PUC Minas

Recursividade

João Pedro Oliveira Batisteli

Março, 2025


Recursão
Recursão
Recursão
Recursão
Recursão
Recursão
Recursão
Recursão
Recursão

Introdução

- Para definir novos conceitos costumamos usar elementos conhecidos.

Introdução

- Para definir novos conceitos costumamos usar elementos conhecidos.
- Ex: *"Defina o conjunto dos números naturais."*

Introdução

- Para definir novos conceitos costumamos usar elementos conhecidos.
- Ex: *"Defina o conjunto dos números naturais."*
  - $0 \in \mathbb{N}$
  - Se $n \in \mathbb{N}$
  - então $(n + 1) \in \mathbb{N}$

Definição Recursiva

- Um **objeto definido em termos dele próprio**.
- Geralmente, utilizada para definição de conjuntos infinitos.

Definição Recursiva

- Um **objeto definido em termos dele próprio**.
- Geralmente, utilizada para definição de conjuntos infinitos.
- Em programação:
  - um método recursivo caracteriza-se por chamar a si mesmo.
  - útil para desenvolver algoritmos mais concisos.

Definição Recursiva

- Um **objeto definido em termos dele próprio**.
- Geralmente, utilizada para definição de conjuntos infinitos.
- Em programação:
  - um método recursivo caracteriza-se por chamar a si mesmo.
  - útil para desenvolver algoritmos mais concisos.
- Uma função é dita recursiva quando ela chama a si própria.

Funções Recursivas

As funções recursivas são constituídas por:

1) Caso base:
    - ponto de paradas do método recursivo.
2) Regras de formação/chamadas recursivas:
    - regra para construção de novos elemento a partir dos elementos básicos.

Funções Recursivas

Três pontos importantes devem ser lembrados quando queremos escrever uma função recursiva.

1) Definir o problema em termos recursivos, ou seja, definir o problema usando ele próprio na definição.
    - Ex: O fatorial de um número **n** pode ser definido pela expressão: $n! = n \times (n-1)!$. (Regra de formação)

Funções Recursivas

Três pontos importantes devem ser lembrados quando queremos escrever uma função recursiva.

1) Definir o problema em termos recursivos, ou seja, definir o problema usando ele próprio na definição.
    - Ex: O fatorial de um número **n** pode ser definido pela expressão: $n! = n \times (n-1)!$. (Regra de formação)
2) Encontrar a condição básica / condição base / condição de parada, que quando é satisfeita, a recursão é interrompida.
    - Ex: Para o problema do fatorial, por definiçao $0! = 1$, logo é a condição de parade. (caso base)

Funções Recursivas

Três pontos importantes devem ser lembrados quando queremos escrever uma função recursiva.

1) Definir o problema em termos recursivos, ou seja, definir o problema usando ele próprio na definição.
    - Ex: O fatorial de um número **n** pode ser definido pela expressão: $n! = n \times (n-1)!$. (Regra de formação)
2) Encontrar a condição básica / condição base / condição de parada, que quando é satisfeita, a recursão é interrompida.
    - Ex: Para o problema do fatorial, por definiçao $0! = 1$, logo é a condição de parade. (caso base)
3) É necessário que cada vez que a função é chamada recursivamente, ela deve estar mais próxima de satisfazer a condição base, garantindo que ela parará em algum momento.
    - Ex: Para o problema do fatorial, a cada chamada, o valor de **n** estará mais próximo de zero.

Como funciona uma função recursiva?

- **Contexto ou escopo do módulo**: A cada chamada do módulo, cria-se uma nova área na memória para armazenar todo o contexto (variáveis, parâmetros, comandos, ponto de retorno) daquela chamada.
- Dessa forma, mesmo que os nomes sejam idênticos, eles são independentes dos correspondentes da chamada anterior por se encontrarem em áreas distintas.

Como funciona uma função recursiva?

