INSTITUTO MAUÁ DE TECNOLOGIA

MAUÁ

Recursividade

O que é recursividade?

- **Uma definição**

  Recursividade:

  Veja Recursividade. Se não entender, **veja Recursão**.

  Recursão:

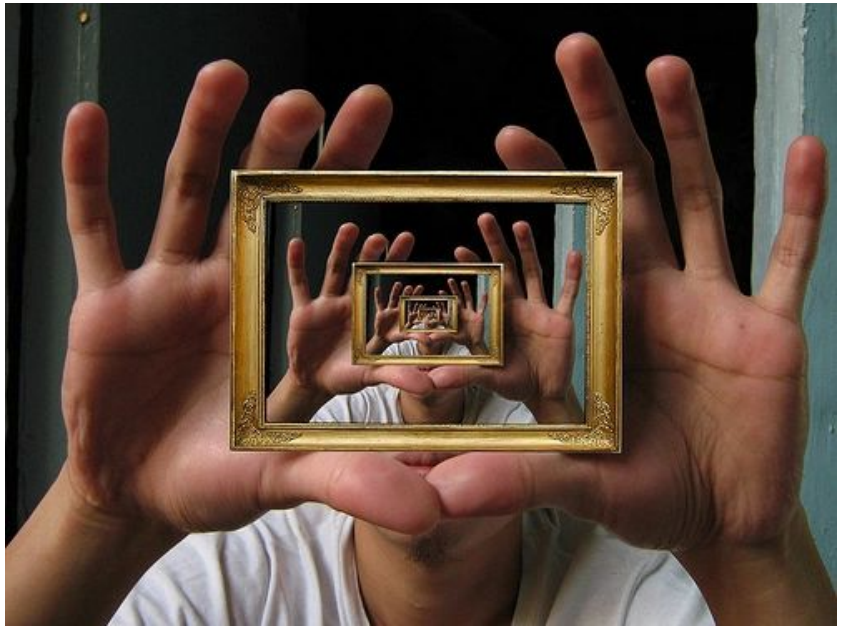
  Veja Recursividade.

O que é recursividade?

- **Recursividade é …**
  - Um **objeto** é **recursivo** se ele **consiste parcialmente** ou é **definido** em termos **dele próprio**.
  - Com a **recursão**, **resolve-se** um **problema** pela **resolução** de **instâncias menores** do **mesmo problema**.

O que é recursividade?

- Imagens recursivas



https://joseartrivera.wordpress.com/2013/05/01/infinite-recursion/

- **Função fatorial**
  - O **fatorial** de um número natural ***n*** é definido assim:

$$n! = \begin{cases} 1, \text{ se } n = 0 \\ n.(n-1)!, \text{ se } s > 0 \end{cases}$$

Recursividade na Matemática

- **Função fatorial**
  - Para **calcular** o **fatorial** de um número, procede-se assim:
    - Para calcular $4!$ é necessário calcular $4.(4-1)! = 4.3!$
    - Para calcular $3!$ é necessário calcular $3.(3-1)! = 3.2!$
    - Para calcular $2!$ é necessário calcular $2.(2-1)! = 2.1!$
    - Para calcular $1!$ é necessário calcular $1.(1-1)! = 1.0!$
    - Mas $0!$ vale exatamente 1 (definição matemática)

Recursividade na Matemática

- **Função fatorial**
  - **Então**:
    - Como $0!$ vale 1, então $1! = 1.0! = 1.1 = 1$
    - Como $1!$ vale 1, então $2! = 2.1! = 2.1 = 2$
    - Como $2!$ vale 2, então $3! = 3.2! = 3.2 = 6$
    - Como $3!$ vale 6, então $4! = 4.3! = 4.6 = 24$
    - **Logo**, $4!$ vale 24.

Recursividade na Matemática

- **Elementos de uma definição recursiva**
  - Deve **haver um ou mais casos base**, onde se **calcula** uma **solução diretamente**, **sem recursão**;
  - Cada **cálculo recursivo** de uma **função** deve ser **realizado** sobre uma **instância menor** do **mesmo problema**, que, **eventualmente**, alcançará o **caso base**.

