Universidade do Estado de Santa Catarina (UDESC)
Prof. Rui Tramontin
Disciplina: Algoritmos (AGT)
Curso: Ciência da Computação

Funções: Exemplos e Prática em Laboratório

Exemplo de função: Cálculo do fatorial

```
ALGORITMO "Fatorial"
VAR
  N, F: INTEIRO

FUNÇÃO FATORIAL(X: INTEIRO): INTEIRO
VAR
  FAT: INTEIRO
INÍCIO
  FAT <- 1
  ENQUANTO X > 0 FAÇA
    FAT <- FAT * X
    X <- X - 1
  FIMENQUANTO
  RETORN FAT
FIMFUNÇÃO

INÍCIO
  LEIA(N)
  F <- FATORIAL(N)
  ESCREVAL(N,"! é ", F)
FIMALGORITMO
```

Observações:

- Cuidado ao transcrever o código no Visualg, pois a ferramenta não reconhece caracteres acentuados e com cedilha;
- Executando o algoritmo passo a passo (pressionando-se F8), pode-se notar que as variáveis globais N e F são alocadas inicialmente, sendo que o parâmetro X da função e a variável local FAT são alocados quando a função é executada;
- Quando o fluxo de execução retorna ao algoritmo principal (após o comando RETORNE), X e FAT são desalocadas da memória;
- A área de memória onde as variáveis locais e os parâmetros de funções são alocadas é chamada Pilha (*Stack*). Quando uma função chama outras funções (ou mesmo ela mesma, num processo chamado *recursão*), novas variáveis são alocadas (empilhadas) na pilha, sendo desalocadas (desempilhadas) quando as respectivas funções são encerradas.

Recursão e Funções Recursivas

Recursão é o nome que se dá ao processo que ocorre quando uma função chama a si mesma. Inicialmente pode-se pensar que isso pode gerar um ciclo infinito, pois sucessivas chamadas ocorrerão sem um critério de parada definido. De fato, isso pode ocorrer, caso a função não seja escrita corretamente.

Além disso, é mais fácil ocorrer uma sobrecarga na pilha (*stack overflow*) devido à chance de um alto número de chamadas sucessivas preencher toda a memória reservada para a pilha.

Uma função recursiva é dividida basicamente em duas partes:

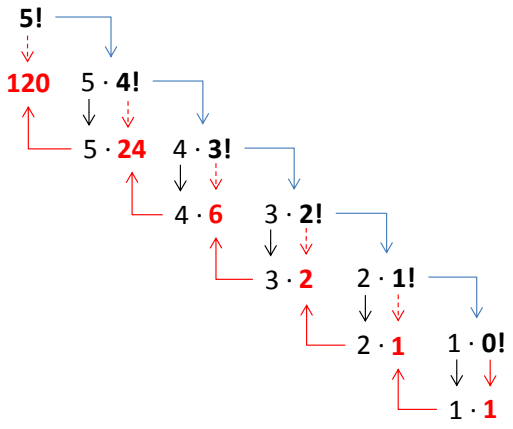
1. Um ou mais **casos base**, onde a função retorna um valor simples (por exemplo, o fatorial de 0 ou de 1 é igual a 1; a potenciação de um valor elevado a 0 é igual a 1);
2. Um **caso geral**, onde a função chama a si mesma, ou seja, ocorre a recursão (por exemplo, o fatorial de x é igual a x * (x-1)!).

Assim, espera-se que as sucessivas recursões se encerrem no momento que um caso base seja alcançado, de tal forma que os retornos das chamadas realizem o processamento esperado. Ao final, o retorno da chamada da função, após realizar a recursão, terá o valor calculado.

Por exemplo, tomemos a definição recursiva da função fatorial:

$$fatorial(n) = \begin{cases} 1 & \text{se } n = 0 \\ n \times fatorial(n-1) & \text{se } n > 0 \end{cases}$$

O caso base retorna 1, quando n for 0. Caso contrário, a função faz uma chamada recursiva, decrementando o valor de n. Por exemplo, o fatorial de 5 gerará a seguinte sequência de recursões:



As setas azuis representam as chamadas recursivas. A cada chamada, a função fica "esperando" o retorno, que ocorrerá ao final quando se chega ao caso base. Os retornos são representados pelas setas vermelhas, sendo as setas vermelhas tracejadas são os resultados implícitos de cada chamada.

