Recursividade

É um recurso muito utilizado na maioria das linguagens atuais de programação. Ela permite que os programas gerados sejam bem menores e mais fáceis de serem lidos e entendidos. Ela permite que você implemente seus programas de forma similar ao raciocínio adquirido no dia a dia. Assim, você não precisa reformular seu modo de raciocinar para desenvolver programas. Vamos ver um exemplo clássico da utilização da recursividade no cálculo do fatorial de um número. Desta forma você poderá comprovar o que acabamos de afirmar.

FATORIAL DE UM NÚMERO

-Como definir o fatorial de um número?

Vamos esquematizar as regras que definem esta função.
Sabemos, da matemática, que:

1- O **FATORIAL** de **zero** é **um**
2- O **FATORIAL** de um número **N** é igual a **N vezes o fatorial de (N -1)**, ou seja : **N! = N(N-1)!**

Podemos notar que possuímos duas regras para a função fatorial, a saber:

⇨ - *Primeira regra*: **regra1-> 0! = 1** ou seja: **Fatorial de 0 =1**

⇨ - *Segunda regra*: **regra2-> N! = N*(N - 1)!**
ou seja Fatorial de **N = N x (Fatorial de (N - 1))**

Vamos ver um exemplo do cálculo do fatorial de um número, por exemplo **4!**.

? ⇨ **4!**

1-O fatorial de quatro, pela **regra2** é:
4 x 3! = 4 x 3 x 2! = 4 x 3 x 2 x 1! = 4 x 3 x 2 x 1x 0!
2- Pela **regra1**
0! =1

Assim, **4! = 4 x 3 x 2 x 1 x 1 = 24**.

A característica de uma função chamar ela mesmo várias vezes até encontrar uma regra que a faça parar é chamada de **recursividade**.

Definição de *Recursividade*

É um recurso, uma ferramenta computacional que permite a uma função chamar ela mesmo, através de uma ou mais regras com **condições de recursão**. Um programa bem elaborado utilizando recursão sempre deverá alcançar uma regra pré-definida que indique ao mesmo quando deverá interromper a recursão. A esta regra denominamos por: **condição de parada**.

Assim, uma função recursiva possui uma ou mais regras de recursão e pelo menos uma regra com a condição de parada. A condição de parada é a regra que vai dar sucesso à chamada da função interrompendo a recursão.

A recursividade pode ser considerada como um método eficaz para resolver um problema originalmente complexo, reduzindo-o em pequenas ocorrências do problema principal. Assim, segue a idéia do dividir para conquistar (divide and conquer), ou seja, para resolver um problema complexo, nós o dividimos em partes menores, nas quais a complexidade pode ser considerada menor em relação ao problema original. Resolvendo, isoladamente, cada uma das partes, podemos obter a solução do problema original como um todo.

No exemplo do **FATORIAL**, temos:

Uma regra com a **condição de recursão**:
FATORIAL n = n*FATORIAL (n-1)

Uma regra com a **condição de parada**:
FATORIAL 0 = 1.

No fatorial, a recursividade ocorreu da seguinte forma:

1- **FATORIAL 4 = 4 x FATORIAL 3**
➔ Pela regra **FATORIAL n = n*FATORIAL (n-1)** (condição de recursão)

2- **FATORIAL 3 = 3 x FATORIAL 2**
➔ Pela regra **FATORIAL n = n*FATORIAL (n-1)** (condição de recursão)

3- **FATORIAL 2 = 2 x FATORIAL 1**
➔ Pela regra **FATORIAL n = n*FATORIAL (n-1)** (condição de recursão)

4- **FATORIAL 1 = 1 x FATORIAL 0**
➔ Pela regra **FATORIAL n = n*FATORIAL (n-1)** (condição de recursão)

5- **FATORIAL 0 = 1**
➔ Pela regra **FATORIAL 0 = 1** (condição de parada)

Resumindo ainda mais a demonstração do processo recursivo, temos que:

FATORIAL 4 =
4 x (**FATORIAL 3**) =
4 x **(3 x (FATORIAL 2))** =
4 x (3 x **(2 x (FATORIAL 1))**) =
4 x (3 x (2 x **(1 x (FATORIAL 0))**)) =
4 x (3 x (2 x (1 x **(1)**))) =
24

Todo processo recursivo consiste em duas partes:

a. **Solução Trivial:** É conseguida por definição, ou seja, não é necessário fazer uso da recursividade para obtê-la.
b. **Solução Geral**: É a solução genérica que funciona em uma parte menor do problema original, mas que pode ser aplicada integralmente ao problema original.

O programa recursivo utiliza uma pilha e a preencherá com **" n"** números, que são os parâmetros passados para cada chamada recursiva. Depois do término da última chamada recursiva, serão multiplicados na mesma ordem.

A versão recursiva, no caso do fatorial, consome mais espaço de memória e levará muito mais tempo para executar, pois ela armazena e recupera todos os números, além de multiplicá-los.

Recursividade

Vantagens:
- É a maneira mais natural e lógica de resolver um problema.

Desvantagens:
- O constante uso da pilha pode levar a uma execução mais lenta.
- O não gerenciamento dinâmico de memória pode levar a um "estouro" da pilha

Exemplo do programa Fatorial (versão recursiva) em C++:

```
#include <iostream>

using namespace std;

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



int main()
{
   int num;

   cout << "Informe o numero:" ;
   cin >> num;
   cout << "FATORIAL= " << fat(num) << endl;
   return 0;
}
```