Elementos de uma definição recursiva

$$n! = \begin{cases} 1, \text{ se } n = 0 \\ n \times (n-1)!, \text{ se } s > 0 \end{cases}$$

caso base

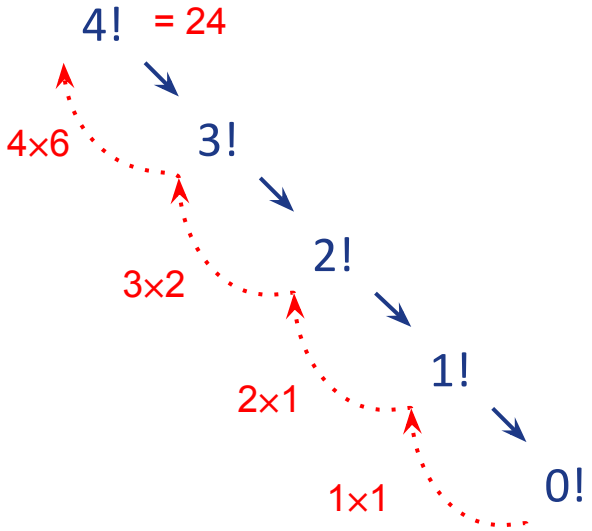
cálculo recursivo sobre uma instância menor (n-1)

Recursividade na Matemática

- **Árvore de recursão**
  - Uma forma de acompanhar o cálculo de uma função recursiva é desenhar uma **árvore de recursão**:
    1. No nó raiz (topo) se encontra a expressão a ser calculada;
    2. Desenha-se um ou mais nós representando uma ou mais expressões recursivas que definem as partes que constituem a expressão do nó anterior;

- **Árvore de recursão**
  - (cont.)
    3. Ligam-se os nós por meio de linhas simples;
    4. Repetir os passos 1-3 até se chegar a nós contendo apenas casos base - folhas da árvore;
    5. O valor calculado é obtido percorrendo a árvore de "baixo para cima", levando os valores calculados da base até a raiz e aplicando as operações necessárias.

- **Árvore de recursão**
  - **Exemplo**: 4!



- **Mais exemplos**
  - Multiplicação de números naturais (a, b ∈ N)

$$m(a,b) = \begin{cases} 0, \text{ se } b = 0 \\ a + m(a, b-1), \text{ se } b \neq 0 \end{cases}$$

  - Número de Fibonacci (n ∈ N)

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

Exercícios

- **Calcular manualmente**
  - 6!
  - m(5,3)
  - F(9)

- **Lembrando o conceito de função em C**
  - Uma **definição de função em C** deve possuir:
    - Um tipo de **retorno**;
    - Um **identificador** (nome);
    - Uma lista de **parâmetros formais** (lista de zero ou mais **declarações** $tipo_{par}$ $nome_{par}$, separados por vírgula) dentro de "(" e ")";
    - Bloco definindo o **corpo da função** ("{" e "}").

Lembrando o conceito de função em C

Função **fatorial (não recursiva)**

```
int fatorial(int n) {
    int fat = 1;
    int i;
    for (i = 1; i <= n; i++) {
        fat *= i;
    }
    return fat;
}
```

tipo de retorno

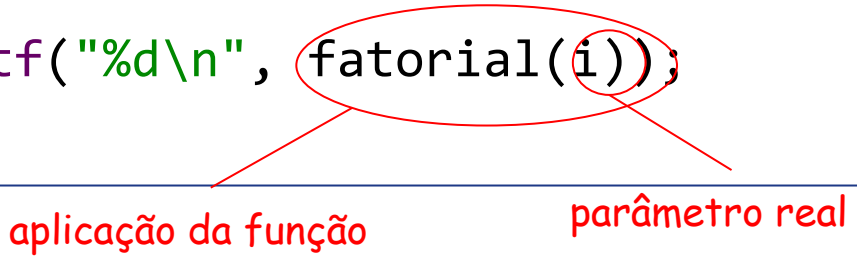
nome

parâmetro

corpo

- **Lembrando o conceito de função em C**
  - A **chamada** (ou **execução**) de uma **função** se dá pela **aplicação** da **expressão** da **função** a **parâmetros reais**, de **tipos consistentes** aos **parâmetros formais** da função:

```
...
    printf("%d\n", fatorial(i));
...
```



❖ Lembrando o conceito de função em C

➢ O que acontece quando uma função é chamada?

```
int main(void) {
    int i;
    for (i = -1; i <= 10; i++) {
        printf("%d\n", fatorial(i));
    }
    return 0;
}
```

