IMD0030 – LINGUAGEM DE PROGRAMAÇÃO I

Aula 05 – Recursividade

INSTITUTO METRÓPOLE DIGITAL

UFRN UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE

Recursão

- Método de programação no qual uma função chama a si própria
- Dividir e conquistar
  - problemas menores
  - combinar soluções
  - soluções mais simples e fáceis de se analisar

```cpp
#include <iostream>
using namespace std;

int removeBolas(int nBolas)
{
    if (nBolas == 0)
    {
        return nBolas;
    }
    else
    {
        cout << "Há " << nBolas << " bolas no balde." << endl;
        return removeBolas(nBolas-1);
    }

}

int main(void)
{
    int N, bolasBalde;

    cout << "Quantas bolas há no balde?" << endl;
    cin >> bolasBalde;

    N = removeBolas(bolasBalde);

    cout << "Pronto! O total de bolas é: " << N <<  endl;

    return 0;
}
```

Cálculo do fatorial (abordagem recursiva)

- 4! = 4*3*2*1
- 3! = 3*2*1
- 2! = 2*1
- 1! = 1
- 0! = 1

- 4! = 4*3!
- 3! = 3*2!
- 2! = 2*1!
- 1! = 1*0!
- 0! = 1

N! = N*(N-1)!

0! = 1

Cálculo do fatorial (abordagem recursiva)

N! = N*(N-1)!

0! = 1 (caso base)

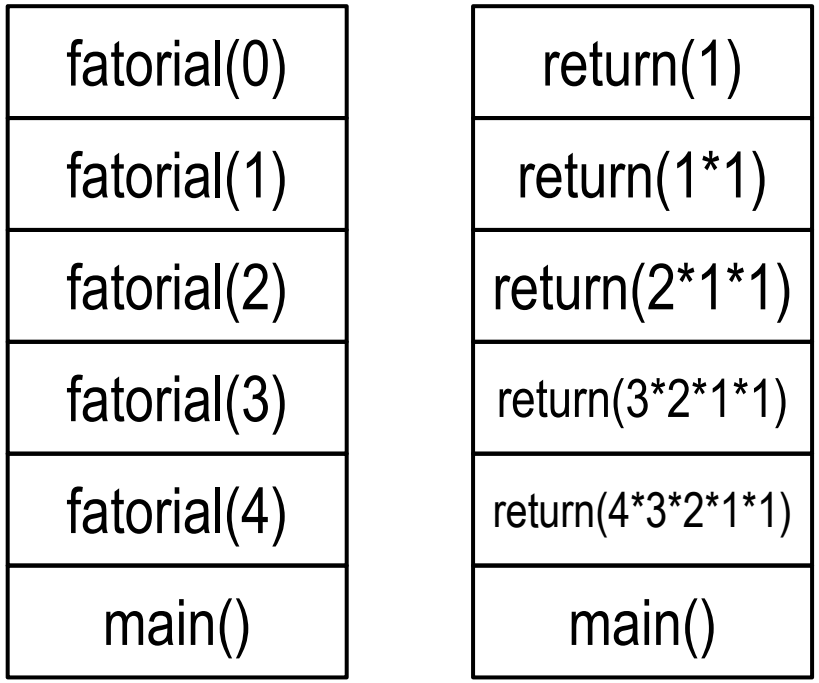
- quando a recursão deve parar
- mudar o valor do parâmetro da chamada recursiva

```
3  #include <iostream>
4  using namespace std;
5
6  int fatorial(int N)
7  {
8      if (N == 0)    critério de parada
9      {
10         return 1;
11     }
12     else
13         return N*fatorial(N-1);parâmetro da chamada recursiva
14
15 }
16
17 int main(void)
18 {
19     int n, F;
20
21     cout << "Digite um número:" << endl;
22     cin >> n;
23
24     F = fatorial(n);
25
26     cout << "O fatorial de " << n << " é: " << F << endl;
27
28     return 0;
29 }
```

Cálculo do fatorial (abordagem recursiva)

N! = N*(N-1)!

0! = 1 (caso base)



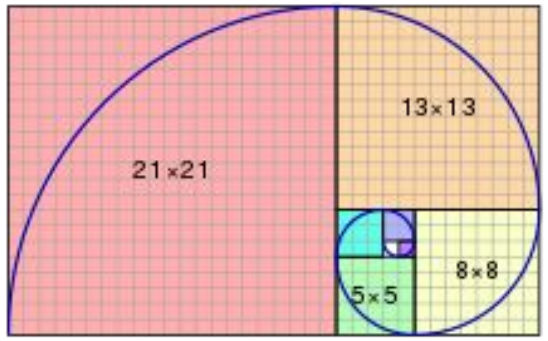
Soma de N números inteiros

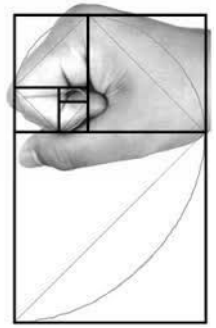
Utilizando recursão, faça um programa que retorne a soma de 1 a N, onde N é um número inteiro positivo fornecido pelo usuário

