RECURSIVIDADE

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

Essa chamada pode ser feita de duas formas:

- direta – quando a função chama ela mesmo
- indireta – quando a a função f1 chama outra função f2, e a função f2 contem 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

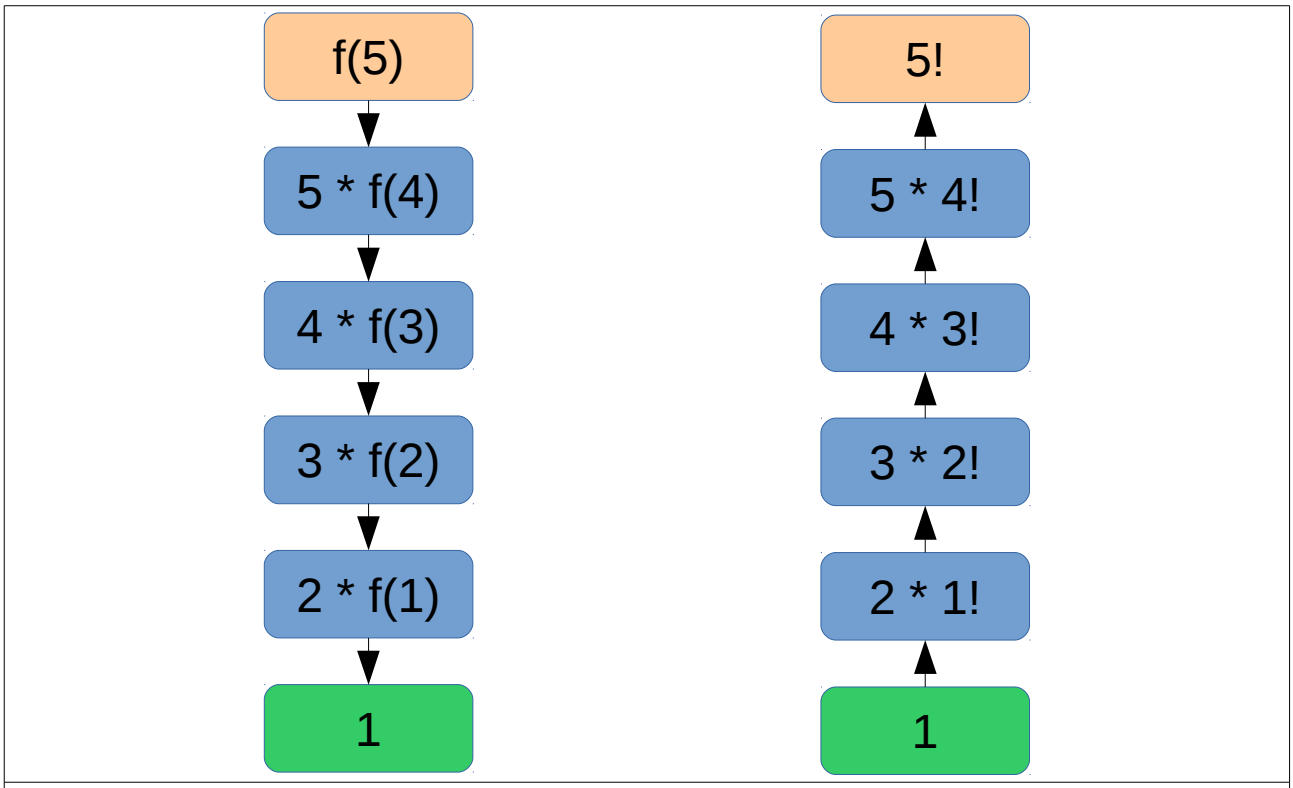
A função recursiva “sabe” como somente como resolver o caso básico, enquanto os dados não corresponderem o caso básico a função chamará ela mesma, mas para um caso simplificado do problema original.

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 formula matemática para achar o fatorial de um número n:

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



5!

Exemplo 1 (2p): fatorial (recursivo)

```c
#include <stdio.h>

int factorial(int n)
{
   if(n <= 1)
   {
      return 1;
   }
   return n * factorial(n - 1);
}
//=========================================================

int main()
{
   int num;

   printf("\n Digite número: ");
   scanf("%i", &num);

   if( num >= 0)
    printf("\n %i! =  %d \n", num, factorial(num));
   else
    printf("\n Número invalido! \n");

   return 0;
}
```

```
 Digite número: 5
5! =  120
```

Outro problema que pode ser facilmente resolvido com uso de recursividade é a geração dos números da sequencia 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 (numero de Fibonacci) corresponde a soma dos dois anteriores.

Matematicamente pode ser representada como:

```
f(0) = 1
f(1) = 1
f(n) = f(n-1) + f(n-2)
```

Exemplo 2 (2p): Fibonacci (recursivo)

```c
#include <stdio.h>

int fibonacci(int n)
{
   if(n == 0)
   {
      return 0;
   }
   if(n == 1)
   {
      return 1;
   }
   return fibonacci(n-1) + fibonacci(n-2);
}
//=======================================================

int main()
{
   int i, num;

   printf("\n Quantos números da serie Fibonacci devem ser gerados? ");
   scanf("%i", &num);

   if( num >= 0)
    {
      for(i=0; i < num; i++)
       printf (" %i  ", fibonacci(i));

    }
   else
    printf("\n Número invalido! \n");

   return 0;
}
```

```
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 tem 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 copia 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 pop uma etapa especifica 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 defendias 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 (2p): #define

```c
#include <stdio.h>
#define PI 3.14159
#define CIRCLE_AREA(x) (PI*(x)*(x))
//============================================================
int main()
{
    float area;
    float n;

    printf("\n Digite raio do circulo: ");
    scanf("%f",&n);

    area = CIRCLE_AREA(n);
    printf("\n Area = %.2f \n", area);

    return 0;
}
```

```
Digite raio do circulo: 1

Area = 3.14
```

ARQUIVOS .h

Com aumento da complexidade dos problemas a quantidade de linhas de código tende 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 contem as declarações de funções podem ser compartilhados por vários programas fonte (**source files**).

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

- as bibliotecas padrão que vem 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 (2p): Programa dividido em dois arquivos (.c e .h)

factorial.h

```c
// factorial.h

int factorial(int n)
{
    if(n <= 1)
    {
        return 1;
    }
    return n * factorial(n - 1);
}
```

El_04_header_v01.c

```c
#include <stdio.h>
#include"factorial.h"

#define N 10
//============================================================
int main()
{
    int i;
    int v[N];

    for (i = 0; i<10; i++)
      v[i]=factorial(i);

    for (i = 0; i<10; i++)
      printf("\n %i! = %i ", i, v[i]);

    return 0;
}
```

```
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 preprocessador #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 a 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
```