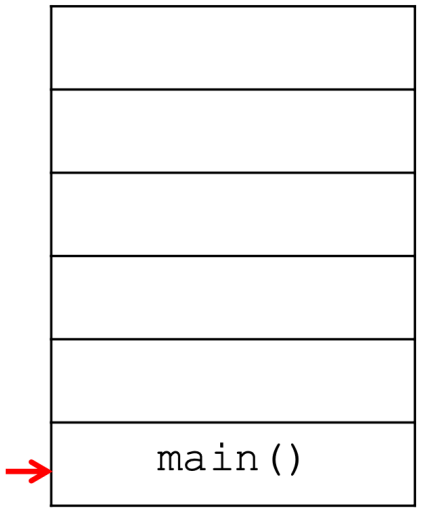
- **Sempre que um método é iniciado**: um novo conjunto de variáveis locais e de parâmetros é alocado.
- **Quando ocorre um retorno**:o conjunto de variáveis locais e parâmetros do método que estava sendo executado é desativado, dando lugar ao conjunto de valores do método que fez a chamada.
- Uma pilha é utilizada para armazenar o estado do método chamador.
- O estado de um método é composto por: todas as suas varáveis, parâmetros e endereço de retorno.

Como funciona uma função recursiva?

```
void main (){
      int x = 4;
      int fat = fatorial (x);
      printf("Fatorial de %d e %d",
x,fat);
}
```

Como funciona uma função recursiva?

```
void main (){
      int x = 4;
      int fat = fatorial(x);
      printf("Fatorial de %d e %d" x,
fat);
}
```

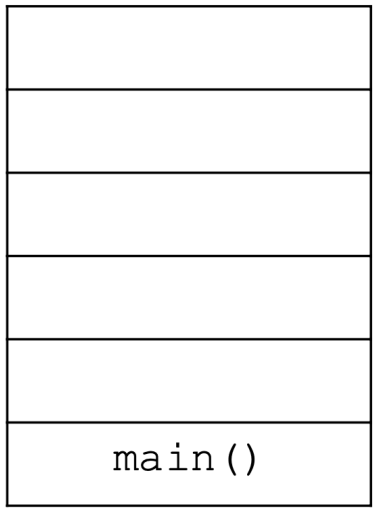


Sempre que um método chama outro, seu estado é armazenado na pilha de execução do programa.

Como funciona uma função recursiva?

```
int fatorial(int n){ ←
    if (n==0)          n = 4
      return 1;
    else
      return (n *
      fatorial(n-1));
}
```

Pilha de execução do programa



Como funciona uma função recursiva?

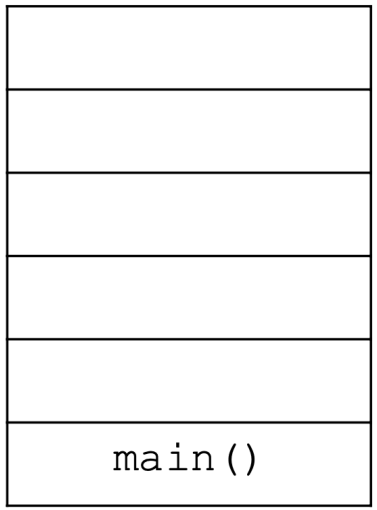
```
int fatorial(int n){
    if (n==0)
      return 1;
    else
      return (n *
      fatorial(n-1));
}
```

n = 4

←

Após a verificação da condição **if**, como **n** é diferente de zero, invoca-se o método novamente, mas agora decrementando o valor de **n**. (Observe que não ocorre o retorno ainda).

Pilha de execução do programa



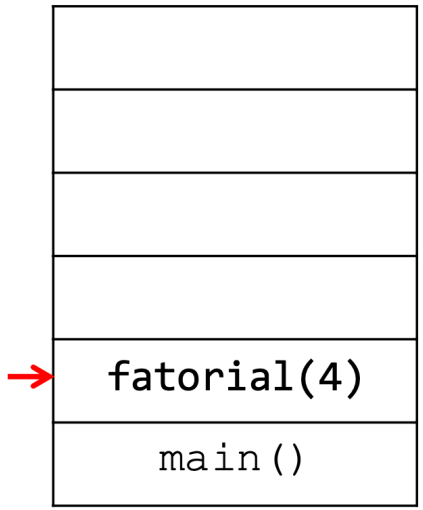
Como funciona uma função recursiva?

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

n = 4

Sempre que um método chama outro, seu estado é armazenado na pilha de execução do programa.

