Interpretador Hall e os números pares e ímpares

Um exemplo básico de algoritmo quando se está estudando lógica de programação, consiste em verificar se um dado número é par ou ímpar. O usuário informa um número e o programa deve ser capaz de decidir se o número informado é par ou ímpar. Isto é muito fácil no interpretador Hall, uma vez que estão disponíveis as funções **e_par(x)** e também **e_impar(x)**. O argumento **x** dessas funções é o valor que foi informado pelo usuário.

Vejamos um exemplo...

Exemplo da função e_impar()

```
//  programa: impar.hal
//     autor: fernandopaim@paim.pro.br
// descricao: verifica se um numero e impar


algoritmo()
{
        inteiro n;

        leia("informe o valor de N: ",n);

        se (E_Impar(n)==1)
        {
                escreva("o numero informado eh impar.");
        }
        senao
        {
                escreva("o numero informado nao eh impar.");
        }
}
```

Comentários:

O programa é bem simples e o item importante é a chamada à função **e_impar**. O comportamento da função **e_impar** é o seguinte:

- se o número passado como argumento para a função for um número ímpar, a função retorna o valor 1, caso contrário, o valor retornado é 0 (zero).

Assim, no algoritmo, fazemos o teste **se (e_impar(n) == 1)**... Isto é, se o valor de retorno da função **e_impar** com argumento **n** for igual a **1**, então o número **n** é ímpar.

Funcionalidade adicional:

O *interpretador Hall* considera que qualquer valor diferente de zero deve ser

interpretado como verdadeiro (true) e, obviamente, o valor zero é interpretado como falso (false). Desse modo, como a instrução **se** é uma instrução de teste booleano (verdadeiro ou falso), o programa acima pode ser reescrito da seguinte forma, veja abaixo...

Exemplo da função e_impar()

```
// descricao: verifica se um numero e impar


algoritmo()
{
        inteiro n;

        leia("informe o valor de N: ",n);

        se (E_Impar(n))
        {
                escreva("o numero informado eh impar.");
        }
        senao
        {
                escreva("o numero informado nao eh impar.");
        }
}
```

Comentário:

A instrução acima **se (E_Impar(n))** irá apresentar o seguinte comportamento:

- se n for ímpar a expressão será avaliada como **se (1)** ou
- se n for par, a expressão será avaliada como **se (0)**

Como a expressão **se (0)** é tem sempre o valor lógico falso, a execução é desviada para a parte **senao** da instrução **se**.

A função **e_impar(x)** também pode ser chamada como **Eimpar(x)** sem o *"underline"*. Assim, o código poderia ser escrito como: **se (Eimpar(x))**...

Analisando detalhadamente

Na verdade, para aqueles que estão aprendendo lógica de programação, talvez seja interessante saber como é que a função **e_impar()** funciona. Digamos que o *interpretador Hall* não tivesse a função **e_impar** e também não tivesse a função **e_par**. Como se resolveria o problema de saber se um número é par ou ímpar ?

Vejamos...

Bom, sabemos da matemática que, por definição, um número é par quando é divisível por dois, isto é sua divisão por dois é exata, não sobra resto. Qualquer número que for dividido por dois e sobrar resto diferente de zero implicará em que esse número é ímpar.

O i*nterpretador Hall* disponibiliza a função **Resto()**. A sintaxe da função é **Resto(x,y)** e é interpretada como: A função retorna o resto da divisão de **x** por **y**. Assim, em nosso exemplo, podemos usar a chamada **Resto(x,2)** em que **x** é o valor informado pelo usuário. Se o valor retornado for igual a zero então o número **x** é par, caso contrário, se o valor retornado for diferente de zero, o número **x** será ímpar.

Vejamos o código...

Exemplo da função Resto()

```
// descricao: verifica se um numero e impar


algoritmo()
{
        inteiro n;

        leia("informe o valor de N: ",n);

        se (Resto(n,2) <> 0)
        {
                escreva("o numero informado eh impar.");
        }
        senao
        {
                escreva("o numero informado nao eh impar.");
        }
}
```

Comentário:

Observe a chamada à função **Resto** em que o valor de retorno da função é testado com o valor zero na expressão **se (Resto(n,2) <> 0)**. O sinal **<>** é o operador de operação relacional **diferente de**.

Melhorando um pouco mais...

OK, a função **Resto()** realmente faz um bom trabalho e resolve nosso problema de descobrir se um dado número é par ou ímpar. Bom, vamos dispensar a função **Resto** e tentar resolver o problema com menos recursos. Isto é, sem usar as funções pré-definidas do interpretador.