A implementação em linguagem de programação (no nosso caso, em pseudocódigo) de uma função recursiva é bem simples, bastando transcrever os casos usando a estrutura condicional. A função fatorial é definida recursivamente da seguinte maneira, conforme o código a seguir:

```
ALGORITMO "Fatorial Recursivo"
VAR
  N, F: INTEIRO

FUNÇÃO FATORIAL_REC(X: INTEIRO): INTEIRO
INÍCIO
  SE X = 0 ENTÃO
    RETORNE 1
  SENÃO
    RETORNE X * FATORIAL_REC(X - 1)
  FIMSE
FIMFUNÇÃO

INÍCIO
  LEIA(N)
  F <- FATORIAL_REC(N)
  ESCREVAL(N,"! é ", F)
FIMALGORITMO
```

Uma boa forma de entender o fluxo de execução durante a recursão é executar passo a passo (F8) para ver os parâmetros sendo empilhados na memória e, ao final, os resultados parciais sendo retornados e desempilhados.

Um aspecto que é importante mencionar é o fato de que qualquer algoritmo implementado de maneira iterativa (ou seja, usando-se estruturas de repetição) também pode ser implementado de maneira recursiva, e vice-versa. Apesar dessa equivalência, existem situações onde uma das maneiras é mais fácil de implementar do que outra. No caso do fatorial, ambas as formas são simples, como foi apresentado nos exemplos.

Há casos onde a implementação recursiva é bastante ineficiente, fazendo o computador realizar cálculos repetidos, devido à forma como a versão recursiva é definida. Para ilustrar essa situação tomemos uma função para calcular o n-ésimo termo da sequência de Fibonacci. Dada a série 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ... a função FIBO(N) retorna o n-ésimo termo. Por exemplo, FIBO(7) retorna 13.

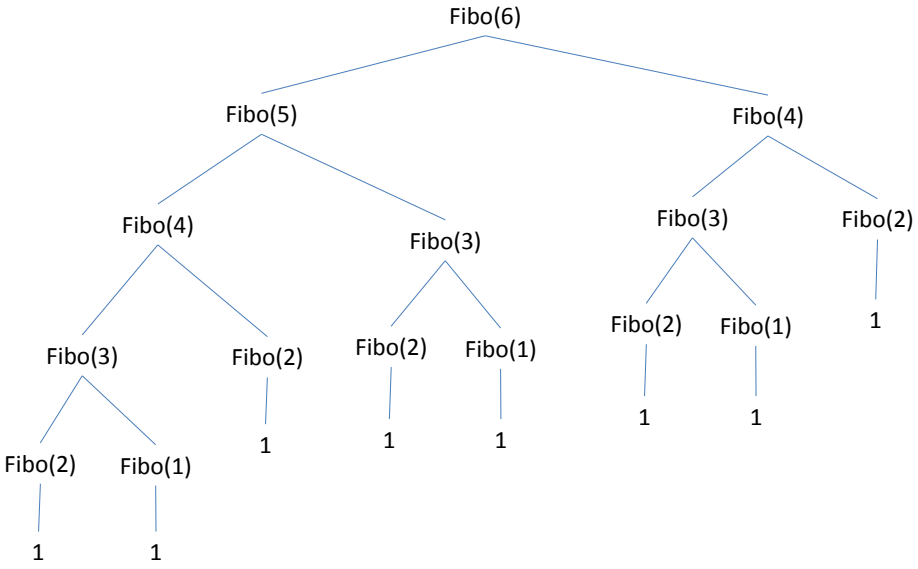
A definição recursiva dessa função é a seguinte:

Fibo(1) = 1 // caso base, pois o 1º termo é 1.

Fibo(2) = 1 // caso base, pois o 2º termo é 1.

Fibo(n) = Fibo(n-1) + Fibo(n-2) // caso geral, soma dos dois termos anteriores.