Pilha de execução do programa

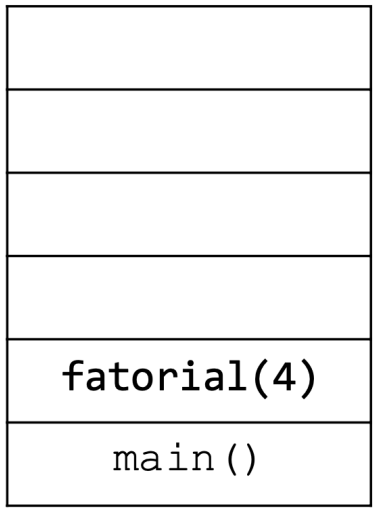


Como funciona uma função recursiva?

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

n = 3

Pilha de execução do programa



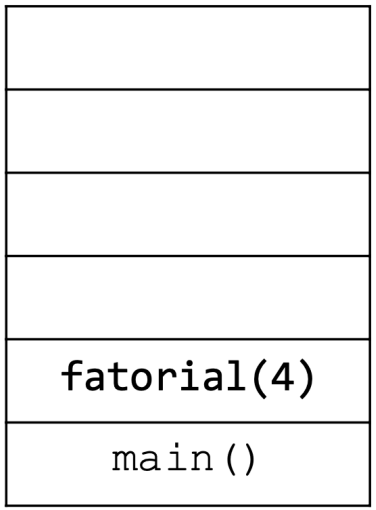
Como funciona uma função recursiva?

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

n = 3

Como **n** ainda é diferente de zero, invoca-se o método novamente, mas agora decrementando o valor de **n.** (Observe que não ocorre o retorno ainda).

Pilha de execução do programa



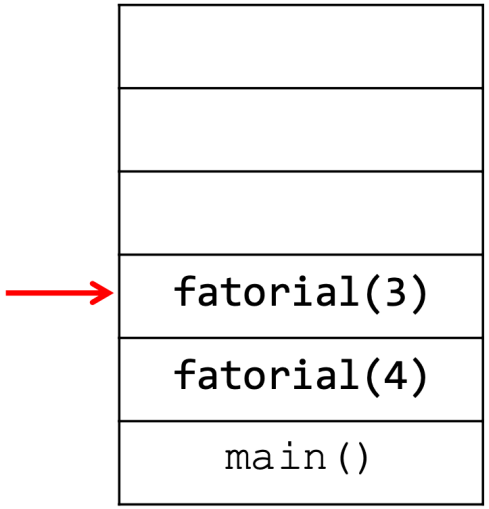
Como funciona uma função recursiva?

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

n = 3

Sempre que um método chama outro, seu estado é armazenado na pilha de execução do programa.

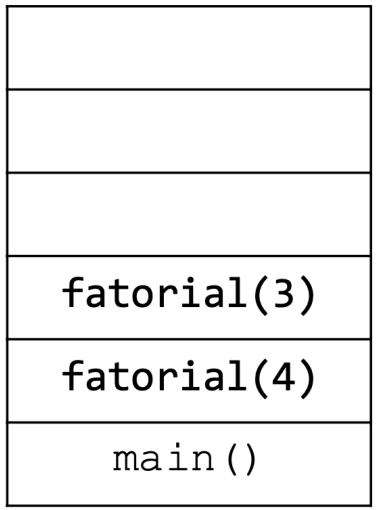
Pilha de execução do programa



Como funciona uma função recursiva?

```
int fatorial(int n){ ←
    if (n==0)                n = 2
      return 1;
    else
      return (n *
      fatorial(n-1));
}
```

Pilha de execução do programa

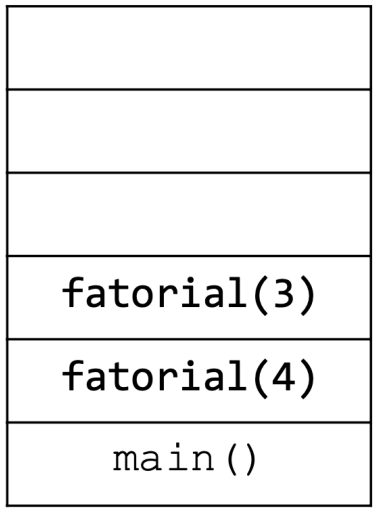


Como funciona uma função recursiva?

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

n = 2

Pilha de execução do programa



Como **n** ainda é diferente de zero, invoca-se o método novamente, mas agora decrementando o valor de **n**. (Observe que não ocorre o retorno ainda).

