



Recursão

Recursão é um método de resolução de problemas que envolve quebrar um problema em subproblemas menores e menores até chegar a um problema pequeno o suficiente para que ele possa ser resolvido trivialmente. Normalmente recursão envolve uma função que chama a si mesma (recursão direta). Embora possa não parecer muito, a recursão nos permite escrever soluções elegantes para problemas que, de outra forma, podem ser muito difíceis de programar. Toda função recursiva deve obedecer a três leis importantes:

1. Deve chamar a si mesmo, recursivamente;
2. Deve ter um caso básico (chamado de **caso base**);
3. Deve mudar o seu estado e se aproximar do caso básico (chamado de **passo recursivo**).

Exercícios

1. (ex1.c) Teste o seguinte programa, fornecendo o valor 6 e outros valores como entrada. As mensagens escritas mostram em que ordem cada execução da função fatorial é iniciada e terminada.

```
1  #include <stdio.h>
2
3  int fatorial(int n){
4      printf("Iniciando a funcao fatorial(%d)\n", n);
5      if (n <= 1){
6          printf("Finalizando a funcao fatorial(%d) com valor 1\n", n);
7          return 1;
8      }
9      int fat = fatorial(n - 1);
10     printf("Finalizando a funcao fatorial(%d) com valor %d\n", n, n*fat);
11     return n*fat;
12 }
13
14 int main() {
15     int n, fat;
16     scanf("%d", &n);
17     fat = fatorial(n);
18     printf("%d! = %d\n", n, fat);
19
20     return 0;
21 }
```

Você pode utilizar o site C Tutor para facilitar a visualização da execução desse código, nesse caso, troque a leitura do valor de n (`scanf("%d", &n);`) por `n = 6`.

Obs.: A função poderia, é claro, ser feita sem todos os comandos de escrita, e o retorno no final da função poderia ser `return n*fatorial(n-1)`, dispensando o uso da variável `fat`.

2. (ex2.c) Analise as funções recursivas abaixo e descubra qual o problema de cada uma. Após corrigir, execute o código para ter certeza que tudo funcionou. Caso não esteja conseguindo descobrir onde está o problema, utilize o site C Tutor para te auxiliar.

```

1  #include <stdio.h>
2
3  int f1(int n){
4      /* Retorna a qtd de números naturais pares menores ou iguais a n */
5      if (n % 2 == 0)
6          return 1 + f1(n - 2); //Por que n - 2?
7      else
8          return f1(n - 1); //Por que n - 1?
9  }
10
11 int f2(int n){
12     /* Retorna a soma dos números naturais menores ou iguais a n */
13     return (n == 1) ? 1 : (n + f2(n + 1)); // Operador ternário em C
14 }
15
16 int main() {
17     printf("f1(%d) = %d\n", 10, f1(10));
18     printf("f2(%d) = %d\n", 10, f2(10));
19     return 0;
20 }
```

Note a utilização do operador ternário em C (ele é muito similar ao Operador Ternário de Python). Para saber mais: <https://www.geeksforgeeks.org/conditional-or-ternary-operator-in-c-c/>

3. (ex3.c) A *sequência de Fibonacci* foi descrita pelo matemático italiano Leonardo Pisano, também conhecido por Leonardo Fibonacci, quando estudava o crescimento de uma população de coelhos. Mais tarde verificou-se que a sequência aparece em muitas outros campos do conhecimento. A sequência é definida recursivamente como

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

Os primeiros valores da série são: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144,

Uma função recursiva para calcular $Fibo(n)$ é simples:

```

1 int fibo(int n){
2     if (n < 2)
3         return n;
4     else
5         return fibo(n - 1) + fibo(n - 2)
6
7 }

