

# CCP130

# Desenvolvimento de Algoritmos

Prof. Danilo H. Perico



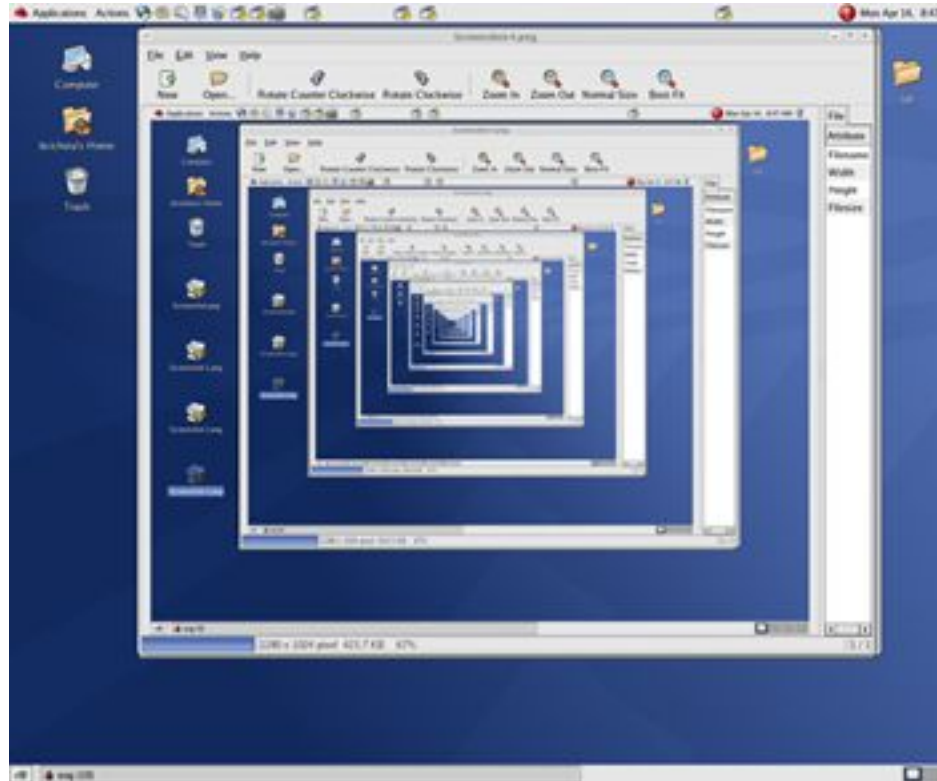
# Recursividade

# Recursividade



- Recursividade é um termo usado de maneira mais geral para descrever o **processo de repetição** de um objeto de um jeito similar ao que já fora mostrado
- Um bom exemplo disso são as imagens repetidas que aparecem quando **dois espelhos são apontados um para o outro**

# Recursividade



# Recursividade



# Recursividade na Programação



- Em termos gerais, a recursão pode ser considerada como um processo de **repetição de um procedimento ou função**
- Portanto, de maneira bem simplista, pode ser definida como um **procedimento ou função que chama a si mesmo(a)**

# Funções Recursivas



- Uma função é recursiva se definida em seus próprios termos
- Toda recursão é composta por:
  - **Um caso base:** um problema que pode ser solucionado facilmente. Por exemplo, é simples fazermos a soma de um vetor que tem um único elemento.
  - **Uma ou mais chamadas recursivas:** em que a função define-se em termos de si própria, tentando convergir para o caso base.

# Funções Recursivas - Exemplo



- **Pergunta:** um aluno tem uma pergunta (*um problema a ser resolvido*)
- O professor sabe a resposta
- Pergunte para alguém que você pode tocar sem sair de sua cadeira e que esteja mais próximo do professor



# Funções Recursivas - Exemplo



- **Pergunta:** um aluno tem uma pergunta (*um problema a ser resolvido*)
- O professor sabe a resposta - **CASO BASE**
- Pergunte para alguém que você pode tocar sem sair de sua cadeira e que esteja mais próximo do professor - **PASSO RECURSIVO**

# Funções Recursivas - Exemplo



Chamada  
Original

# Funções Recursivas - Exemplo



Chamada  
Original

1ª  
Chamada  
Recursiva

# Funções Recursivas - Exemplo



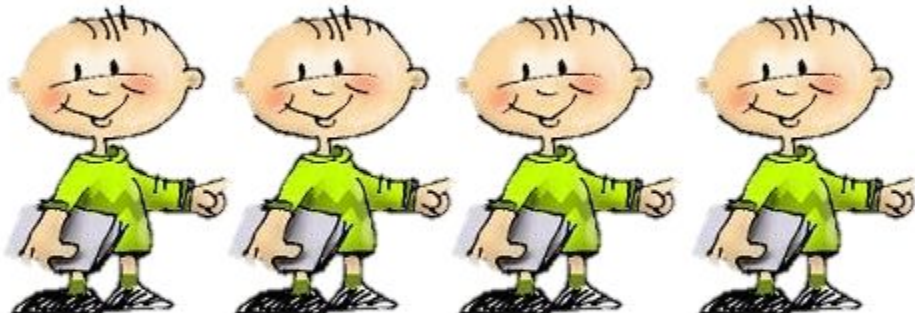
Chamada  
Original

1<sup>a</sup>  
Chamada  
Recursiva

2<sup>a</sup>  
Chamada  
Recursiva

# Funções Recursivas - Exemplo

?



Chamada  
Original

1<sup>a</sup>  
Chamada  
Recursiva

2<sup>a</sup>  
Chamada  
Recursiva

3<sup>a</sup>  
Chamada  
Recursiva

# Funções Recursivas - Exemplo



?