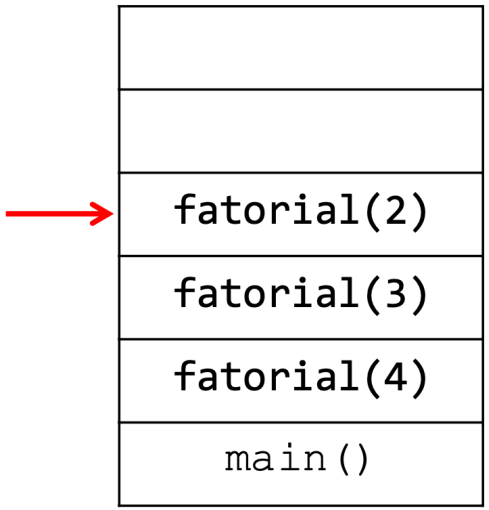
Como funciona uma função recursiva?

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

n = 2

Sempre que um método chama outro, seu estado é armazenado na pilha de execução do programa.

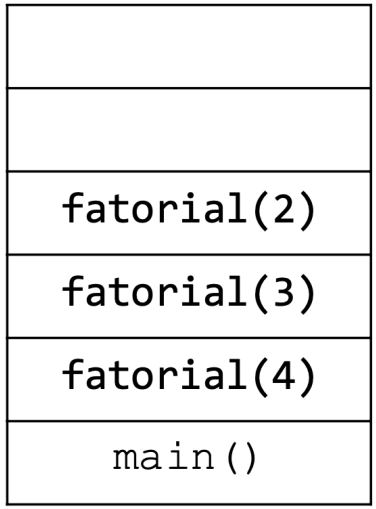
Pilha de execução do programa



Como funciona uma função recursiva?

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

Pilha de execução do programa



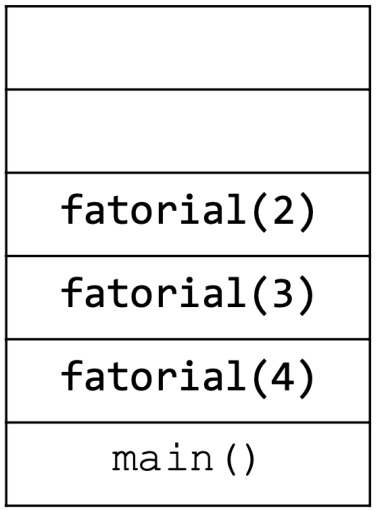
Como funciona uma função recursiva?

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

n = 1

Como **n** ainda é diferente de zero, invoca-se o método novamente, mas agora decrementando o valor de **n.** (Observe que não ocorre o retorno ainda).

Pilha de execução do programa



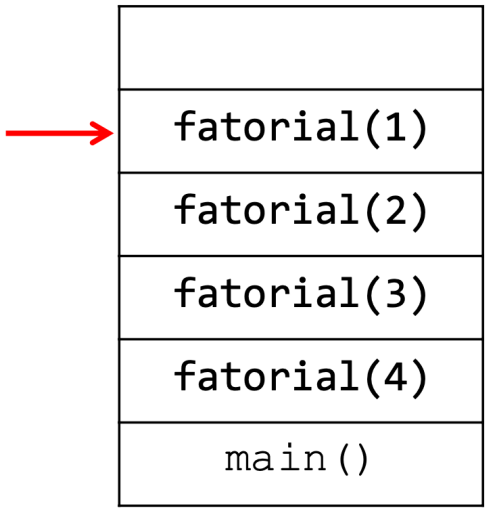
Como funciona uma função recursiva?

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

n = 1

Sempre que um método chama outro, seu estado é armazenado na pilha de execução do programa.

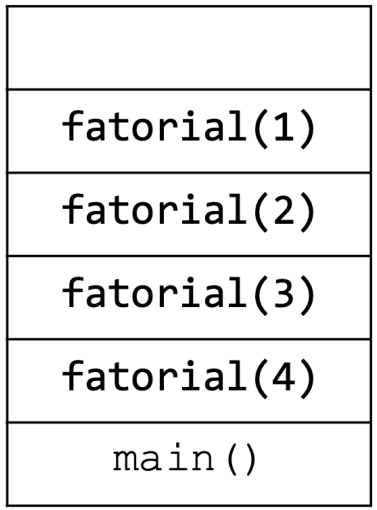
Pilha de execução do programa



Como funciona uma função recursiva?

