1 Recursividade

Muitas vezes nos deparamos com alguns problemas de computação envolvendo loops que chegam a ficar muito complexos.

Na maioria das vezes podemos simplificar estes algoritmos usando a recursividade.

Algoritmos recursivos são muito usados em pesquisas de diretórios, inteligência artificial em muitas outras áreas.

A recursividade ocorre quando um algoritmo ou método chama a si mesmo.

Por exemplo: Imaginemos um algoritmo simples para calcular o fatorial de um número.

```
1 int fatorial(int x){
2   int a, fatx=1;
3   for(a=x; a>1; a--){
4     fatx=fatx*a;
5   }
6   return fatx;
7 }
```

Agora vamos pensar como se calcula o fatorial de um número:

- O fatorial de zero é, por definição, 1.
- O fatorial de 1 é 1.
- O fatorial de 5! é 5x4x3x2x1.

Agora qual é o fatorial de N?
Olhando a definição, podemos dizer que o fatorial de N é:

$$\begin{cases} 1 & se \quad N < 2; \\ N & * \quad (N-1)! \end{cases} \tag{1}$$

ou seja,

- o fatorial de 5 é igual a $5x4!$
- o fatorial de 4 é igual a $4x3!$
- o fatorial de 3 é igual a $3x2!$
- o fatorial de 2 é igual a $2x1!$

Viram a recursividade?

1.1 Versão recursiva do fatorial

Agora que sabemos o que é recursividade, vamos a uma versão recursiva da função fatorial:

```
1 int fatRecursivo(int x){
2     if((x == 0) || (x == 1))
3         return 1;
4     return fatRecursivo(x-1) * x;
5 }
```

Explicando:
O *if* testa se o argumento passado para a função é menor que dois e em caso afirmativo a função retorna 1 e acaba.
Se o o argumento não for menor que dois, a função chama a ela própria com o argumento (x) decrementado de uma unidade e multiplica o valor de retorno por x.

1.2 Outro exemplo

Como você calcularia a soma dos números naturais de 0 a X, num programa em C++?

Um modo seria esse: se X for 0, a soma da zero, e eu já tenho a resposta! Caso contrário, eu somo X com a soma dos naturais calculada de 1 ate X-1. E exatamente assim que implementamos um algoritmo recursivo. Veja esse código:

```
1  int soma_recursiva(int n){
2     if (n==0){
3       return 0;
4     }
5     else {
6       return n + soma_recursiva(n-1);
7     }
8  }
9
10 int soma_nao_recursiva(int n){
11
12    int soma=0, i;
13
14    for(i=1;i<=n;i++){
15    soma+=i;
16    }
17    return soma;
18 }
```

Há um pequeno preço a pagar pela recursividade: espaço em memória. Explicaremos o por quê.

Primeiramente faremos o teste de mesa: imagine-se trabalhando como o computador e analise cada passo do algoritmo.

Na versão não-recursiva, ele fará o cálculo de T(10) (o valor da soma de zero a dez) assim:

soma começa de 0; para i de 0 ate n, some i ao valor da soma e jogue em soma(sobrescrever).

Assim, ocorre algo assim nas posições de memória:

soma	0	1	3	6	10	15	21	28	36	45	55
i	0	1	2	3	4	5	6	7	8	9	10

Rápido, fácil e simples.

Agora a versão recursiva. Nela, os valores desconhecidos são colocados numa pilha na memória. Veja:

$$
\begin{aligned}
T(10) &= T(9)+10 \\
T(9) &= T(8)+9 \\
T(8) &= T(7)+8 \\
T(7) &= T(6)+7 \\
T(6) &= T(5)+6 \\
T(5) &= T(4)+5 \\
T(4) &= T(3)+4 \\
T(3) &= T(2)+3 \\
T(2) &= T(1)+2 \\
T(1) &= T(0)+1=1
\end{aligned}
$$

Até agora cada valor foi colocado numa pilha, com a ordenação inversa do modo que eu ilustrei. Assim, agora a máquina irá retornar os valores, um a um:

$$\begin{aligned} T(2) &= 3 \\ T(3) &= 6 \\ T(4) &= 10 \\ T(5) &= 15 \\ T(6) &= 21 \\ T(7) &= 28 \\ T(8) &= 36 \\ T(9) &= 45 \\ T(10) &= 55 \end{aligned}$$

Perceberam como acontece o gerenciamento de uma função recursiva?
Então, segue a dica: não abuse da recursividade! Use-a apenas em situações especiais, tais como:

1. Não houver problemas com espaço de memória ou dos registradores;
2. Não houver problemas com tempo de execução do programa.
3. A solução recursiva ser mais limpa e legível que a não-recursiva.
4. Tiver que ensinar recursividade!!!

1.3 Exemplo

Em matemática, a Sucessão de Fibonacci (também Sequência de Fibonacci), é uma sequência de números inteiros, começando normalmente por 0 e 1, na qual, cada termo subsequente (numero de Fibonacci) corresponde a soma dos dois anteriores. A sequência recebeu o nome do matemático italiano Leonardo de Pisa, mais conhecido por Fibonacci (contração do italiano filius Bonacci), que descreveu, no ano de 1202, o crescimento de uma população de coelhos, a partir desta. Tal sequência já era no entanto, conhecida na antiguidade.

Os números de Fibonacci são, portanto, os números que compõem a seguinte sequência:
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, ?

A sequência de Fibonacci tem aplicações na análise de mercados financeiros, na ciência da computação e na teoria dos jogos. Também aparece em configurações biológicas, como, por exemplo, na disposição dos galhos das árvores ou das folhas em uma haste, no arranjo do cone da alcachofra, do abacaxi, ou no desenrolar da samambaia.

Definição matemática

$$\begin{aligned} F(1) &= F(2) = 1 \\ F(n+2) &= F(n+1) + F(n) \quad n>0 \end{aligned}$$

Elabore um programa em C++ que implemente a sequencia de Fibonacci de forma recursiva e de forma não-recursiva. apresentando na tela o número Fibonacci de uma determinada posição.

Solução

Forma não-recursiva ou iterativa

```
int fibonacci_iterativo(int n){
    if(n == 1)
      return 0;
    if(n == 2)
      return 1;
    int f1=0, f2=1, fib;
    for(i = 3; i <= n; i++){
        fib=f1+f2;
        f1=f2;
        f2=fib;
    }
    return fib;
}
```

Entendendo a função:

Quando estamos fazendo as contas na mão, seguimos este procedimento para calcular F(10):

$$\begin{cases} F(1) = 0 \\ F(2) = 1 \\ F(3) = F(2) + F(1) = 0 + 1 = 1 \\ F(4) = F(3) + F(2) = 1 + 1 = 2 \\ F(5) = F(4) + F(3) = 2 + 1 = 3 \\ F(6) = F(5) + F(4) = 3 + 2 = 5 \\ F(7) = F(6) + F(5) = 5 + 3 = 8 \\ F(8) = F(7) + F(6) = 8 + 5 = 13 \\ F(9) = F(8) + F(7) = 13 + 8 = 21 \\ F(10) = F(9) + F(8) = 21 + 13 = 34 \end{cases} \tag{2}$$

Que idéia estamos usando? Simples: a cada momento, pegamos os dois valores anteriores e somamos e gravamos na terceira, sem esquecer de gravar as duas posições anteriores!

Forma recursiva

```
int fibonacci_recursivo(int n){
    if(n == 1){
       return 0;
    }
    else if (n == 2){
       return 1;
    }
    else{
       return fibonacci_recursivo(n-1)+fibonacci_recursivo(n-2);
     }
}
```