Chamada  
Original

1ª  
Chamada  
Recursiva

2ª  
Chamada  
Recursiva

3ª  
Chamada  
Recursiva

4ª  
Chamada  
Recursiva

# Funções Recursivas - Exemplo



Chamada  
Original

1ª  
Chamada  
Recursiva

2ª  
Chamada  
Recursiva

3ª  
Chamada  
Recursiva

4ª  
Chamada  
Recursiva

5ª  
Chamada  
Recursiva

# Funções Recursivas - Exemplo



Chamada  
Original



1<sup>a</sup>  
Chamada  
Recursiva



2<sup>a</sup>  
Chamada  
Recursiva



3<sup>a</sup>  
Chamada  
Recursiva



4<sup>a</sup>  
Chamada  
Recursiva



5<sup>a</sup>  
Chamada  
Recursiva

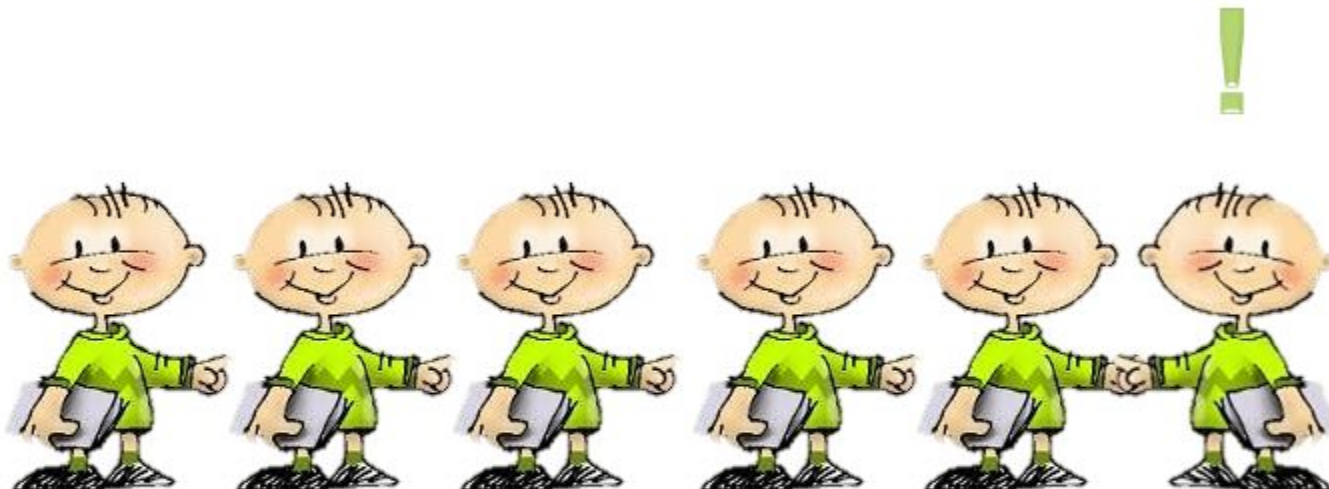
?



6<sup>a</sup>  
Chamada  
Recursiva  
(Caso base)



# Funções Recursivas - Exemplo



Chamada  
Original

1<sup>a</sup>  
Chamada  
Recursiva

2<sup>a</sup>  
Chamada  
Recursiva

3<sup>a</sup>  
Chamada  
Recursiva

4<sup>a</sup>  
Chamada  
Recursiva

5<sup>a</sup>  
Chamada  
Recursiva

# Funções Recursivas - Exemplo



Chamada  
Original

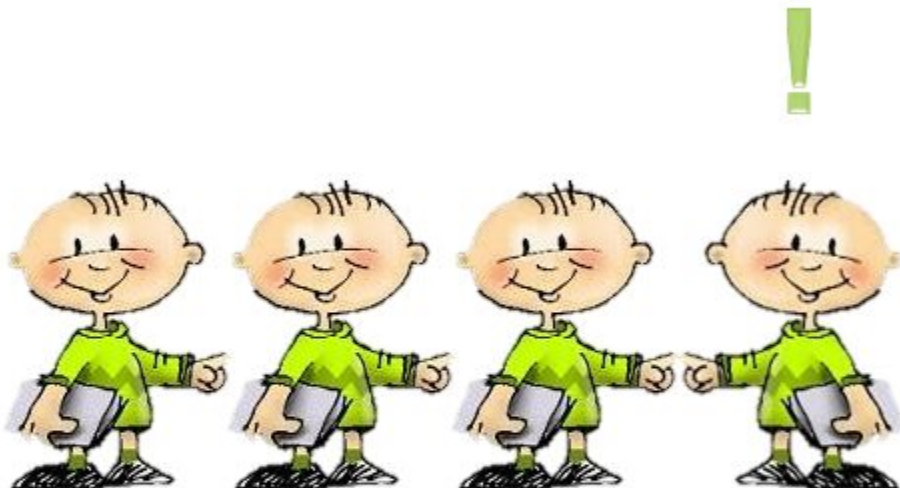
1ª  
Chamada  
Recursiva

2ª  
Chamada  
Recursiva

3ª  
Chamada  
Recursiva

4ª  
Chamada  
Recursiva

# Funções Recursivas - Exemplo



Chamada  
Original

1<sup>a</sup>  
Chamada  
Recursiva

2<sup>a</sup>  
Chamada  
Recursiva

3<sup>a</sup>  
Chamada  
Recursiva

# Funções Recursivas - Exemplo



Chamada  
Original

1<sup>a</sup>  
Chamada  
Recursiva

2<sup>a</sup>  
Chamada  
Recursiva

# Funções Recursivas - Exemplo



Chamada  
Original

1ª  
Chamada  
Recursiva

# Funções Recursivas - Exemplo



Chamada  
Original

# Recursividade na Programação



- A recursão consiste em **dividir um problema em subproblemas** do mesmo tipo
- Como técnica de programação, isto se denomina **divisão e conquista**, e constitui a chave para o desenvolvimento de muitos algoritmos importantes


# Recursividade na Programação



- Para se definir um programa recursivo, primeiramente se define uma quantidade limitada de **casos base**, que **são situações triviais**, ou seja, não envolvem recursão.



