

RECURSIVIDADE

Uma função é recursiva se ela contém uma chamada dela mesmo dentro do próprio corpo.

Essa chamada pode ser feita de duas formas:

- direta – quando a função chama ela mesma
- indireta – quando a função f1 chama outra função f2, e a função f2 contém a chamada da função f1.

A ideia de recursividade pode ser aplicada quando é possível dividir o problema original em duas partes:

- um caso básico, que geralmente é simples de resolver
- uma versão simplificada do problema original

A função recursiva “sabe” como somente como resolver o caso básico, enquanto os dados não corresponderem o caso básico a função chamará ela mesma, mas para um caso simplificado do problema original.

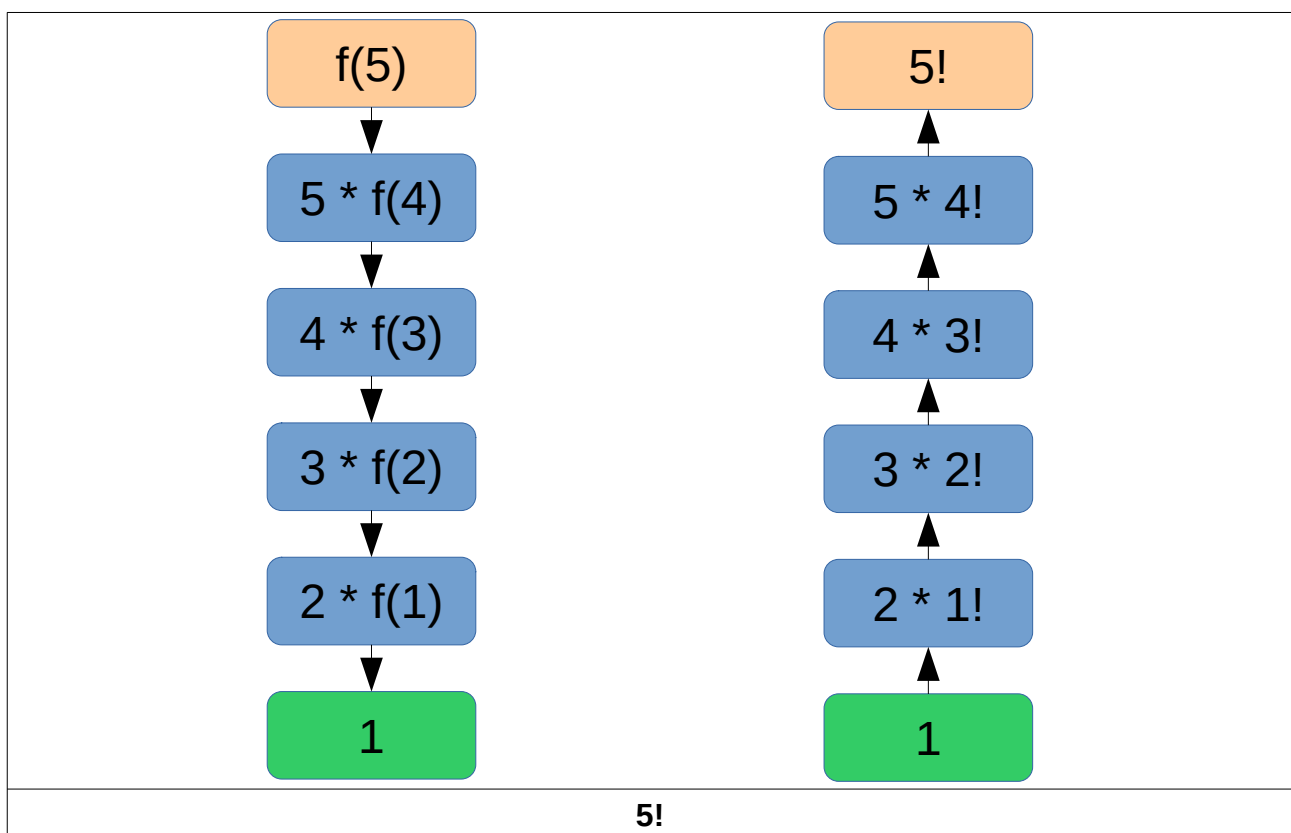
Cada chamada da função recursiva é conhecida como passo de recursão.

Em cada passo de recursão a função retornará um resultado que será utilizado para achar a solução final.

Vamos analisar uma função recursiva que calcula o fatorial de um número.

A fórmula matemática para achar o fatorial de um número n:

$$n! = n * (n - 1) * (n - 2) * \dots * 1$$



Exemplo 1: fatorial (recursivo)

```
1  #include <stdio.h>
2
3  int factorial(int n)
4  {
5      if(n <= 1)
6      {
7          return 1;
8      }
9      return n * factorial(n - 1);
10 }
11 //=====
12
13 int main()
14 {
15     int num;
16
17     printf("\n Digite número: ");
18     scanf("%i", &num);
19
20     if( num >= 0)
21         printf("\n %i! =  %d \n", num, factorial(num));
22     else
23         printf("\n Número invalido! \n");
24
25     return 0;
26 }
```

Digite número: 5
5! = 120

Outro problema que pode ser facilmente resolvido com uso de recursividade é a geração dos números da sequência de Fibonacci.

Sequência de Fibonacci (também chamada de sucessão de Fibonacci), é uma sequência de números inteiros, começando normalmente por 0 e 1, na qual, cada termo subsequente (número de Fibonacci) corresponde a soma dos dois anteriores.

