Lab. 15

Funções Recursivas

ECT2303 - T02 - 19.1

A recursão é um mecanismo alternativo às implementações iterativas baseado na construção de função que chamam a si mesma. Embora seja menos eficientes que seus equivalentes iterativos em linguagens como C++, ela é útil em vários problemas, pois permite uma descrição mais intuitiva da solução de um problema.

A solução recursiva dos problemas envolve duas etapas importantes:

1. A definição de um **caso base**, que é aquele onde se conhece o valor da solução. Este caso serve de referência para o caso recursivo;
2. A definição de um **caso recursivo**, que é aquele cuja definição faz chamada a própria função, e deve sempre ser construído de maneira a ser reduzido/expandido ao caso base.

Os problemas a seguir servirão como prática para o estudo de recursão.

15.1 Exercícios de Fixação

1. O que o seguinte programa faz?

```
1  #include <iostream>
2
3  using namespace std;
4
5  int func( int, int );
6
7  int main()
8  {
9    int x, y;
10
11   cout << "Digite dois inteiros: ";
12   cin >> x >> y;
```

```
13    cout << " O resultado eh: " << func( x, y ) << endl;
14
15    return 0;
16  }
17
18  int func( int a, int b)
19  {
20    if ( b == 1 )
21      return a;
22    else
23      return a + func( a, b - 1 );
24  }
```

2. Sendo a definição matemática para o fatorial de um número n dada por

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

implemente esta função de forma recursiva. Implemente também a função `main`, de modo que o usuário do seu programa possa informar um número inteiro n e visualizar o fatorial de $0 - n$ na tela.

3. Sendo a definição matemática para a soma dos números de 0 a n dada por

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

implemente esta função de forma recursiva. Implemente também a função `main`, de modo que o usuário do seu programa possa informar um número inteiro n e visualizar a soma de todos os números de 0 a n na tela.

4. Escreva um função recursiva **`fibonacci( n )`**, que calcula o n-ésimo número de Fibonnaci.

5. Uma função matemática $f(n)$ é dada por

$$f(n) = \sum_{i=1}^{n} \frac{2i}{i+1}.$$

Implemente uma função recursiva que calcula o valor do somatório para um dado valor de n. Utilizando a função implementada, faça um programa que receba como entrada um número inteiro n e imprima na tela o resultado de $f(n)$.

6. Uma função matemática $f(n)$ é dada por

$$f(n) = \prod_{i=1}^{n} 2i.$$

Implemente uma função recursiva que calcula o valor do produtório para um dado valor de n. Utilizando a função implementada, faça um programa que receba como entrada um número inteiro n e imprima na tela o resultado de $f(n)$.

7. Em uma sequência numérica, o termo b_k pode ser rescrito como:

$$b_k = b_{k-1} + 2b_{k-2}$$

A sequência é inicializada nos termos $b_1 = 1$ e $b_2 = 2$.
Implemente uma função recursiva que recebe o valor de k e determina o k-ésimo termo b_k da série. Utilizando a função implementada, faça um programa que receba como entrada um número inteiro k e imprima na tela o termo b_k obtido.

Exemplos de execução:

```
--Exemplo 1:
Escreva o valor de n:
2
O termo 2 da sequência eh 2
--Exemplo 2:
Escreva o valor de x:
6
O termo 6 da sequência eh 32
```

8. Uma sequência de números $a(0)$, $a(1)$, $a(2)$,...,$a(n)$,... é definida matematicamente por

$$a(n) = \begin{cases} a(0) = 1 \\ a(1) = 2 \\ a(n) = 2a(n-2) - a(n-1). \end{cases}$$

Implemente uma função recursiva que computa o enésimo termo (termo $a(n)$) da sequência. Utilizando a função implementada, faça um programa que leia um número inteiro n e imprima na tela todos os $a(0)$, $a(1)$, $a(2)$,...,$a(n)$ termos da sequência.

9. Implemente uma função recursiva que calcula o produto entre dois números inteiros. **Não utilize o operador** $*$. Implemente também a função `main`, de modo que o usuário do seu programa possa informar dois números inteiros e visualizar o produto computado na tela. **Dica:** para achar o passo base e passo recursivo, leve em consideração que, por exemplo, $2 * 3 = 2 + 2 * 2 = 2 + 2 + 2 * 1...$ e que $2 * 0 = 0$.

10. Escreva uma função recursiva `pot(base, expoente )` que, quando invocada, retorna $base^{expoente}$. Por exemplo, `pot( 3, 4 ) = 3 * 3 * 3 * 3`. Suponha que `expoente` é um inteiro maior que ou igual a 1.

Dica: O passo de recursão utilizaria o relacionamento $base^{expoente} = base \cdot base^{expoente-1}$ e a condição de terminação ocorre quando `expoente` é igual a 1 porque $base^1 = base$.

Exemplo de execução:

```
Digite a base e um expoente: 2 32

2 elevado a 32 é 4294967296
```

11. Escreva uma função recursiva que permita imprimir na tela todos os dígitos de um número inteiro, **considerando mostrá-los em ordem inversa**, isto é, se o número passado for o número 1234, deve-se primeiro exibir o dígito 4, depois o 3, depois o dígito 2 e depois o dígito 1. Crie também a função `main`, que recebe o valor (inteiro a ser impresso).

12. Escreva uma função recursiva que permita imprimir na tela todos os dígitos de um número inteiro, **considerando mostra-los em ordem direta**, isto é, se o número passado for o número 1234, deve-se primeiro exibir o dígito 1, depois o dígito 2, depois o dígito 3 e depois o dígito 4. Crie também a função `main`, que recebe o valor (inteiro a ser impresso).

15.2 Referências Bibliográficas

1. MANZANO, J.A.; OLIVEIRA, J.F.; **Algoritmos - Lógica para Desenvolvimento de Programação**. Editora Erica.

2. ASCENCIO, A.F.G.; CAMPOS, E.A.V. **Fundamentos da Programação de Computadores - Algoritmos, Pascal e C/C++**. 3ed. Editora Pearson.

3. DEITEL, H. M.; DEITEL, P. J. **C++ Como Programar**. 3ed. Editora Bookman.