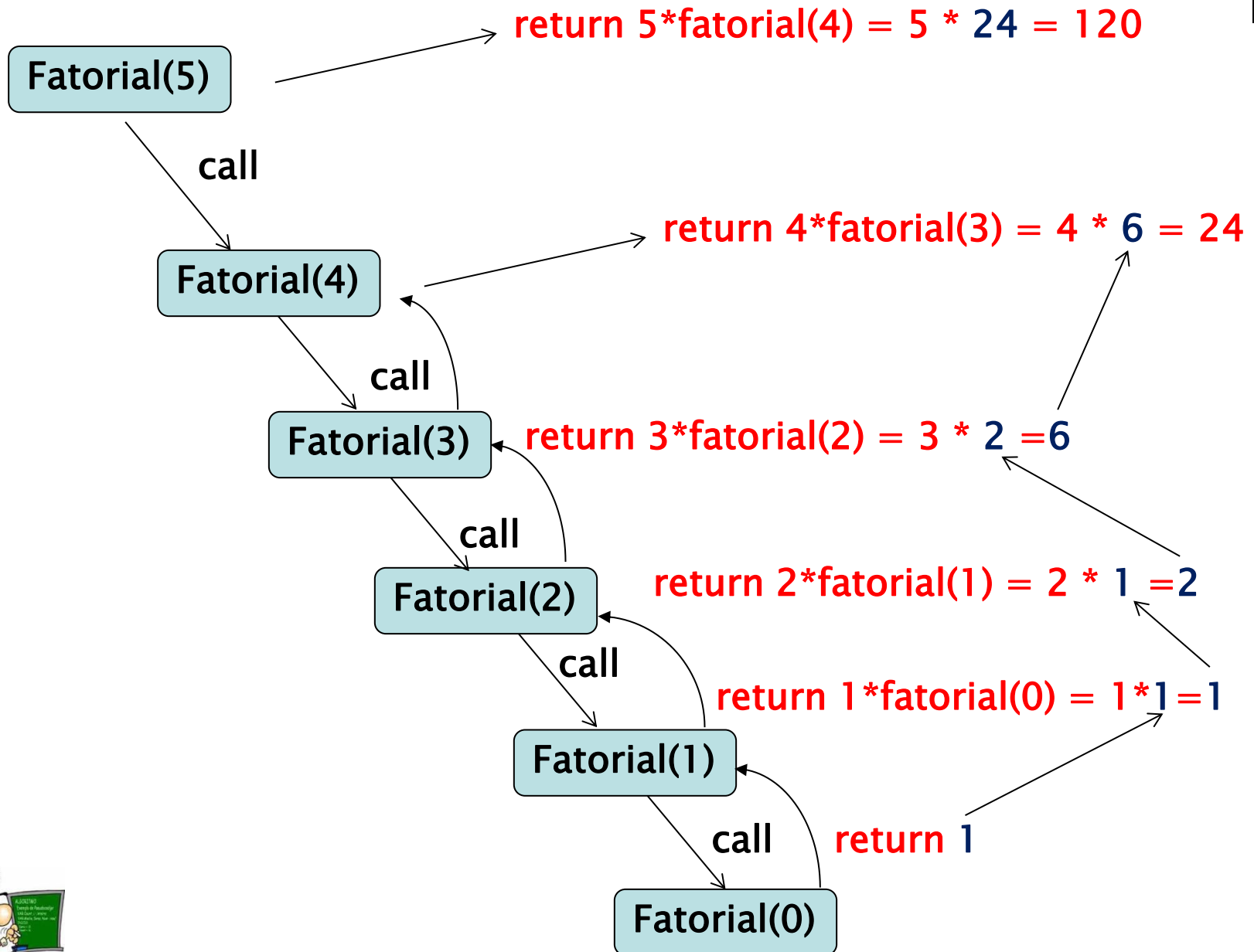
int fatorial(int);
int main() {
    int n=5;
    printf("Fatorial de %d = %d ", n, fatorial(n) );

}

int fatorial(int n) {
    if (n==0)
        return 1;    //caso básico
    else
        return(n*fatorial(n-1)); //caso recursivo
}
```



Trça de Recursão



Série de Fibonacci

- A sucessão de **Fibonacci** ou sequência de **Fibonacci** é uma sequência de números naturais, na qual os primeiros dois termos são **0** e **1**, e cada termo subsequente corresponde à soma dos dois precedentes.
- Os números de **Fibonacci** são, portanto, compostos pela seguinte sequência de números inteiros:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ...



Série de Fibonacci

- Em termos matemáticos, a sequência é definida recursivamente pela fórmula abaixo, sendo os dois primeiros termos $F_0 = 0$ e $F_1 = 1$.

$$F(n) = \begin{cases} 0, & \text{se } n=0 \\ 1, & \text{se } n=1 \\ F(n-1) + F(n-2) & \text{se } n > 1 \end{cases}$$



Série de Fibonacci – Pseudocódigo

Fibonacci(n)

if (n <= 1)

return n ;

else

return **Fibonacci**(n-1) + **Fibonacci**(n-2);



Série de Fibonacci

```
#include <stdio.h>

int fibonacci(int);
int main() {

    int n=10;
    printf("Fibonacci de %d = %d ", n, fibonacci(n) );

}

int fibonacci(int n) {
    if (n <= 1)
        return n;    //caso básico
    else
        return (fibonacci(n-1) + fibonacci(n-2)); //caso recursivo
}
```



Exemplo - 1

```
#include <stdio.h>
int main() {

    int n = 5;
    printf("%d      ", func(n));
    return 0;
}

int func (int n) {

    if (n == 0 )
        return 10;
    else
        return 1 + func(n-1);
}
```



Exemplo - 2

```
#include <stdio.h>

int recursao(int);

int main() {

    printf("%d", recursao (9));
    return 0;
}

int recursao (int n) {

    if (n <= 4 )
        return n*2;
    else {
        return recursao (recursao (n/3));
    }
}
```