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

Pilha de execução do programa

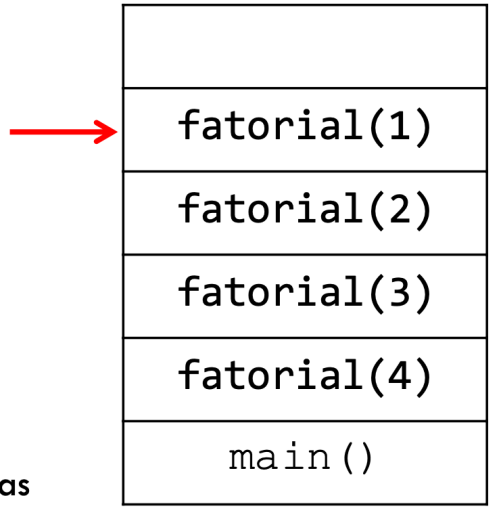


Como funciona uma função recursiva?

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

n = 0

Pilha de execução do programa



- Como **n** é igual a zero, apenas retorna 1 para a chamada imediatamente anterior do próprio método. As chamadas recursivas acontecem até que o caso base seja alcançado (condição de parada).

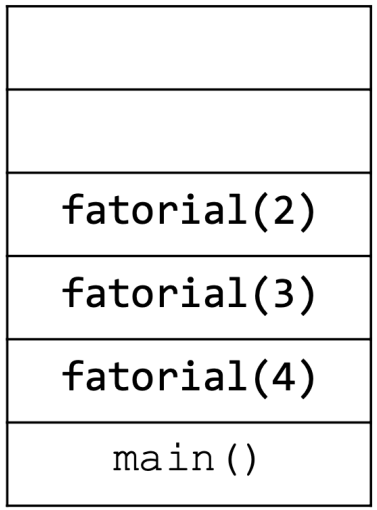
Como funciona uma função recursiva?

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

n = 1

} A chamada atual retornará o valor da chamada anterior (igual a 1) vezes **n** (igual a 1, nesta chamada).

Pilha de execução do programa



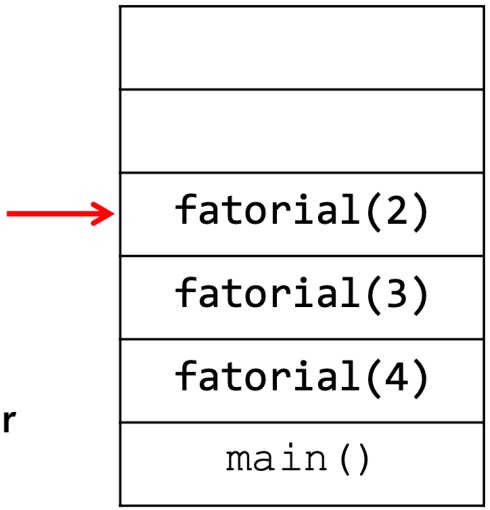
Como funciona uma função recursiva?

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

n = 1

Retorna 1 para a chamada imediatamente anterior do próprio método.

Pilha de execução do programa



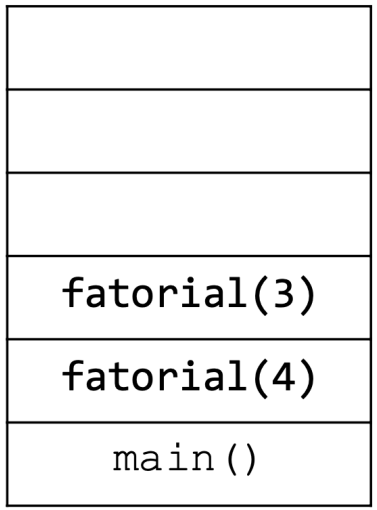
Como funciona uma função recursiva?

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

n = 2

A chamada atual retornará o valor da chamada anterior (igual a 1) vezes **n** (igual a 2, nesta chamada).