# Recursividade na Programação - Exemplo

- 
- Resolver o **Fatorial** de um número natural ***n***
    - *O que nós sabemos para resolver este problema?*

# Recursividade na Programação - Exemplo

- Resolver o **Fatorial** de um número natural  **$n$** 
  - *O que nós sabemos para resolver este problema?*
    - **$0! = 1$**
    - Dado  **$n$** ,  **$n > 0$** , seu fatorial é  **$n(n-1)!$** 
      - Exemplo:  **$5! = 5 \times 4!$**

# Recursividade na Programação - Exemplo

- Resolver o **Fatorial** de um número natural  **$n$** 
  - *O que nós sabemos para resolver este problema?*
    - **$0! = 1$**
    - Dado  **$n$** ,  **$n > 0$** , seu fatorial é  **$n(n-1)!$**
  - Assim,  **$n!$**  É igual a:
    - **$n \times (n-1) \times (n-2) \times (n-3) \times \dots \times (n-n)!$**
    - **$(n-n)!$**  é o caso base

# Recursividade na Programação - Exemplo

- Função recursiva para Fatorial:

```
1  #include <stdio.h>
2
3  int fatorial(int n){
4      if(n<=0)
5          return 1;
6      else
7          return (n * fatorial(n-1));
8  }
9
10 int main(void){
11     printf("%d", fatorial(5));
12 }
```

A função  
chama ela  
própria

# Recursividade na Programação - Exemplo

```
1  #include <stdio.h>
2
3  int fatorial(int n){
4      if(n<=0)
5          return 1;
6      else
7          return (n * fatorial(n-1));
8  }
9
10 int main(void){
11     printf("%d", fatorial(5));
12 }
```

n=5

int fatorial(5);

# Recursividade na Programação - Exemplo

```
1  #include <stdio.h>
2
3  int fatorial(int n){
4  if(n<=0)  falso
5      return 1;
6  else
7      return (n * fatorial(n-1));
8  }
9
10 int main(void){
11     printf("%d", fatorial(5));
12 }
```

n=5

int fatorial(5);

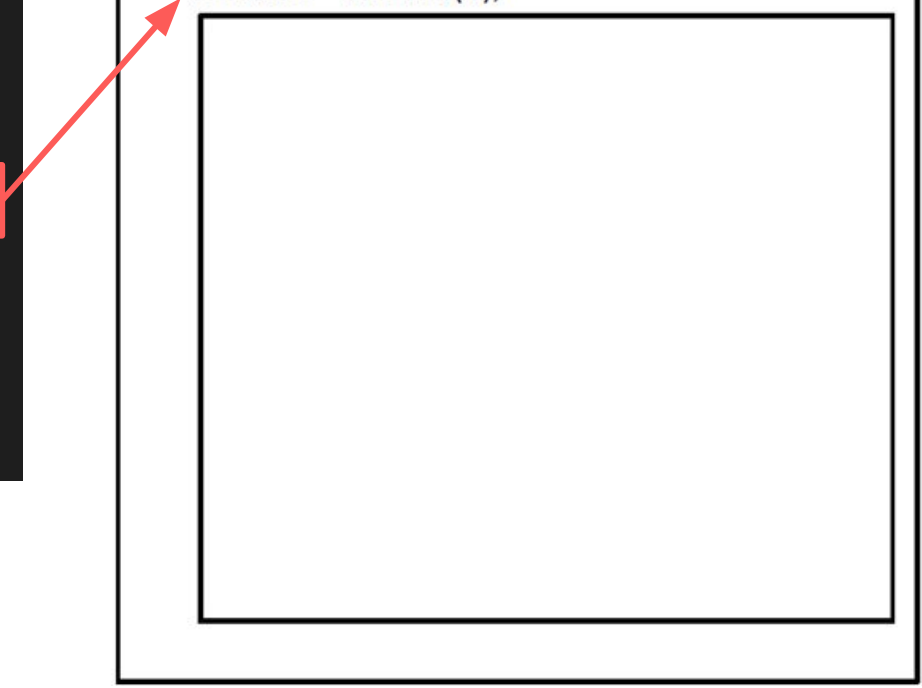
# Recursividade na Programação - Exemplo

```
1  #include <stdio.h>
2
3  int fatorial(int n){
4      if(n<=0)
5          return 1;
6      else
7          return (n * fatorial(n-1));
8  }
9
10 int main(void){
11     printf("%d", fatorial(5));
12 }
```

**n=5**

int fatorial(5);

return 5 \* fatorial(4);



# Recursividade na Programação - Exemplo

```
1  #include <stdio.h>
2
3  int fatorial(int n){
4  if(n<=0) false
5      return 1;
6  else
7      return (n * fatorial(n-1));
8  }
9
10 int main(void){
11     printf("%d", fatorial(5));
12 }
```

n=4

int fatorial(5);

return 5 \* fatorial(4);



# Recursividade na Programação - Exemplo

```
1  #include <stdio.h>
2
3  int fatorial(int n){
4      if(n<=0)
5          return 1;
6      else
7          return (n * fatorial(n-1));
8  }
9
10 int main(void){
11     printf("%d", fatorial(5));
12 }
```

**n=4**

int fatorial(5);

return 5 \* fatorial(4);

return 4 \* fatorial(3);