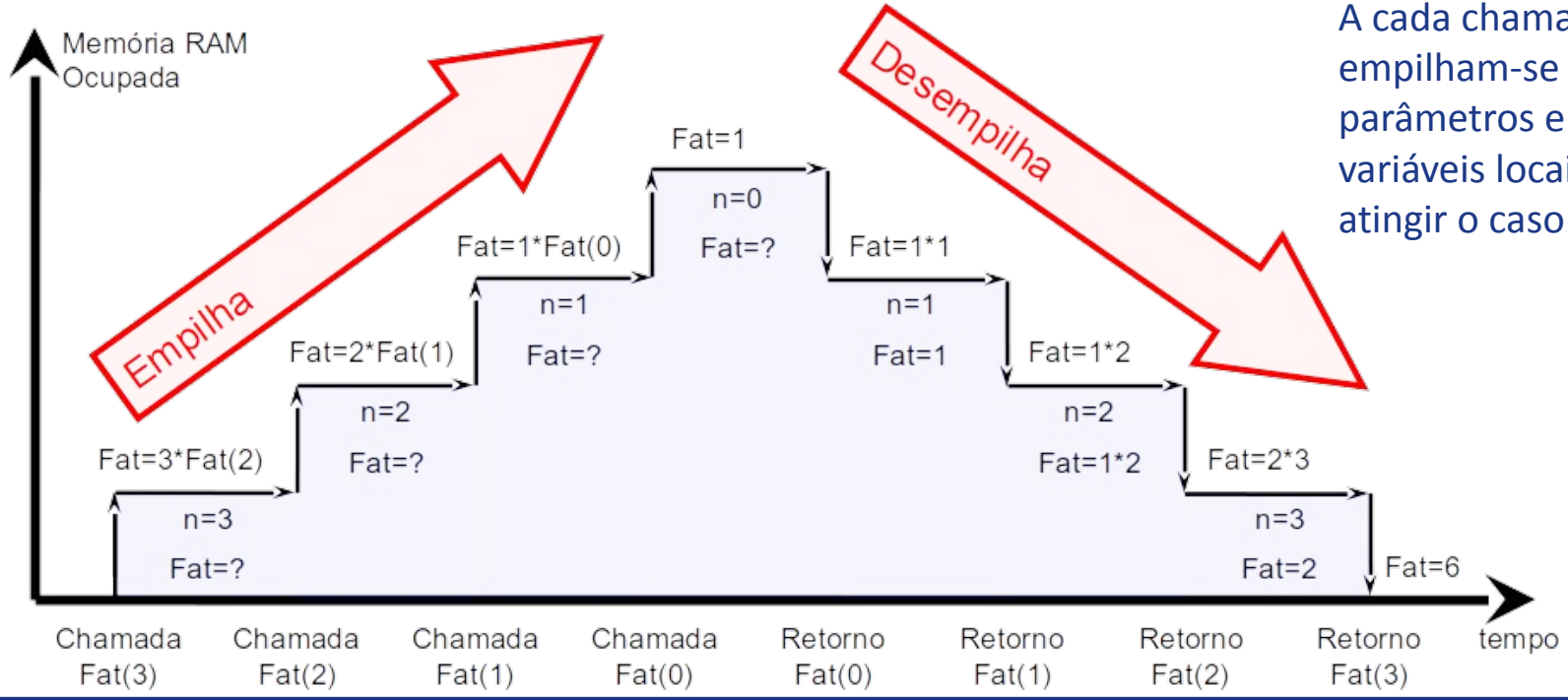
1. Os **parâmetros reais** são **copiados** aos **parâmetros formais**;
2. O **endereço** da **próxima instrução** e os **parâmetros formais** da **função** e suas **variáveis locais** são **armazenadas** em uma área denominada **pilha**, dentro do espaço de memória do programa;
3. A **função** é **executada**;
4. **Depois** da sua execução, suas **variáveis locais** e **parâmetros** são **removidas** da **pilha**, bem como o **endereço** do **próxima instrução**;
5. O **código segue** para **executar** a **próxima instrução**.

- **Versão recursiva da função fatorial em C**
  - Uma **função recursiva** em **C** possui uma **definição** como qualquer outra função, exceto que **sempre** haverá uma **chamada** dela **própria**:

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

chamada recursiva

Execução da função fatorial recursiva



A cada chamada, empilham-se parâmetros e variáveis locais, até se atingir o caso base

Recursividade em Computação

- **Quando usar funções recursivas?**
  - É **apropriada** quando o **problema** em questão é **definido** em **termos recursivos ou quando** os **dados** a serem **tratados** são definidos em **termos recursivos**;
  - **Todo programa recursivo** pode ser **convertido** a um **programa iterativo**. Em **alguns casos**, é **necessário** o uso **explícito** de uma **estrutura** de **dados** do tipo **pilha** (**exemplo**: exploração com *backtracking*).

Recursividade em Computação

- **Vantagens e desvantagens**
  - **Vantagens**
    - Grande **facilidade** na **implementação** de algoritmos com **características recursivas**;
  - **Desvantagens**
    - **Aumenta** o **consumo** de **memória** (pilha);
    - **Tempo** de **execução** é **maior** do que em algoritmos iterativos (chamar funções gasta tempo!).

Exercícios

- **Implementar as seguintes funções recursivas em C**
  1. Multiplicação de números naturais
  2. Função de Fibonnaci