Pilha de execução do programa



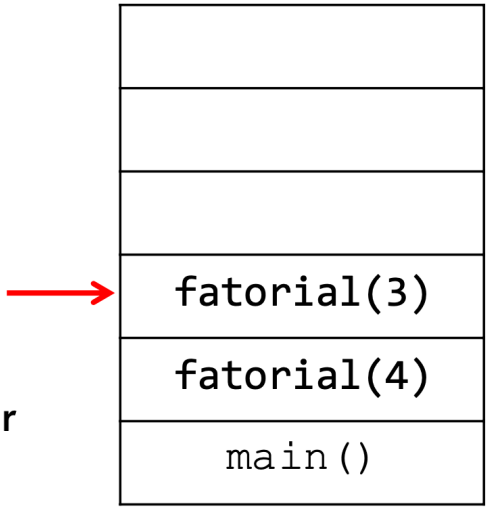
Como funciona uma função recursiva?

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

n = 2

Retorna 2 para a chamada imediatamente anterior do próprio método.

Pilha de execução do programa



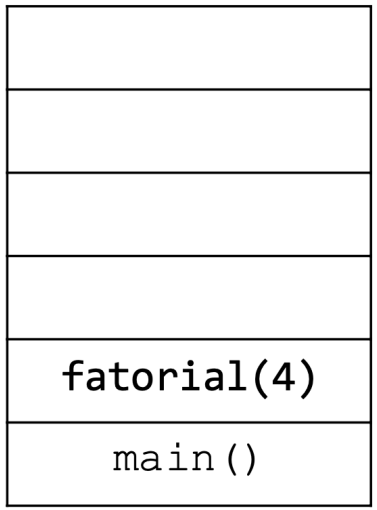
Como funciona uma função recursiva?

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

n = 3

A chamada atual retornará o valor da chamada anterior (igual a 2) vezes **n** (igual a 3, nesta chamada).

Pilha de execução do programa



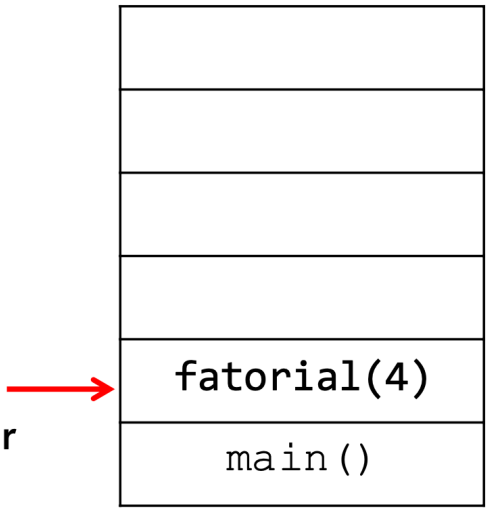
Como funciona uma função recursiva?

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

n = 3

Retorna 6 para a chamada imediatamente anterior do próprio método.

Pilha de execução do programa



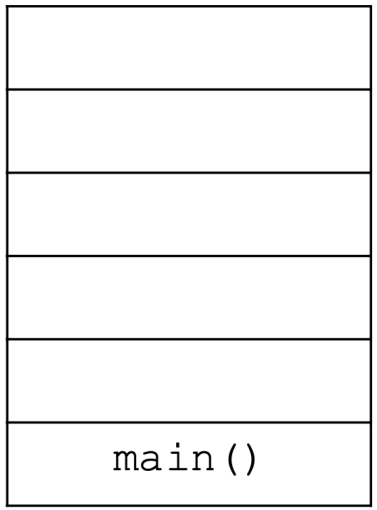
Como funciona uma função recursiva?

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

n = 4

A chamada atual retornará o valor da chamada anterior (igual a 6) vezes **n** (igual a 4, nesta chamada).

Pilha de execução do programa



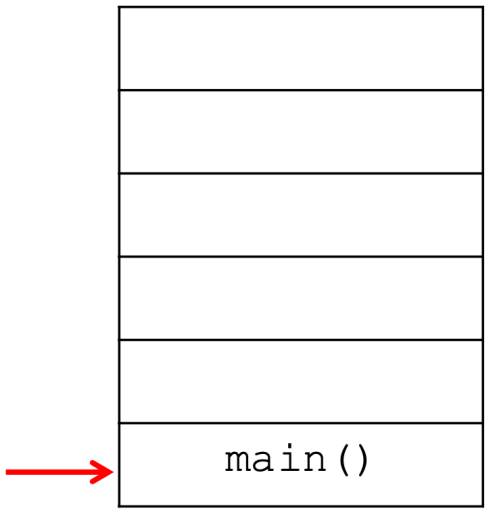
Como funciona uma função recursiva?