Matematicamente pode ser representada como:

$$\begin{aligned} f(0) &= 0 \\ f(1) &= 1 \\ f(n) &= f(n-1) + f(n-2) \end{aligned}$$

Exemplo 2: Fibonacci (recursivo)

```
1  #include <stdio.h>
2
3  int fibonacci(int n)
4  {
5      if(n == 0)
6      {
7          return 0;
8      }
9      if(n == 1)
10     {
11         return 1;
12     }
13     return fibonacci(n-1) + fibonacci(n-2);
14 }
15 //=====
16
17 int main()
18 {
19     int i, num;
20
21     printf("\n Quantos números da serie Fibonacci devem ser gerados? ");
22     scanf("%i", &num);
23
24     if( num >= 0)
25     {
26         for(i=0; i < num; i++)
27             printf (" %i ", fibonacci(i));
28     }
29     else
30         printf("\n Número invalido! \n");
31
32     return 0;
33 }
34
```

Quantos números da serie Fibonacci devem ser gerados? 10

0 1 1 2 3 5 8 13 21 34

Recursividade vs iteração

Ambas as técnicas têm algumas características em comum, como:

- uso de repetição para solução dos problemas
- condição de parada

De forma geral, todo problema que pode ser resolvido usando recursividade também poderá ser resolvido usando iteração.

A desvantagem de recursividade é elevado custo computacional: em toda chamada recursiva uma cópia da função é criada.

Uma solução recursiva é indicada quando ela representa a solução de forma mais transparente.

DIRETIVAS DE PRE-PROCESSADOR

O preprocessador não faz parte do compilador.

Ele é responsável por uma etapa específica no processo de compilação de código.

De forma simplificada a função de preprocessador consiste em fazer as substituições no código de acordo com as regras definidas no programa.

Esse processamento é feito antes mesmo do próprio compilador entrar em ação.

As regras para pre processamento são definidas usando as diretivas de preprocessador:

Diretiva	Funcionalidade
#include <filename>	incluir um arquivo (header) que se encontra nas pastas padrão do sistema
#include "filename"	incluir um arquivo (header) que se encontra na pasta atual
#define	criar uma constante simbólica ou um macros
#undef	revoga a definição
#ifdef	retorna true se o macros é definido
#ifndef	retorna true se o macros não é definido
#if	verifica se a condição é verdadeira
#else	alternativa para #if
#elif	#else e #if em uma única expressão
#endif	termino da diretiva condicional #if

Exemplo 3: #define

```
1  #include <stdio.h>
2  #define PI 3.14159
3  #define CIRCLE_AREA(x) (PI*(x)*(x))
4  //=====
5  int main()
6  {
7      float area;
8      float n;
9
10     printf("\n Digite raio do circulo: ");
11     scanf("%f",&n);
12
13     area = CIRCLE_AREA(n);
14     printf("\n Area = %.2f \n", area);
15
16     return 0;
17 }
```

Digite raio do circulo: 1

Area = 3.14

ARQUIVOS .h

Com aumento da complexidade dos problemas a quantidade de linhas de código tende a aumentar.

Para deixar os programas mais legíveis e mais bem estruturados costuma-se usar arquivos **.h (header)**.

Nesse tipo de arquivo costuma-se colocar as declarações de funções e outros parâmetros globais do sistema como variáveis globais e macros (caso existam).

Os arquivos **.h** que contêm as declarações de funções podem ser compartilhados por vários programas fonte (**source files**).

Existem dois tipos de arquivos **.h**:

- as bibliotecas padrão que vêm com o próprio compilador
- os arquivos criados pelo próprio programador

Incluir um arquivo **.h** em um programa é equivalente a copiar o conteúdo dele em programa.

As principais vantagens de estruturar o programa em arquivos fonte e arquivos **.h**:

- facilidade na manutenção do código
- o desenvolvimento colaborativo se torna mais simples

Exemplo 4: Programa dividido em dois arquivos (.c e .h)**factorial.h**

```
1 // factorial.h
2
3 int factorial(int n)
4 {
5     if(n <= 1)
6     {
7         return 1;
8     }
9     return n * factorial(n - 1);
10 }
```

El_04_header_v01.c

```
1 #include <stdio.h>
2 #include "factorial.h"
3
4 #define N 10
5 //=====
6 int main()
7 {
8     int i;
9     int v[N];
10
11     for (i = 0; i<10; i++)
12         v[i]=factorial(i);
13
14     for (i = 0; i<10; i++)
15         printf("\n %i! = %i ", i, v[i]);
16
17     return 0;
18 }
```

```
0! = 1
1! = 1
2! = 2
3! = 6
4! = 24
5! = 120
6! = 720
7! = 5040
8! = 40320
9! = 362880
```

Observações:

- os dois arquivos (*.c e *.h) devem estar dentro da mesma pasta.
- o tamanho do vetor foi definido com uso da diretiva de pré-processador #define, que é uma aplicação bastante comum dessa diretiva.

Inclusão de arquivos .h

Quando o projeto é composto por vários arquivos pode acontecer que, devido à complexidade do código, o mesmo arquivo **.h** será incluído mais de uma vez em um programa fonte.

Nesse caso o compilador vai incluir o arquivo duas vezes que vai resultar em error.

Para prevenir esse tipo de situação costuma-se fazer o controle da inclusão de arquivos **.h** da seguinte forma:

```
#ifndef HEADER_FILE
#define HEADER_FILE

#include "factorial.h"

#endif
```