Linguagem de Programação

Recursividade

ECT2303

helton.maia@ect.ufrn.br

Funções recursivas

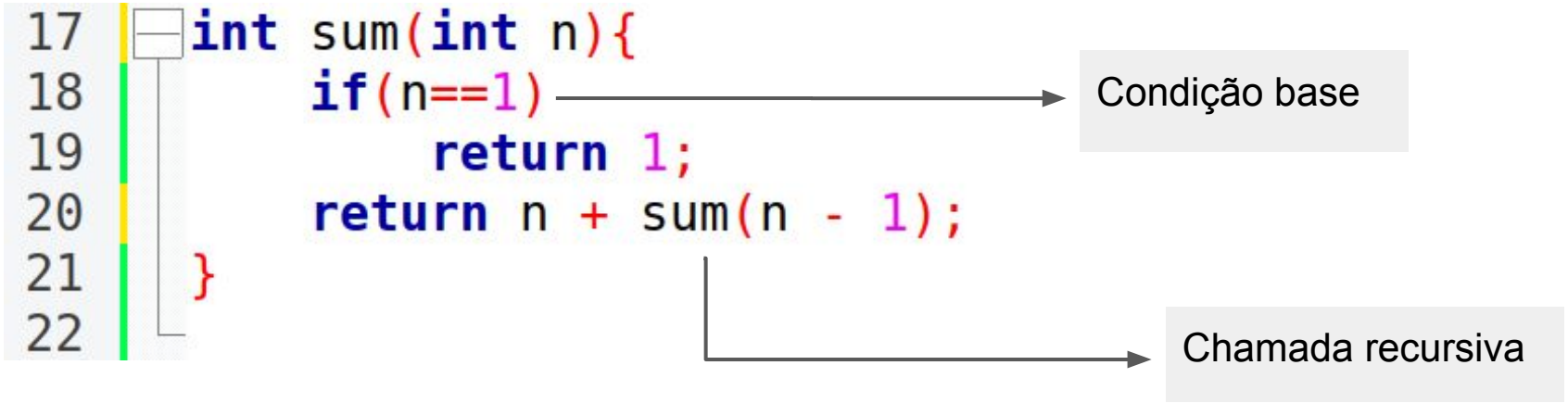
Qual a importância na resolução de problemas?

Definição:

- Recursividade ocorre quando uma função chama a si mesma, cria-se um *loop* e, portanto, nunca terminará até que se chegue a uma condição de parada (base);
- Na ciência da computação a recursão é um método para resolver um problema em que a solução depende de soluções para instâncias menores do mesmo problema.

Exemplo de recursividade

Seu programa deve receber do usuário um número inteiro positivo. Depois, deve-se calcular de forma recursiva a soma de todos os números de 1 até o número dado.



Continuando exemplo, considerando n = 5

```
17  int sum(int n){
18      if(n==1)
19          return 1;
20      return n + sum(n - 1);
21  }
22
```

Fig. 1

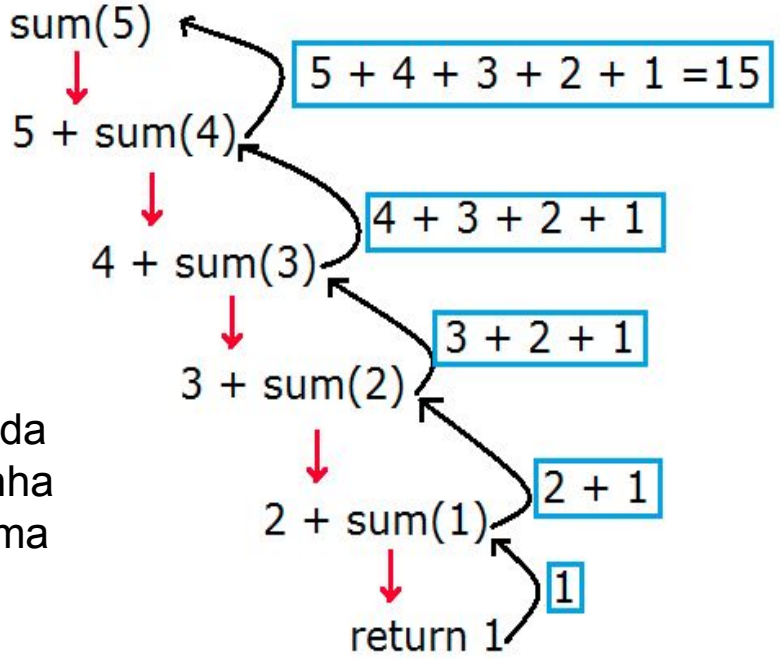


Fig. 2

No exemplo da Fig. 1, (n == 1) na linha 18, é chamada de condição de base. Note que em soma (n-1) na linha 20, a função está chamando a si mesma, ou seja, uma recursão.