Esta função é muito mais simples de programar do que a versão iterativa. No entanto, esta versão é muito ineficiente, pois são feitas duas chamadas recursivas. Isto acaba gerando uma explosão combinatória, com o computador tendo que realizar o mesmo cálculo várias vezes. Para ilustrar esta situação, vejamos a sequência de chamadas quando é preciso gerar o 6º termo:



Para calcular Fibo(6) temos de calcular Fibo(5) e Fibo(4). Por sua vez, para calcular Fibo(5) temos de calcular Fibo(4) e Fibo(3), e assim sucessivamente. Este tipo de processamento é ineficiente porque o computador realiza cálculos repetidamente, que já foram calculados em outras chamadas. Numa implementação iterativa, tal como já apresentado na disciplina, cada termo é calculado somente uma vez, sendo muito mais eficiente.

Para finalizar, execute o algoritmo a seguir, que gera uma sequência de K termos, sendo K definido pelo usuário. Veja quanto tempo leva para gerar 30 termos (boa sorte ;-)).

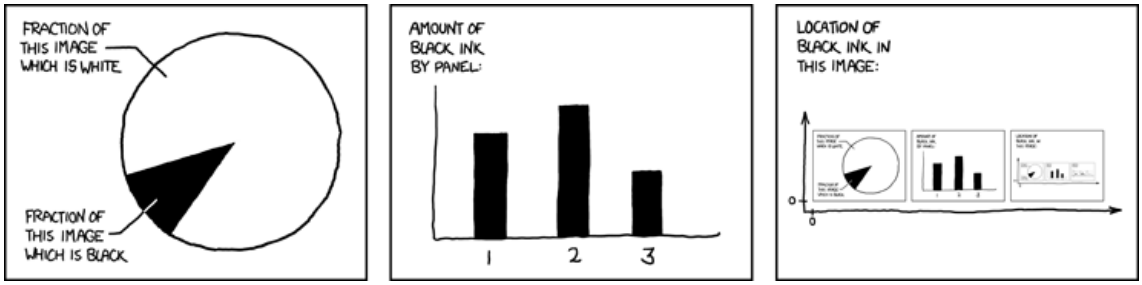
```
ALGORITMO "Fibonacci Recursivo"
VAR
  K, I: INTEIRO

FUNÇÃO FIBO(N: INTEIRO): INTEIRO
INÍCIO
  SE (N = 1) OU (N = 2) ENTÃO
    RETORNE 1
  SENÃO
    RETORNE FIBO(N - 1) + FIBO(N – 2)
  FIMSE
FIMFUNÇÃO

INÍCIO
  LEIA(K)
  PARA I DE 1 ATÉ K FAÇA
    ESCREVAL(I, " : ", FIBO(I))
  FIMPARA
FIMALGORITMO
```

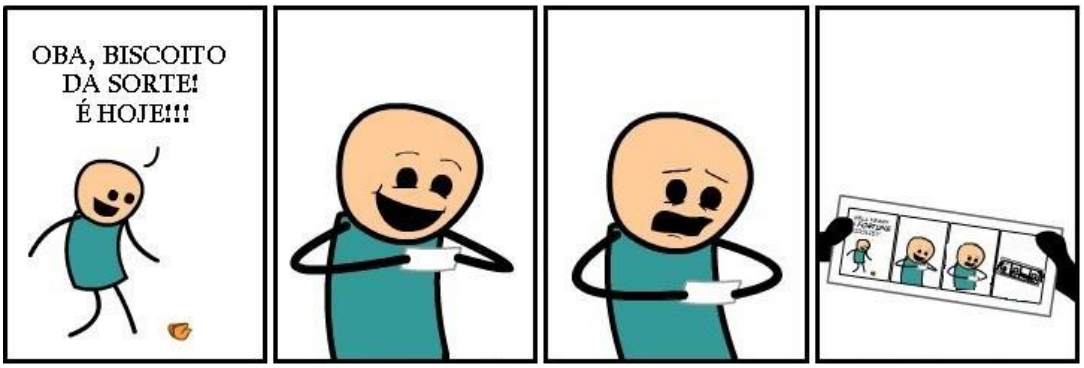
Epílogo (para rir um pouco)

Self-Description



Fonte: https://xkcd.com/688/

"Para entender a recursividade, antes, você tem que entender a recursividade."



Fonte: http://geraldoferraz.blogspot.com.br/2011/11/recursividade.html