# Recursividade na Programação - Exemplo

```
1  #include <stdio.h>
2
3  int fatorial(int n){
4  if(n<=0) false
5      return 1;
6  else
7      return (n * fatorial(n-1));
8  }
9
10 int main(void){
11     printf("%d", fatorial(5));
12 }
```

**n=3**

int fatorial(5);

return 5 \* fatorial(4);

return 4 \* fatorial(3);

# Recursividade na Programação - Exemplo

```
1  #include <stdio.h>
2
3  int fatorial(int n){
4      if(n<=0)
5          return 1;
6      else
7          return (n * fatorial(n-1));
8  }
9
10 int main(void){
11     printf("%d", fatorial(5));
12 }
```

**n=3**

int fatorial(5);

return 5 \* fatorial(4);

return 4 \* fatorial(3);

return 3 \* fatorial(2);

# Recursividade na Programação - Exemplo

```
1  #include <stdio.h>
2
3  int fatorial(int n){
4  if(n<=0)  falso
5      return 1;
6  else
7      return (n * fatorial(n-1));
8  }
9
10 int main(void){
11     printf("%d", fatorial(5));
12 }
```

n=2

int fatorial(5);

return 5 \* fatorial(4);

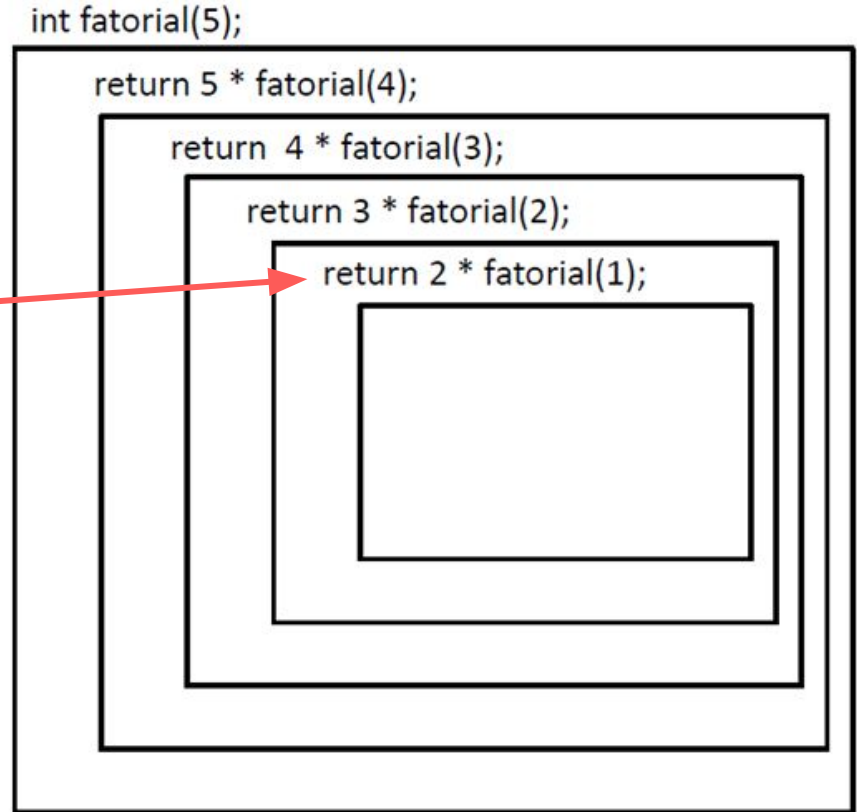
return 4 \* fatorial(3);

return 3 \* fatorial(2);

# Recursividade na Programação - Exemplo

```
1  #include <stdio.h>
2
3  int fatorial(int n){
4      if(n<=0)
5          return 1;
6      else
7          return (n * fatorial(n-1));
8  }
9
10 int main(void){
11     printf("%d", fatorial(5));
12 }
```

**n=2**



# Recursividade na Programação - Exemplo

```
1  #include <stdio.h>
2
3  int fatorial(int n){
4  if(n<=0) falso
5      return 1;
6  else
7      return (n * fatorial(n-1));
8  }
9
10 int main(void){
11     printf("%d", fatorial(5));
12 }
```

n=1

int fatorial(5);

return 5 \* fatorial(4);

return 4 \* fatorial(3);

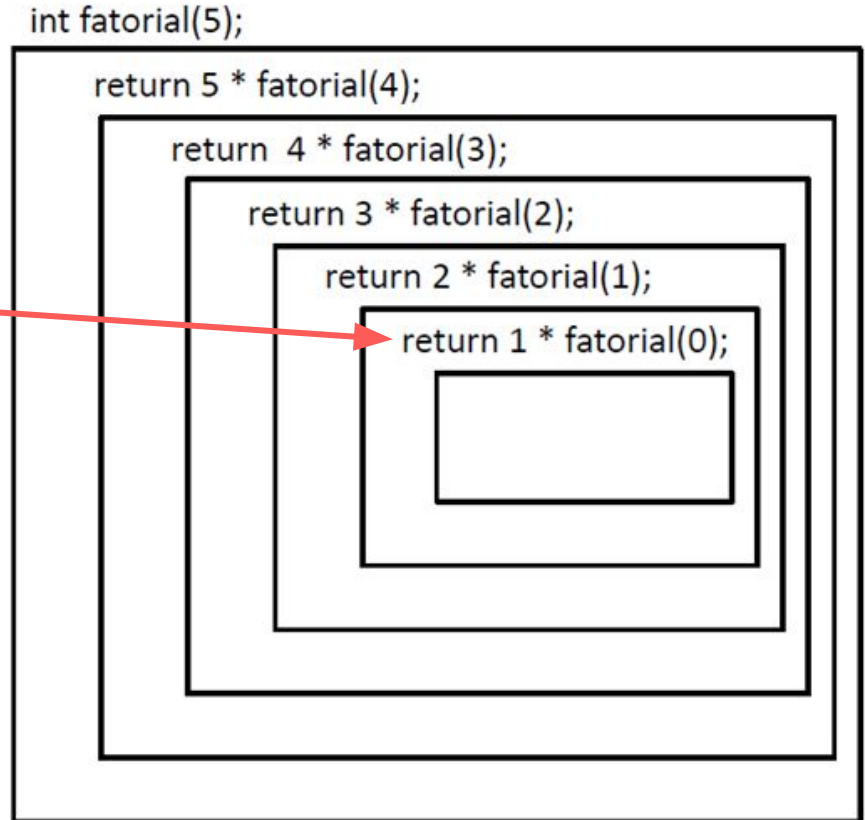
return 3 \* fatorial(2);