Bom, usando a definição de números pares, sabemos que: “um número é par quando é divisível por dois, isto é sua divisão por dois é exata, não sobra resto.”

Vamos raciocinar da seguinte maneira... Pegamos um número par qualquer, por exemplo 4. Dividimos esse número por 2 e atribuimos o quociente a uma variável x. Nesse caso, a variável **x** conterá o valor 2, o quociente da divisão de 4 por 2. Agora, se multiplicarmos esse quociente **x** por 2, obteremos novamente o valor original 4.

Vamos fazer a mesma coisa, só que agora vamos considerar um número ímpar, digamos 5. Dividimos esse número por 2 e atribuimos o quociente a uma variável **x**. Nesse caso, a variável **x** conterá o valor 2 ou 2.5 dependendo do seu **tipo**. Se a variável **x** for do tipo **inteiro** o valor armazenado será 2 e a parte decimal será truncada. Se a variável **x** for do tipo **real**, o valor armazenado será 2.5. Essa consideração não foi realizada no caso do número par pois o quociente será sempre o mesmo uma vez que a divisão por 2 de números pares é sempre exata, assim, o tipo da variável não tinha importância. Diferentemente, agora, na análise de um número ímpar.

Observemos que, se a variável **x** que for armazenar o quociente da divisão do valor dado por 2 for do tipo **inteiro**, irá resolver nosso problema. Considerando o valor 5 anterior, ao efetuarmos a divisão, teremos como quociente o valor 2 (a parte decimal 0.5 foi perdida pois a variável é do tipo inteiro). Assim, ao tentarmos reconstituir o valor original pela multiplicação do quociente **x** por 2 notaremos que o valor agora calculado será diferente do valor original. Assim, temos um outro método de resolver o problema de se decidir se um número é par ou ímpar. Veja o código abaixo:

Par ou ímpar com inteiros

```
algoritmo()
{
        inteiro n;    // o numero informado
        inteiro x;    // quociente de n por 2

        leia("informe o valor de N: ",n);

        x := n / 2;

        se (2*x == n)
        {
                escreva("o numero informado eh par.");
        }
        senao
        {
                escreva("o numero informado eh impar.");
        }
}
```

Comentário:

Notamos que, o *interpretador Hall*, permite que resolvamos de forma fácil, a questão de saber se um número é par ou ímpar e de várias formas diferentes.

Tornando-se profissional

Agora que você conhece os bastidores do *show* vamos desenvolver uma outra solução. Você irá criar a sua própria função para resolver o problema. A única restrição é que sua função não poderá ter o mesmo nome de uma função interna do interpretador, caso contrário, um erro será gerado. As funções internas tem precedência sobre as funções de usuário (as suas funções).

Vamos denominar a sua função de Impar.

Vejamos o código...

Sua própria função Impar()

```
04 // verifica se um numero e par ou impar
05
06 algoritmo()
07 {
08     // declaracao de variaveis
09
10     inteiro n;
11
12     leia("informe o valor de N: ",n);
13
14     se (Impar(n))
15     {
16         escreva("o numero informado é impar.");
17     }
18     senao
19     {
20         escreva("o numero informado é par.");
21     }
22 }
23 //-------------------------------------------------------------
24 funcao Impar(inteiro x)
25 {
26     se (Resto(x,2)<>0)
27     {
28         retorne 1;
29     }
30     senao
31     {
32         retorne 0;
33     }
34 }
```

Comentários:

No programa acima há duas funções: **algoritmo** e **Impar**.

Cada uma delas é identificada pelo abre e fecha-chaves que delimita o corpo da função. As chaves encapsulam o código da função e uma não interefe na outra. O código da função Impar() está entre as linhas 24 a 34 e o código da função algoritmo está entre as linhas 06 a 22. A linha 23 é um apenas comentário e não tem influência na execução do programa. Observe que o código de sua função **Impar()** implementa exatamente a funcionalidade já estudada anteriormente. A função **Impar()** espera receber um argumento do tipo **inteiro** e este fato é especificado no cabeçalho da função na expressão:

funcao Impar(inteiro x)

Dentro do corpo da função usamos a chamada à função **Resto()** já estudada anteriormente. Desse modo, você criou sua própria função para resolver o problema e o *interpretador Hall* reconhece isso e age como se fosse uma função interna. Fantástico !

Pois bem, um pequeno exemplo e um grande aprendizado.

Por enquanto é só pessoal,

Até a próxima.

fernandopaim@paim.pro.br