Recursividade

Uma rotina é dita recursiva se ela chama a si mesmo para obter um resultado.

Um exemplo comum de recursividade é a rotina para calcular fatorial.

Sabemos que:

n! = n * (n-1)!

1! = 1

0! = 1

A partir disso podemos construir a rotina fatorial.

```
#include <iostream>
using namespace std;
int fatorial(int n) {
   if (n > 1) {
      return n * fatorial(n-1);
   } else {
      return 1;
   }
}
int main()
{
   int n, f;
   cout << "Digite N: ";
   cin >> n;
   f = fatorial(n);
   cout << n << "! = " << f << endl;
   return 0;
}
```

É extremamente importante temos uma condição de parada para a recursão. Um teste deve ser efetuado para verificar o ponto onde a rotina não será mais chamada recursivamente. Esse teste normalmente é feito com os próprios parâmetros locais da rotina.

Um outro exemplo de recursividade é a série de Fibonacci. Sabemos que:

n-ésimo termo = (n-1) + (n-2)

segundo termo = 1

primeiro termo = 1

A partir disso podemos construir a rotina fibonacci.

```
#include <iostream>
using namespace std;
int fibonacci(int n) {
   if (n > 2) {
      return fibonacci(n-1) + fibonacci(n-2);
   } else {
      return 1;
   }
}
int main()
{
   int n, f;
   cout << "Digite N: ";
   cin >> n;
   f = fibonacci(n);
   cout << "Termo " << n << " = " << f << endl;
   return 0;
}
```

Não precisamos necessariamente ter recursão somente em uma rotina (recursão direta). Podemos ter recursão entre rotinas diferentes (recursão indireta), onde a primeira rotina chama a segunda rotina e a segunda rotina chama a primeira rotina recursivamente.

Um exemplo disso pode ser observado no cálculo de paridade de um número natural.

```
#include <iostream>
using namespace std;
int impar(int n);
```

```
int par(int n) {
   if (n == 0) {
      return 1;
   } else {
      if (n == 1) {
         return 0;
      } else {
         return impar(n-1);
      }
   }
}
int impar(int n) {
   if (n == 0) {
      return 0;
   } else {
      if (n == 1) {
         return 1;
      } else {
         return par(n-1);
      }
   }
}
int main()
{
   int n;
   cout << "Digite N: ";
   cin >> n;
   if (par(n)) {
      cout << "Numero Par" << endl;
   } else {
      cout << "Numero Impar" << endl;
   }
   return 0;
}
```

Exercícios

1) Faça uma rotina recursiva para calcular a somatória de todos os número de 1 a N (N será lido do teclado).

2) Faça uma rotina recursiva para o problema da Torre de Hanói.

O problema da Torre de Hanói consiste de três pinos, A, B e C, denominados origem, destino e trabalho, respectivamente, e n discos de diâmetros diferentes. Inicialmente, todos os discos se encontram empilhados no pino origem, em ordem decrescente de tamanho, de baixo para cima. O objetivo é empilhar todos os discos no pino destino, atendendo às seguintes restrições:

- Apenas um disco pode ser removido de cada vez.
- Qualquer disco não pode ser jamais colocado sobre outro de tamanho menor.

Utilize o programa a seguir como base.

```
#include <iostream>
using namespace std;
void moveDisco(char origem, char destino) {
   cout << origem << " -> " << destino << endl;
}
void torreHanoi(int altura, char origem, char destino, char trabalho) {
      ...
      moveDisco(origem, destino);
      ...
}
int main()
{
   int n;
   cout << "Digite N: ";
   cin >> n;
   torreHanoi(n, 'A', 'B', 'C');
   return 0;
}
```