```

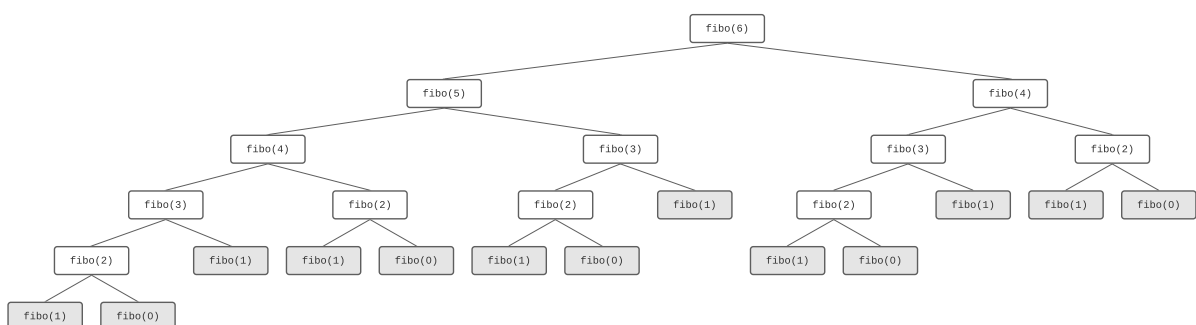
a) Execute o código abaixo fornecendo valores pequenos para n , como 5 ou 6. Note quantas vezes a função `fibo` é chamada.

```

1 #include <stdio.h>
2
3 int fibo(int n){
4     printf("Iniciando a funcao fibo(%d)\n", n);
5     if (n < 2){
6         printf("Finalizando a funcao fibo(%d) com valor %d\n", n, n);
7         return n;
8     }
9
10    int f = fibo(n - 1) + fibo(n - 2);
11    printf("Finalizando a funcao fibo(%d) com valor %d\n", n, f);
12    return f;
13
14 }
15
16 int main() {
17     int num, fibonacci;
18     scanf("%d", &num);
19     fibonacci = fibo(num);
20     printf("fibo(%d) = %d\n", num, fibonacci);
21     return 0;
22 }

```

A figura abaixo ilustra as chamadas recursivas da função para $n = 6$. Tente seguir os passos do programa por ela. Na chamada da função, o programa “desce” um nível. E no retorno da função, ele “sobe” de volta.



Na Ciência da Computação, esse formato é chamado árvore. Nesse caso, árvore de recursão, sendo a primeira chamada (ou seja, `fib(6)`) a raiz da árvore, e as chamadas não recursivas (`fib(1)` e `fib(0)`) as folhas da árvore.

- b) Observe que algumas chamadas recursivas (por exemplo, `fib(3)`) são calculadas várias vezes. Por isso, para esse cálculo é claramente mais vantajoso fazer uma função iterativa, em vez da função recursiva, a não ser que os valores já calculados sejam de alguma forma armazenados para não serem calculados novamente. Faça uma função iterativa para calcular os termos da série. Note que ela não é tão simples quanto a recursiva, porém é mais eficiente. Meça o tempo de execução¹ da versão recursiva e da versão iterativa para diferentes valores de n .

4. (ex4.c) Analise as funções recursivas abaixo.

```

1  int M(int a, int b) {
2      if (b == 0)
3          return 0;
4      else if (b == 1)
5          return a;
6      else
7          return a + M(a, b - 1);
8  }
9  int DI(int a, int b) {
10     if (a < b)
11         return 0;
12     else
13         return 1 + DI(a - b, b);
14 }
15 double RD(double a, double b) {
16     if (a < b)
17         return a;
18     else
19         return RD(a - b, b);
20 }
```

- a) Qual o objetivo de cada função?
- b) Qual será a saída quando for executado, dentro da função main, os comandos abaixo?

```

1  printf("%d\n", M(2, 3));
2  printf("%d\n", M(4, 2));
3  printf("%d\n", DI(2, 3));
4  printf("%d\n", DI(3, 2));
5  printf("%d\n", DI(4, 2));
6  printf("%d\n", DI(10, 6));
7  printf("%lf\n", RD(2, 3));
8  printf("%lf\n", RD(3, 2));
9  printf("%lf\n", RD(25.5, 10.5));
```

¹Para medir o tempo, você pode usar a função `clock()` da biblioteca `time.h` ou, se estiver no Linux, você pode usar o comando `'time ./executavel'`

5. (ex5.c) Analise o código abaixo.

```
1  #include <stdio.h>
2
3  void imprime(char s[]){
4      if (s[0] == '\0')
5          return;
6      printf("%c", s[0]);
7      imprime(&s[1]);
8  }
9
10 int main() {
11     char msg[257]; //Última posição para o '\0'
12     //Lê 256 caracteres ou até encontrar um fim de linha (ou seja, um \n)
13     fgets(msg, 256, stdin);
14     imprime(msg);
15     return 0;
16 }
```

- a) O que você espera da execução do seguinte programa? Execute e confira o resultado.
 - b) Coloque o comando `printf("%c", s[0]);` da função após a chamada recursiva, ou seja, coloque-o como último comando da função. O que você espera do resultado? Execute e veja a diferença.
6. (ex6.c) Escreva uma função recursiva para calcular a n -ésima potência de um número, ou seja, x^n . O seu algoritmo deve ter complexidade $O(\log n)$. Como você deve imaginar, você não pode usar a função `pow` ou outra similar.