LABORATÓRIO DE PROGRAMAÇÃO AVANÇADA
DÉCIMO SEGUNDO TRABALHO PRÁTICO
- - FUNÇÕES RECURSIVAS - -

Este trabalho visa o uso de Funções Recursivas. Só relembrando que uma função recursiva é uma função que se refere a si própria, cuja ideia consiste em utilizar a função que está sendo definida na sua própria definição. Um exemplo clássico é o problema do fatorial, que pode ser definido recursivamente da seguinte forma:

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

Em todas as funções recursivas existe:
(a) Um passo básico (ou mais) cujo resultado é imediatamente conhecido; e
(b) Um passo recursivo em que se tenta resolver um sub-problema do problema maior.

Se analisarmos a função fatorial...

- o caso básico é o teste de igualdade a zero, cujo resultado imediato é 1;
- e o passo recursivo é `n * fatorial (n-1)`.

A execução de uma função recursiva consiste em ir resolvendo sub-problemas sucessivamente mais simples até se atingir o caso mais simples de todos, cujo resultado é imediato. Os erros mais comuns associados às funções recursivas são, naturalmente:

- Não detectar os casos simples
- A recursão não diminuir a complexidade do problema.
- No caso de erro em função recursivas, o mais usual é a recursão nunca parar.

Exemplo

Considere uma versão extremamente primitiva da linguagem C, em que as únicas funções numéricas existentes são **`zero`** e duas funções **`incr`** e **`decr`**, que incrementa/decrementa, respectivamente, seu argumento em uma unidade. Isto implica que as operações >, <, = e similares não podem ser utilizadas. Portanto, assuma que as funções **`zero`**, **`incr`** e **`decr`** já estão definidas, da seguinte forma:

```
int zero (int x)
{
  return (x==0);
}

int decr (int x)
{
  return (--x);
}

int incr (int x)
{
  return (++x);
}
```

Exemplo – Parte 1:

Nesta linguagem, defina a função **`menor`**, que recebe dois número inteiros positivos e determina se o primeiro argumento é numericamente inferior ao segundo.

Resposta:

```
int menor (int x, int y)
{
  if (zero(y)) {
    return 0;     // retorne FALSO
  }
  else {
    if (zero(x)) {
      return 1;  // retorne VERDADEIRO
    }
    else {
      return menor(decr(x), decr(y));
    }
  }
}
```

Note que, no problema de encontrar o menor, há DOIS casos básicos, isto é, quando **`y=0`**, ou **`x = 1`**. Estamos considerando que 0 significa **FALSO** e 1 significa **VERDADEIRO**.

Exemplo – Parte 2:

Faça uma função main que use a função **`menor`**.

Resposta:

```
void main()
{
  int n1, n2;
  printf ("Entre com 2 numeros: ");
  scanf ("%d %d", &n1, &n2);
  if (menor(n1,n2)) {
    printf ("%d eh MENOR que %d\n", n1, n2);
  }
  else {
    printf ("%d nao eh MENOR que %d\n", n1, n2);
  }
}
```

EXERCÍCIOS:

Agora vamos aplicar o que discutimos anteriormente através de um exercício dividido em 8 partes:

1) Considere que nessa versão do C não exista o operador de igualdade. Defina a função recursiva **`igual`**, que recebe dois número inteiros **positivos** e retorne 1, se estes são iguais, ou 0 se não. Implemente uma função **`main`** que use a função **`igual`** para testar 10 valores inteiros positivos entre 1 e 1000 gerados aleatoriamente.

2) Até ao momento, essa linguagem apenas trabalha com números inteiros positivos. Admitindo que as operações **incr**, **decr** e **zero** também funcionam com números negativos, defina a função recursiva **simétrico** que recebe um número inteiro **positivo** e retorna o seu simétrico. Por exemplo, a chamada de **simetrico(3)** vai retornar o valor **-3**; a chamada de **simetrico(100)** vai retornar o valor **-100**. Implemente uma função **main** que use a função **simetrico** para testar 10 valores inteiros positivos entre 1 e 150 gerados aleatoriamente.

3) Implemente a função recursiva **soma** de dois números inteiros positivos apenas recorrendo às funções incrementa e decrementa (**incr** e **decr**). Implemente uma função **main** que use a função **soma** para testar 15 pares de valores inteiros positivos entre 10 e 100, ambos gerados aleatoriamente.

4) Faça uma função recursiva **mult** que multiplique dois números inteiros positivos através de somas sucessivas. Por exemplo: 6 * 4 = 4 + 4 + 4 + 4+ 4 + 4). Use a função **soma** definida no exercício anterior. Implemente uma função **main** que use a função **mult** para testar 20 pares de valores inteiros positivos entre 4 e 40, ambos gerados aleatoriamente.

A partir daqui, não precisa mais usar as funções da **versão extremamente primitiva da linguagem C** (**zero**, **incr**, **decr**, etc).

5) Faça uma função recursiva que receba o valor de **n** e calcule o somatório: `1 + 1/2 - 1/3 + 1/4 - 1/5 ... +/- 1/n`. O valor de **n**, deve ser passado como argumento do programa (**argc** e **argv**).

6) A recursividade pode ser utilizada para gerar todas as possíveis permutações de um conjunto de símbolos. Por exemplo, existem seis permutações no conjunto de símbolos A, B e C: ABC, ACB, BAC, BCA, CBA e CAB. O conjunto de permutações de **N** símbolos é gerado tomando-se cada símbolo por vez e prefixando-o a todas as permutações que resultam dos (N - 1) símbolos restantes. Consequentemente, permutações num conjunto de símbolos podem ser especificadas em termos de permutações num conjunto menor de símbolos. Receba um conjunto de **k** símbolos (no caso, caracteres) e imprima todas as permutações. Os **k** símbolos devem ser passados como parâmetro do programa (**argc** e **argv**).

7) Considere uma partida de futebol entre duas equipes A x B, cujo placar final é **m** x **n**, em que **m** e **n** são os números de gols marcados por A e B, respectivamente. Implemente um programa recursivo em C que imprima todas as possíveis sucessões de gols marcados. Por exemplo, para um resultado de 3 x 1 as possíveis sucessões de gols são "A A A B", "A A B A", "A B A A" e "B A A A". Os valores de **m** e **n** devem ser passados como argumento do programa (**argc** e **argv**).

8) Faça uma função recursiva que gere a sequência dada por:
```
F(1) = 1
F(2) = 2
F(n) = 2 * F(n - 1) + 3 * F(n - 2)
```
O valor de **n**, deve ser passado como argumento do programa (**argc** e **argv**).

9) Euclides. A seguinte função calcula o maior divisor comum dos inteiros positivos m e n. Escreva uma função recursiva equivalente. Os valores de m e n devem ser passados como argumento do programa (**argc** e **argv**).

```
int Euclides( int m, int n) {
   int r;
   do {
      r = m % n;
      m = n;
      n = r;
   } while (r != 0);
   return m;
}
```

10) Escreva uma função recursiva que receba dois inteiros positivos k e n e calcule k*n. Suponha que k*n cabe em um long int. Os valores de k e n devem ser passados como argumento do programa (**`argc`** e **`argv`**).

Este trabalho deve ser entregue no dia **09/07/2015** (quinta).
IMPORTANTE! Após esta data, o trabalho não será mais aceito.
O trabalho deve ser possível ser compilado pelo **GCC** e executado no **Linux**.
Envie para o professor (**xbarretox@gmail.com**) e o monitor (**gabriel.leitao@gmail.com**).
Coloque no assunto: [**LPAV-TP12**].