return 2 \* fatorial(1);

# Recursividade na Programação - Exemplo

```
1  #include <stdio.h>
2
3  int fatorial(int n){
4      if(n<=0)
5          return 1;
6      else
7          return (n * fatorial(n-1));
8  }
9
10 int main(void){
11     printf("%d", fatorial(5));
12 }
```

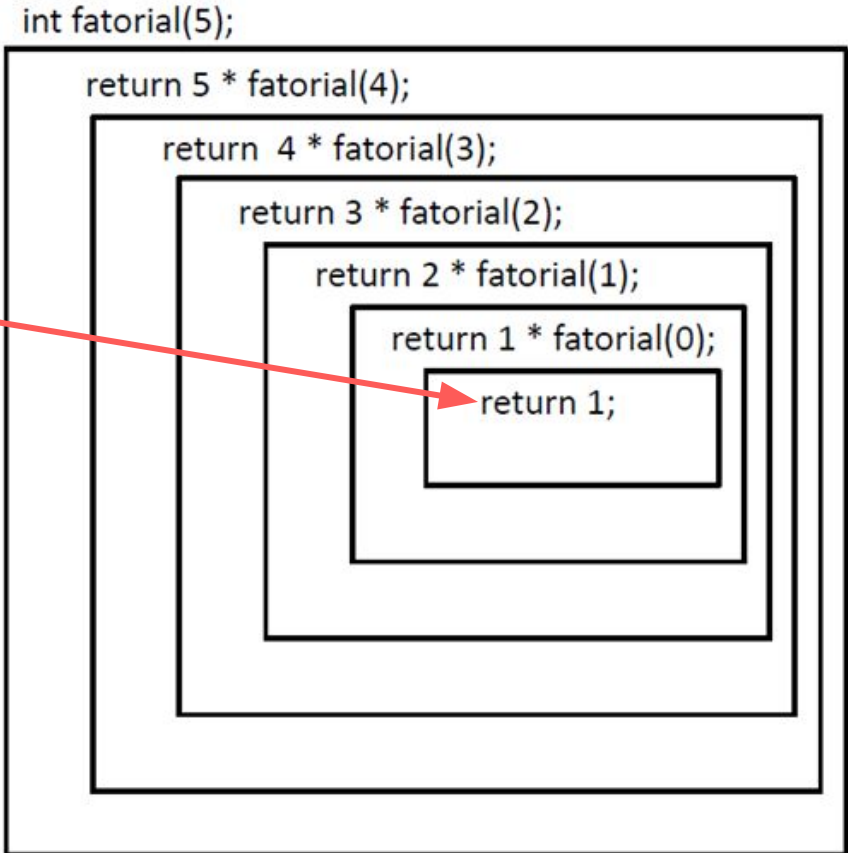
**n=1**



# Recursividade na Programação - Exemplo

```
1  #include <stdio.h>
2
3  int fatorial(int n){
4      if(n<=0) true
5          return 1;
6      else
7          return (n * fatorial(n-1));
8  }
9
10 int main(void){
11     printf("%d", fatorial(5));
12 }
```

