Recursividade

Estrutura de Dados

Introdução

Na arte, o efeito Droste é o nome que se dá para uma imagem que aparece dentro dela mesma.

Este efeito dá uma ideia de repetição, mas não uma repetição comum e sim uma onde o objeto se repete dentro dele mesmo.

Recursão é um processo parecido com este efeito, onde um objeto (no nosso caso uma função) é definido parcialmente dentro dele mesmo.

O cálculo da potência

Matematicamente, podemos definir a potência de 2 da seguinte forma:

$$2^n = \begin{cases} 1 & \text{se } n = 0 \\ 2.(2^{n-1}) & \text{se } n \geq 1 \end{cases}$$

É possível resolver este problema de forma convencional ou utilizando recursividade.

Potência sem recursividade

```
int potencia(int n) {
     int r = 1;

     if (n == 0) {
          return 1;
     } else {
          for(int i = 0; i < n; i++) {
               r = r * 2;
          }
          return r;
     }
}
```

Potência com recursividade

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

Condições para uma função recursiva

Para implementar uma função recursiva, é preciso estabelecer dois elementos:

- Uma condição de parada.
- Uma mudança de estado a cada chamada recursiva.

Caso um desses elementos não esteja presente, a recursão irá se repetir infinitamente.

Imprimindo números

Para exemplificar os elementos necessários de uma função recursiva, é apresentado o algoritmo para imprimir certa quantidade de números até 0:

```
void imprimir(int n) {
     cout << n << endl;
     if (n >= 1) {
          imprimir(n - 1);
     }
}
```

Caso não houvesse uma condição para parada *(`n >= 1`)* ou a chamada da função sem a mudança de estado *(`imprimir(n - 1)`)*, o programa executaria infinitamente.

O cálculo do fatorial

Matematicamente, podemos definir o fatorial de um número da seguinte forma:

$$n! = \begin{cases} 1 & \text{se } n = 0 \\ n * (n - 1) * (n - 2) \ldots * 3 * 2 * 1 & \text{se } n \geq 1 \end{cases}$$

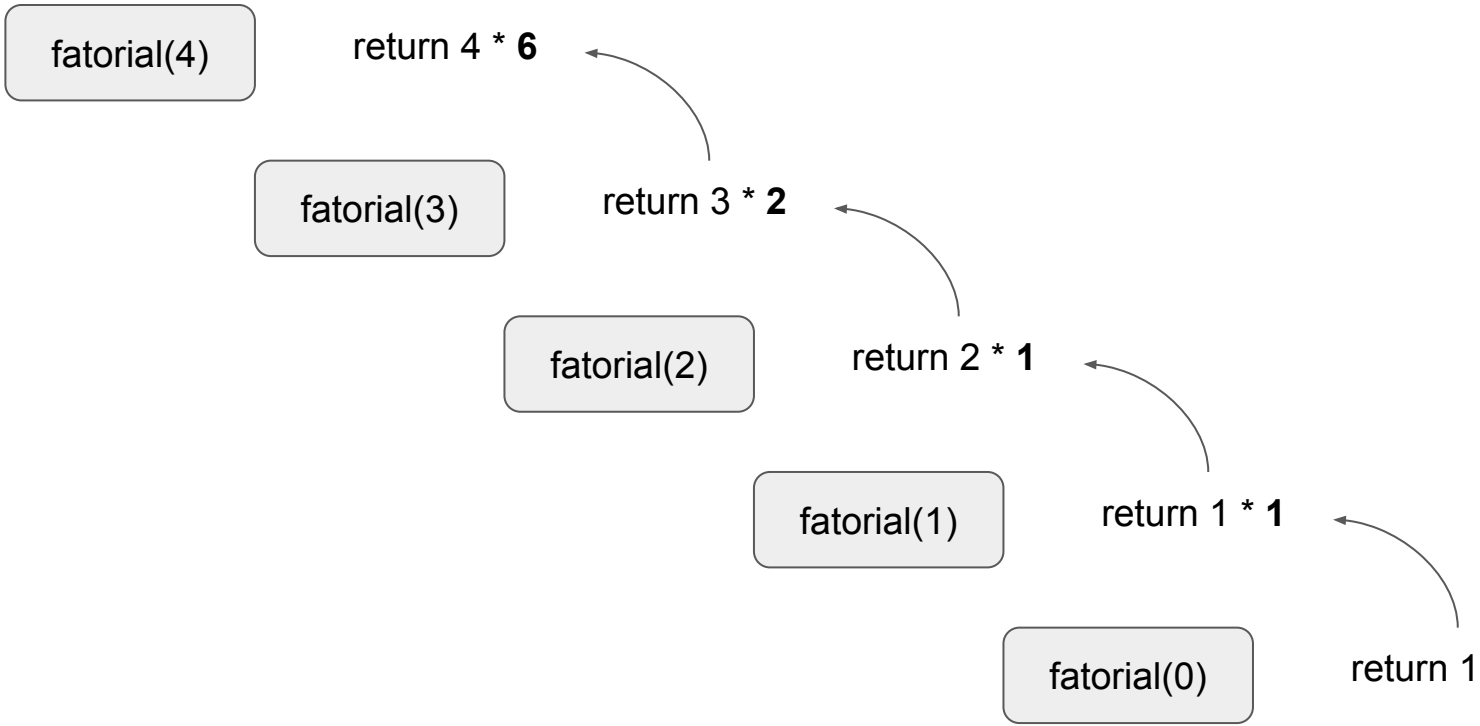
Fatorial sem recursividade

```
int fatorial(int n) {
     int r = 1;
     for (int i = n; i > 0; i--) {
          r = r * i;
     }
     return r;
}
```

Fatorial com recursividade

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

Teste de mesa do fatorial



Problemas relacionado a recursividade

Para cada chamada da função recursiva, um novo espaço na memória é alocado para empilhar a instância da função.

Se o número de chamadas for muito grande, pode haver um estouro de pilha e um erro de execução do programa.

Algoritmos recursivos consomem mais recursos computacionais, portanto, sua utilização deve ser avaliada com cautela.

A única vantagem da recursão é a escrita de códigos mais claros e enxutos.

A sequência Fibonacci

Matematicamente, podemos definir a sequência fibonacci da seguinte forma:

$$f(n) = \begin{cases} 0, & \text{se } n = 0 \\ 1, & \text{se } n = 1 \\ f(n - 1) + f(n - 2) & \text{se } n > 1 \end{cases}$$

Teste de mesa de Fibonacci