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:

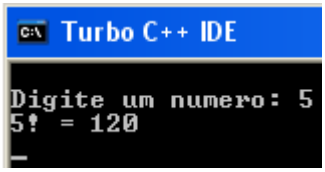
n! = n * (n-1)!

1! = 1

0! = 1

A partir disso podemos construir a rotina fatorial.

```
1  #include <stdio.h>
2  #include <conio.h>
3
4  int fatorial(int n) {
5     if (n > 1) {
6        return n * fatorial(n-1);
7     } else {
8        return 1;
9     }
10 }
11
12 void main() {
13    int n, f;
14    clrscr();
15    printf("\nDigite um numero: ");
16    scanf("%d", &n);
17    f = fatorial(n);
18    printf("%d! = %d\n", n, f);
19    getch();
20 }
```



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:

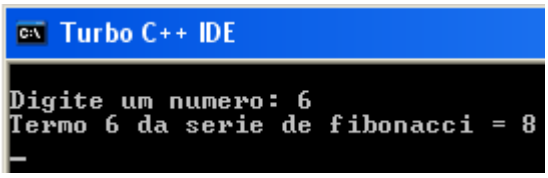
n-ésimo termo = (n-1) + (n-2)

segundo termo = 1

primeiro termo = 1

A partir disso podemos construir a rotina fibonacci.

```
1  #include <stdio.h>
2  #include <conio.h>
3
4  int fibonacci(int n) {
5     if (n > 2) {
6        return fibonacci(n-1) + fibonacci(n-2);
7     } else {
8        return 1;
9     }
10 }
11
12 void main() {
13    int n, f;
14    clrscr();
15    printf("\nDigite um numero: ");
16    scanf("%d", &n);
17    f = fibonacci(n);
18    printf("Termo %d da serie de fibonacci = %d\n", n, f);
19    getch();
20 }
```



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.

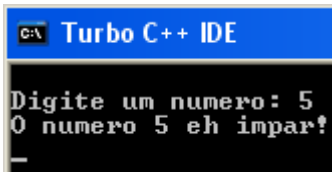
```c
#include <stdio.h>
#include <conio.h>

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);
   }
}

void main() {
   int n;
   clrscr();
   printf("\nDigite um numero: ");
   scanf("%d", &n);
   if (par(n)) {
      printf("O numero %d eh par!\n", n);
   } else {
      printf("O numero %d eh impar!\n", n);
   }
   getch();
}
```



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.

```
1  #include <stdio.h>
2  #include <conio.h>
3
4  void moveDisco(char origem, char destino) {
5     printf("%c -> %c\n", origem, destino);
6  }
7
8  void torreHanoi(int altura, char origem, char destino, char trabalho) {
9     ...
10    moveDisco(origem, destino);
11    ...
12 }
13
14 void main() {
15    int n;
16    clrscr();
17    printf("\nDigite a altura da torre: ");
18    scanf("%d", &n);
19    torreHanoi(n, 'A', 'B', 'C');
20    getch();
21 }
```