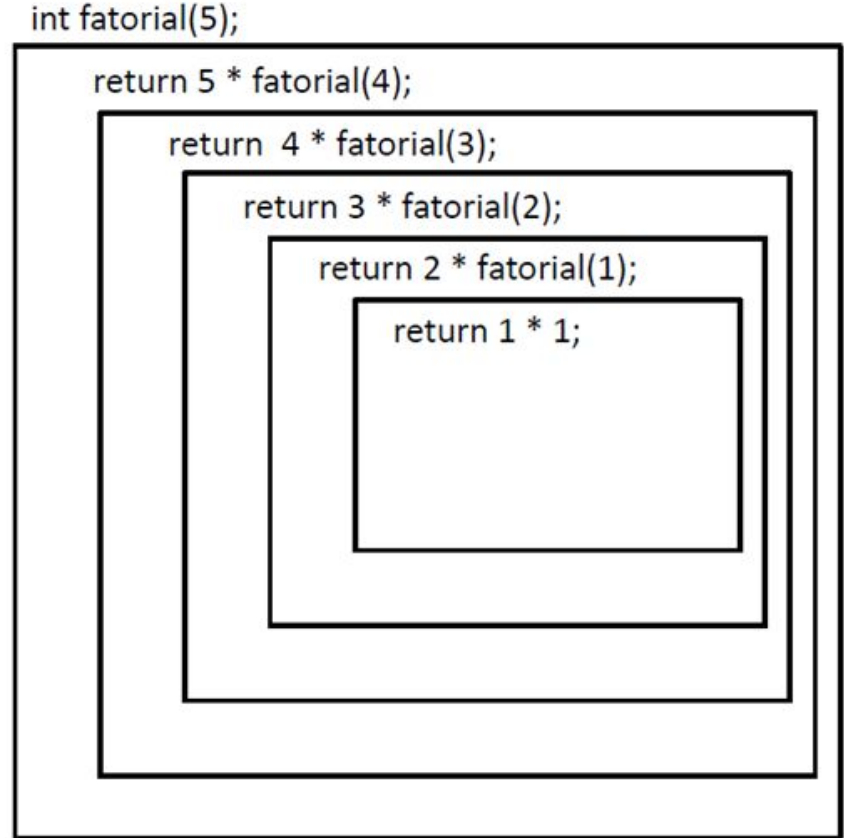
**n=0**





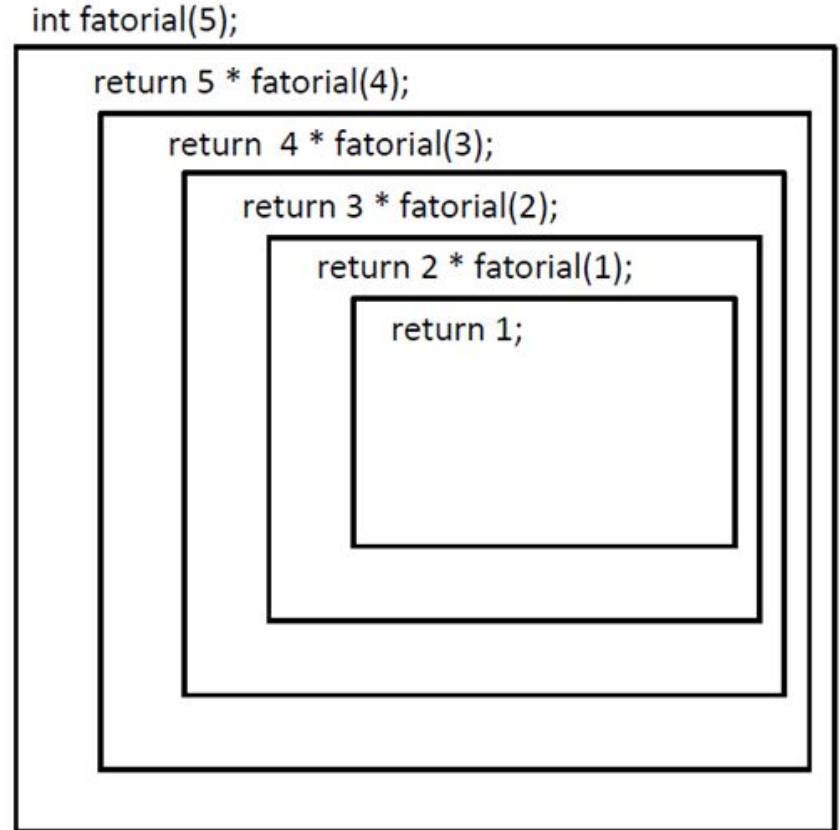
# Recursividade na Programação - Exemplo

```
1  #include <stdio.h>
2
3  int fatorial(int n){
4      if(n<=0)
5          return 1;
6      else
7          return (n * fatorial(n-1));
8  }
9
10 int main(void){
11     printf("%d", fatorial(5));
12 }
```



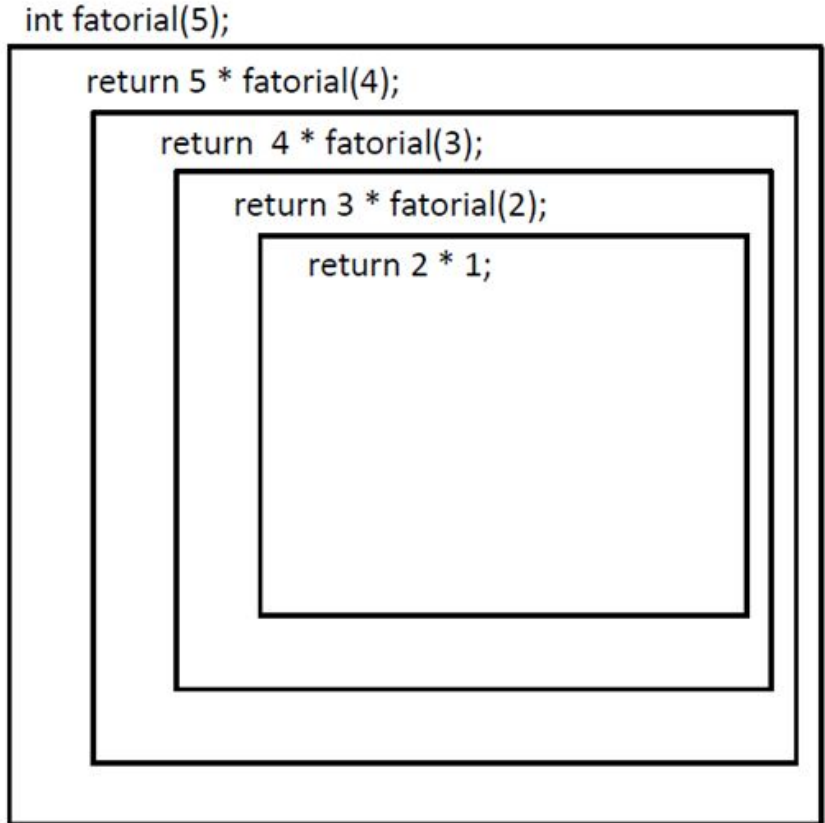
# Recursividade na Programação - Exemplo

```
1  #include <stdio.h>
2
3  int fatorial(int n){
4      if(n<=0)
5          return 1;
6      else
7          return (n * fatorial(n-1));
8  }
9
10 int main(void){
11     printf("%d", fatorial(5));
12 }
```