Portanto, quando a função sum é chamada inicialmente com um valor de argumento igual a 5, ou seja, sum(5), o processo recursivo similar ao mostrado na fig. 2 é executado.

Como calcular o fatorial de um número recursivamente?

Condições de parada e recursão estabelecidas:

$$n! = \begin{cases} 1, & se\ n = 0 \\ 1, & se\ n = 1 \\ n * (n-1)!, & se\ n > 1 \end{cases}$$

Escreva a função em c++ para testar!

Exemplo: calculando fatorial

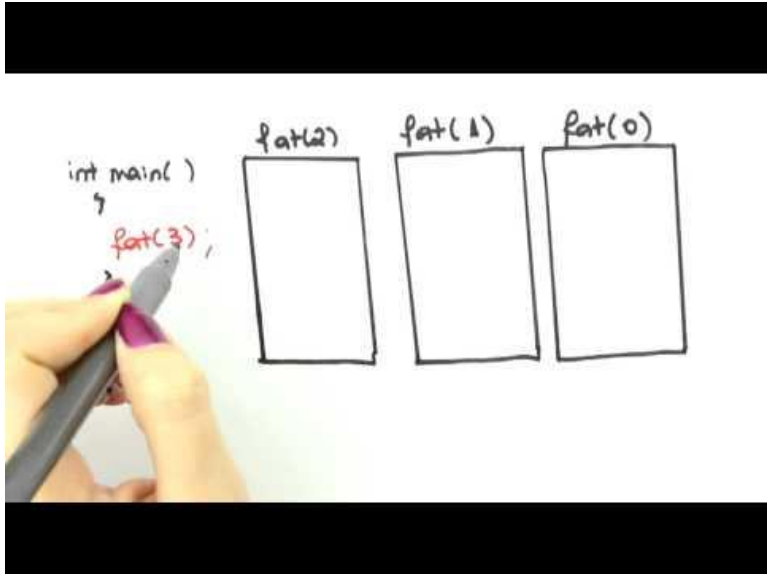
```
1  #include<iostream>
2  using namespace std;
3
4  int fat(int n);
5
6  int main(){
7      int n;
8
9      cout << "Entre com um inteiro positivo: ";
10     cin >> n;
11     cout << "Fatorial: " << n << " = " << fat(n);
12
13     return 0;
14 }
15
16 int fat(int n){
17     if(n == 0 || n == 1)
18         return 1;
19     return n * fat(n - 1);
20 }
21
```

$$n! = \begin{cases} 1, & se\ n = 0 \\ 1, & se\ n = 1 \\ n * (n-1)!, & se\ n > 1 \end{cases}$$

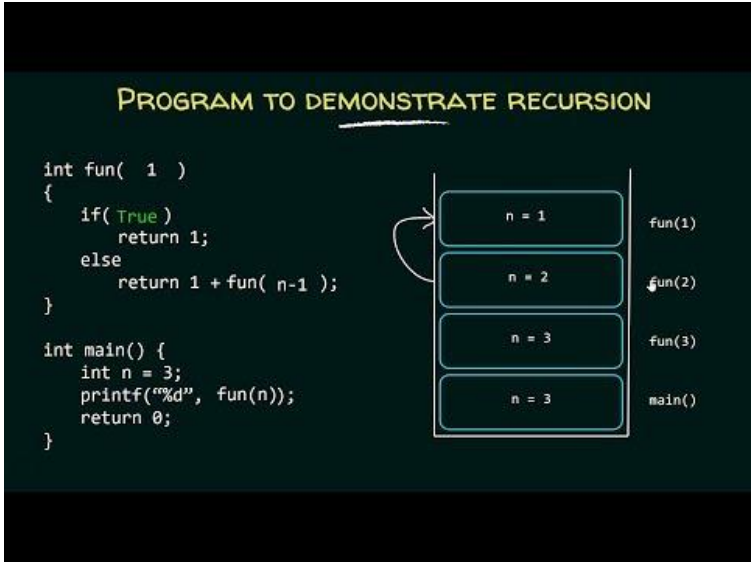
Condição base

Chamada recursiva

Vídeos auxiliares sobre recursividade:



https://youtu.be/kS_VJYWeqIQ



https://youtu.be/kepBmgvWNDw

Exercício 1

Escreva uma função recursiva que recebe dois números inteiros “a” e “b”, e computa a soma dos números inseridos no intervalo [a , b].