Fibonacci

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, …

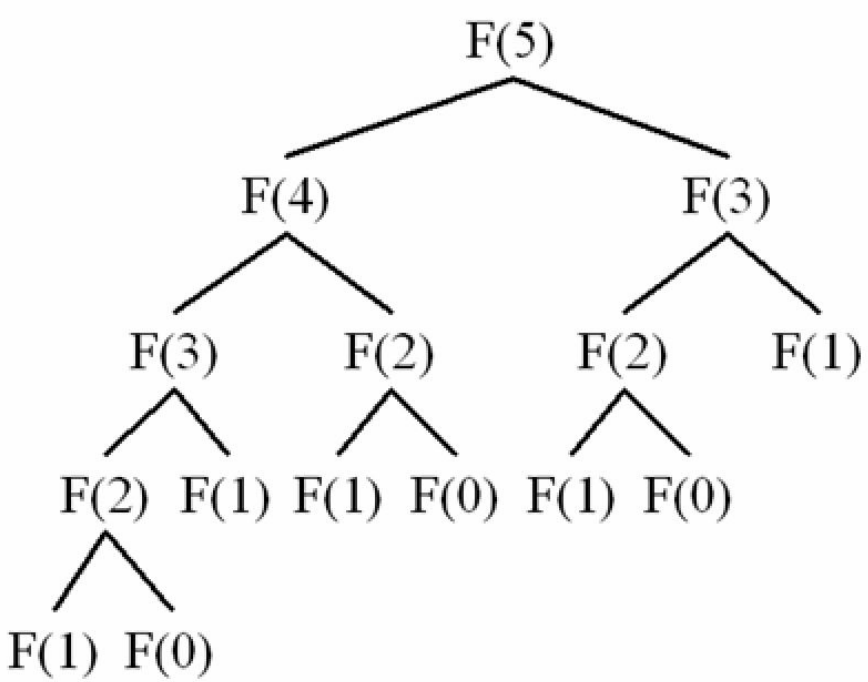
- $F_n = F_{n-1} + F_{n-2}$
- $F_0 = 0, F_1 = 1$





Fibonacci

- F_{31}: 4.356.617 chamadas
- F_{32} : 7.049.155 chamadas



Máximo Divisor Comum

Algoritmo de Euclides

$$mdc(p,q) = \begin{cases} p, & se \;\; q = 0 \\ mdc(q,\; p\%q) & caso\ contrário \end{cases}$$

Recursão vs Iteração

- quando utilizar recursão?
- quais os pontos críticos na implementação de um programa recursivo?

Recursão	Iteração
Estruturas de seleção *if*, *if-else* ou *switch*.	Estruturas de repetição *for*, *while* ou *do-while*.
Repetição por chamadas de funções repetidas.	Repetição explícita.
Termina quando um caso base é reconhecido.	Termina quando teste do laço falha.
Pode ocorrer infinitamente.	Pode ocorrer infinitamente.
Lento.	Rápido.
Soluções simples e de fácil manutenção.	Soluções complicadas e de difícil manutenção.

www.ic.unicamp.br/~oliveira/doc/mc102_2s2004/Aula19.pdf

Tipos de recursão

- recursão direta
  - função chama a mesma função
- recursão indireta
  - uma função “a” é dita indireta se chama uma função “b” que por sua vez chama a função “a”
- recursão em cauda
  - a chamada recursiva é a última ação da função. Se houver a necessidade de avaliar alguma expressão após o retorno da chamada recursiva, a função não caracteriza uma recursão em cauda

Recursão em cauda

```
void imprime(int n) {
    if (n == 0)
        return;
    else
        cout << n;
    return imprime(n-1);
}
```

```
int main(void) {
    imprime(4);
    return 0;
}
```

Recursão em cauda

```
void imprime(int n) {
    if (n == 0)
        return;
    imprime(n-1);
    cout << n;
}
```

```
int main(void) {
    imprime(4);
    return 0;
}
```

Recursão em cauda

```
int fact(int n){
    if (n == 0)
     return 1;

    return n*fact(n-1);
}

int main(){
    cout << fact(5);
    return 0;
}
```

Recursão em cauda

```
int fact(int n, int a){
    if (n == 0)
        return a;

    return fact(n-1, n*a);
}

int main(){
    cout << fact(5,1);
    return 0;
}
```

?

Aula 05

Recursividade

Aula 06

Laboratório 1