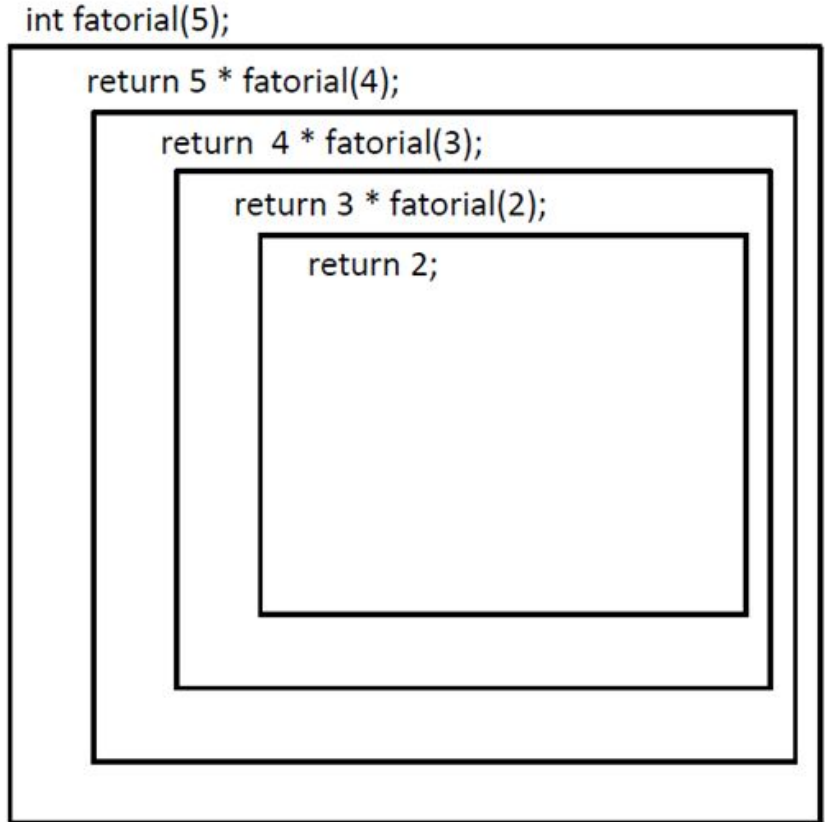
# Recursividade na Programação - Exemplo

```
1  #include <stdio.h>
2
3  int fatorial(int n){
4      if(n<=0)
5          return 1;
6      else
7          return (n * fatorial(n-1));
8  }
9
10 int main(void){
11     printf("%d", fatorial(5));
12 }
```



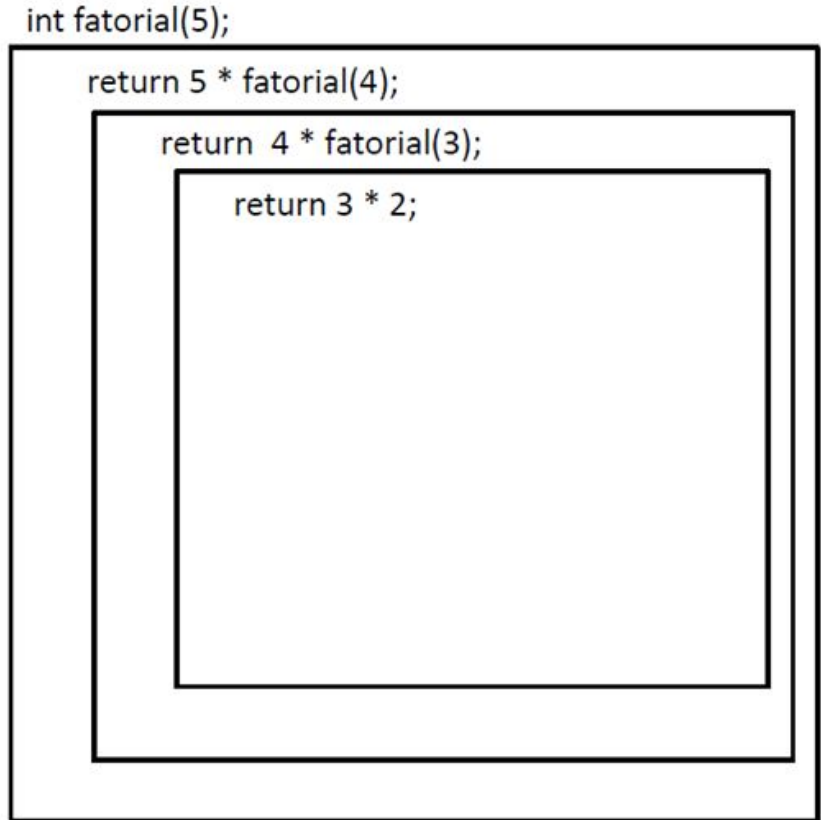
# Recursividade na Programação - Exemplo

```
1  #include <stdio.h>
2
3  int fatorial(int n){
4      if(n<=0)
5          return 1;
6      else
7          return (n * fatorial(n-1));
8  }
9
10 int main(void){
11     printf("%d", fatorial(5));
12 }
```