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

n = 4

Retorna 24 para o método chamador.

Pilha de execução do programa

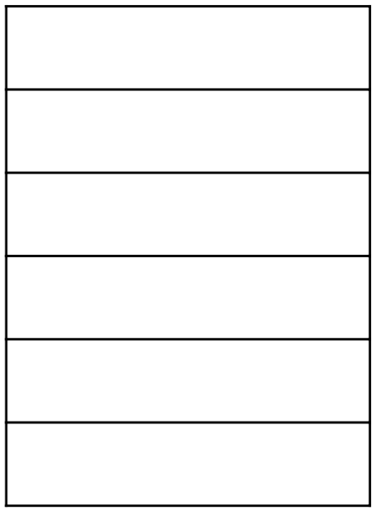


Como funciona uma função recursiva?

```
void main (String[] args){
      int x = 4;
      int fat = fatorial(x);
      printf("Fatorial de %d e %d" x,
fat);
}
```

Valor de retorno da primeira chamada do método fatorial() pelo método main(). A variável **fat** receberá o valor 24 (resultado de 4!). O valor 24 será exibido na tela e o programa será finalizado.

Pilha de execução do programa



Exercício

Escreva um programa que leia dois valores inteiros, calcule por meio de uma função recursiva, a divisão **inteira** entre eles e imprima o resultado. A divisão deve ser realizada por meio de subtrações sucessivas recursivas. A divisão inteira despreza a parte decimal do resultado, por exemplo: $5 \,/\, 2 = 2$ e $15 \,/\, 4 = 3$.

Tipos de Recursividade

- Recursividade direta.
- Recursividade indireta.
- Recursividade de cauda.
- Recursividade sem cauda.
- Recursividade em árvore.

Recursividade direta

- O tipo mais comum/simples de recursão.
- Recursão direta é o tipo de recursão em que uma função chama a si mesma diretamente dentro do seu próprio bloco de código (escopo da função).
- Isso significa que a função aparece como parte da sua própria definição, e a função chama a si mesma para executar sua tarefa.

Recursividade de cauda

- Uma recursão de cauda ocorre quando a chamada recursiva é a última operação feita pela função antes de retornar. Ou seja, não há mais nenhuma operação pendente após a chamada recursiva.
- Não é necessário guardar a posição onde foi feita a chamada, visto que a chamada é a última operação realizada pela função.

Recursividade sem cauda

- Uma função recursiva é considerada não recursiva sem cauda se a chamada recursiva não for a última coisa feita na função.
- Há código a ser executado após a chamada recursiva.
- Pode gerar certa dificuldade para compreensão da lógica de funcionamento do método recursivo.

Recursividade SEM cauda - ex.

```
void inverter(){
  char letra;
  scanf("%c", &letra);
  if(letra != ‘\ n’){
    inverter();
    printf("%c", letra);
  }
}
int main(){
  inverter();
  return 0;
}
```

Exercício

Faça uma função recursivo que receba dois números inteiros e retorne a multiplicação do primeiro pelo segundo fazendo somas.

$$\text{Ex: } 3 \times 5 = (3 + 3 + 3 + 3 + 3)$$

Desafio 1

Implemente uma função **recursiva** chamada `numeroPerfeito` que receba um número inteiro positivo e retorne *1* se for um número perfeito, ou *0* caso contrário.

Definição:

- Divisores próprios de um número são todos os divisores naturais menores que o número em questão.
- Um número perfeito é um número natural que é igual à soma de seus divisores próprios.

Exemplo de funcionamento:

- Ao receber o número 6 como entrada, a função deve retornar True, pois os divisores próprios de 6 são 1, 2 e 3, e a soma deles é 6.

Desafio 2

Implemente uma função **recursiva** chamada `numeroHarshad` que receba um número inteiro positivo e retorne *1* se for um número Harshad, ou *0* caso contrário.

Definição:

- Um número Harshad (também chamado de número Niven) é um número inteiro divisível pela soma de seus dígitos.

Exemplo de funcionamento:

- Ao receber o número 18 como entrada, a função deve retornar *True*, pois a soma dos dígitos de 18 é 1+8=9, e 18÷9=2, que é exato.
- Ao receber o número 19 como entrada, a função deve retornar *False*, pois a soma dos dígitos de 19 é 1+9=10, e 19÷10 não resulta em um número inteiro.

Dúvidas?