Entrada: 5 15
Saída: 110

Exercício 2

Crie um função recursiva que recebe como entrada um número inteiro e retorna seu número de dígitos.

Entrada: 223452
Saída: 6

Exercício 3

Escreva uma função recursiva que calcule o número equivalente em binário de um número decimal inserido pelo usuário. Lembre-se que os números decimais possuem base 10 e os binários são de base 2.

Entrada: 7 Saída: 111
Entrada: 100 Saída: 1100100

Exercício 4

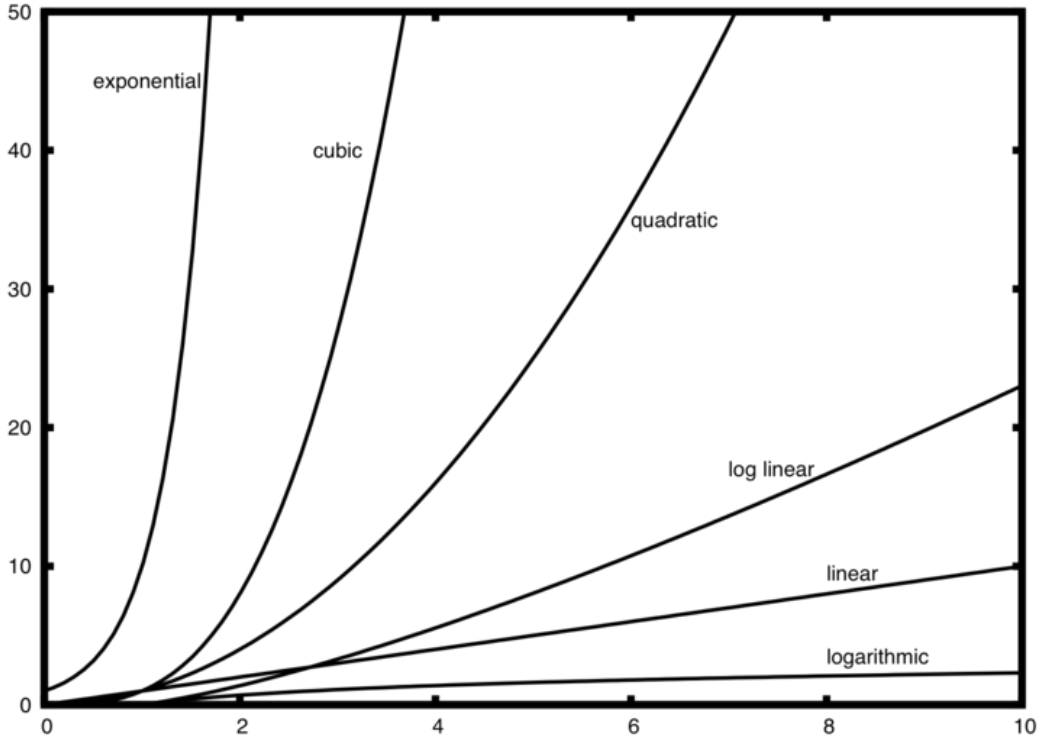
Escreva uma função recursiva que determina se um número natural é primo ou não. A função deve realizar os testes e retornar verdadeiro ou falso.

Entrada: 12 Saída: false
Entrada: 13 Saída: true

Complexidade computacional (tempo de execução)

Big-O Notation:

f(n)	Name
1	Constant
$\log n$	Logarithmic
n	Linear
$n \log n$	Log Linear
n^2	Quadratic
n^3	Cubic
2^n	Exponential



referência: http://interactivepython.org/runestone/static/pythonds/AlgorithmAnalysis/BigONotation.html

Perguntas ?