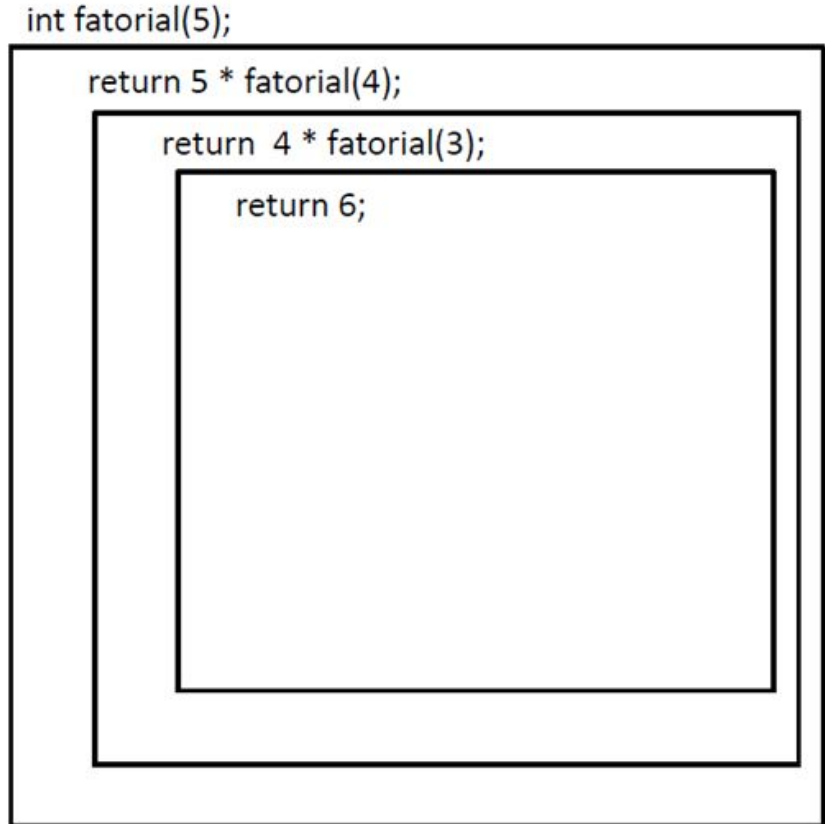
# Recursividade na Programação - Exemplo

```
1  #include <stdio.h>
2
3  int fatorial(int n){
4      if(n<=0)
5          return 1;
6      else
7          return (n * fatorial(n-1));
8  }
9
10 int main(void){
11     printf("%d", fatorial(5));
12 }
```



# Recursividade na Programação - Exemplo

```
1  #include <stdio.h>
2
3  int fatorial(int n){
4      if(n<=0)
5          return 1;
6      else
7          return (n * fatorial(n-1));
8  }
9
10 int main(void){
11     printf("%d", fatorial(5));
12 }
```



# Recursividade na Programação - Exemplo

```
1  #include <stdio.h>
2
3  int fatorial(int n){
4      if(n<=0)
5          return 1;
6      else
7          return (n * fatorial(n-1));
8  }
9
10 int main(void){
11     printf("%d", fatorial(5));
12 }
```

int fatorial(5);

return 5 \* fatorial(4);

return 4 \* 6;

# Recursividade na Programação - Exemplo

```
1  #include <stdio.h>
2
3  int fatorial(int n){
4      if(n<=0)
5          return 1;
6      else
7          return (n * fatorial(n-1));
8  }
9
10 int main(void){
11     printf("%d", fatorial(5));
12 }
```

int fatorial(5);

return 5 \* fatorial(4);

return 24;



# Recursividade na Programação - Exemplo

```
1  #include <stdio.h>
2
3  int fatorial(int n){
4      if(n<=0)
5          return 1;
6      else
7          return (n * fatorial(n-1));
8  }
9
10 int main(void){
11     printf("%d", fatorial(5));
12 }
```

int fatorial(5);

return 5 \* 24;

# Recursividade na Programação - Exemplo

```
1  #include <stdio.h>
2
3  int fatorial(int n){
4      if(n<=0)
5          return 1;
6      else
7          return (n * fatorial(n-1));
8  }
9
10 int main(void){
11     printf("%d", fatorial(5));
12 }
```

int fatorial(5);

return 120;

# Recursão vs. Iteração



- Tanto iteração quanto recursão usam repetição
- A **iteração** usa repetição em forma de comandos de repetição (**for**, **while**, **do-while**)
- Já a **recursão** usa a repetição na forma de **chamadas repetitivas a uma função**
- Ambas precisam de um **teste de terminação**
- A iteração termina quando a condição de teste falha e a recursão termina quando se atinge o caso base

# Recursão vs. Iteração



- Ambas podem entrar em loop infinito, no caso da iteração se o teste nunca se tornar falso e no caso da recursão se o problema não for reduzido de forma que convirja para o caso base

# Ordem da chamada

- A ordem da chamada de uma função recursiva pode mudar o resultado do algoritmo. Exemplos:

```
1  #include <stdio.h>
2
3  void conta(int num){
4      if(num < 5){
5          printf("%d\n", num);
6          conta(num+1);
7      }
8  }
9
10 int main(void){
11     conta(0);
12 }
```

VS.

```
1  #include <stdio.h>
2
3  void conta(int num){
4      if(num < 5){
5          conta(num+1);
6          printf("%d\n", num);
7      }
8  }
9
10 int main(void){
11     conta(0);
12 }
```

# Exercícios

# Exercícios

1. Implemente o problema do número Fatorial de forma iterativa (*utilize **while**, **for** ou **do-while***). Neste exercício você não usará recursão, mas poderá comparar as duas técnicas e entender melhor quando utilizar recursão.
2. Escreva uma função recursiva para cálculo de potência: **pot(base, expoente)** que, quando invocada, retorna **base<sup>expoente</sup>**.

Por exemplo, **pot(3, 4) = 3 \* 3 \* 3 \* 3**. Suponha que o expoente seja um número inteiro maior ou igual a 1. Dica: A etapa de recursão pode usar o relacionamento:

$$\text{base}^{\text{expoente}} = \text{base} * \text{base}^{\text{expoente} - 1}$$

a condição final ocorre quando o expoente é igual a 1