Paradigmas: Para que servem!? - Parte IV (Final)

Mário Leite

...

Nas partes anteriores foram apresentados sete tipos de paradigmas nos quais são classificadas as linguagens de programação com alguns exemplos mais expressivos de cada um desses tipos. Por outro lado, numa análise mais detalhada desses paradigmas, podem ser consideradas algumas perguntas que todo programador deveria fazer antes de se aventurar por uma linguagem, baseando-se apenas no seu paradigma ou na "modernidade" que tal linguagem acrescentaria ao seu currículo, ou mesmo nas conversas junto aos seus colegas. A pergunta básica é a seguinte: "*codificar numa determinada linguagem significa, automaticamente, que esteja seguindo determinado paradigma!!??*" Bem; como foi visto nas três partes anteriormente apresentadas, tem linguem que trata a codificação baseada em mais de um paradigma; portanto, a resposta é: NÃO!

O programa **"CriaChavePublica"**, mostrado abaixo, é parte de um sistema maior (deste autor) para encriptação/decriptação de mensagens pelo Método RSA. Neste trecho do sistema o objetivo é criar os elementos da chave pública **(c,n)**. Este programa é apresentado em pseudocódigo para que o leitor possa codificá-lo na linguagem de sua preferência e/ou naquela que ele domina. Note que a solução foi modularizada com três funções, além da rotina principal que valida os dados de entrada que são os dois números primos sobre os quais é aplicado o tratamento adequado para produzir a chave pública. Então, uma pergunta que poderia ser feita, neste caso seria: "*Em qual paradigma foi criado o pseudocódigo do programa"?* Resposta: **Em Nenhum**, especificamente, e **em Muitos** no geral. Isto porque tem elementos "imperativos", "funcionais" e "estruturais". E na codificação, caso seja implementada com interface gráfica, pode-se acrescentar "Orientado a Objetos" e "Orientado a Eventos". Em princípio não foi pensado na codificação, pois o importante era "COMO FAZER", que é a solução algoritmizada do programa; e isto serve em QUALQUER PARADIGMA seguindo a velha e boa "Máquina de von Neumann": sem nenhuma culpa. Então, caberia aqui outra pergunta bem objetiva: "*Se este programa fosse implementado na linguagem Python, por exemplo, o programador estaria seguindo qual Paradigma!?*" De acordo com algumas literaturas, Python é "Funcional" (por dar ênfase em funções matemáticas embutidas), "Estrutural", "Imperativa", "Orientada a Objetos", e pode até trabalhar com interfaces gráficas! Então: QUAL PARADIGMA!!??

Assim, respondendo à pergunta-título deste artigo: "**Para que servem os Paradigmas!?**", eu diria que, *do ponto de vista teórico servem para definir o modo como determinado tipo de programação é utilizado na sua implementação;* mas, *do ponto de vista prático não servem para se obter uma solução do problema proposto, pois cada linguagem define sua própria maneira de implementar a solução, introduzindo nuances particulares*. Portanto, rotular uma linguagem baseando-se em algum paradigma, eu, particularmente, acho discutível; e se isto é usado para justificar a qualidade do programa produzido, definitivamente não faz o menor sentido!

O programa completo sobre "Criptografia" e outros assuntos de Programação podem vistos nos *pdfs* dos meus livros. Para adquirir, entre em contado:

marleite@gmail com

```
//Programa "CriaChavePublica"
//Programa para criar a chave pública de encriptação de mensagens pelo Método RSA.
//Em Pseudocódigo
//Autor: Mário Leire
//--------------------------------------------------------------------------------
Função FunChavePub(n:inteiro):inteiro
//Gera e retorna o elemento c da chave pública (c,n).
   Declare j, EleChPub: inteiro
      Primo: lógico
Início
   {Cria um vetor de coprimos com n, baseando na "Função Fi"}
   Para j De 1 Até (n-1) Faça
      Se(Cont>=50) Então  //limita o número de coprimos em 50 (suficiente)
         Abandone  //abandona o loop incondicionalmente
      Senão
         MDC ← FunCalcMDC(n,j)  //chama função para calcular MDC(n,j)
         Se(MDC=1) Então  //(n,j) são primos entre si?
            Cont ← Cont + 1  //conta os coprimos
            VetCoPrimos[Cont] ← j
         FimSe
      FimSe
   FimPara
   {Verifica os coprimos e escolhe um primo com Cont}
   EleChPub ← 2
   Para j De 1 Até Cont Faça
       Primo ← FunVerifPrimo(VetCoPrimos[j])
       Se(Primo) Então
          {Escolhe VetCoPrimos[j] primo com Fi(n) c>=3
          Se(FunCalcMDC(VetCoPrimos[j],Cont)=1) Então
             Se((VetCoPrimos[j]>EleChPub) e (VetCoPrimos[j]>=3)) Então
                EleChPub ← VetCoPrimos[j]
                Abandone
             FimSe
          FimSe
       FimSe
    FimPara
    Retorne EleChPub  //retorno do elemento c da chave pública (c,n)
 FimFunção  //fim da função "FunChavePub"

//--------------------------------------------------------------------------------
 Função FunCalcMDC(N1,N2:inteiro): inteiro
 //Calcula e retorna o MDC de N1 e N2 utilizando o "Algoritmo de Euclides".
   Declare Aux: inteiro
Início
   Enquanto (N2<>0) Faça
      Aux ← N1
      N1  ← N2
      N2  ← (Aux Resto N2)
   FimEnquanto
   MDC ← N1
   Retorne MDC   //retorno do MDC(N1,N2)
FimFunção  //fim da função "FunCalcMDC"
```

```
//-------------------------------------------------------------------------------
Função FunVerifPrimo(N:inteiro): lógico
//Verifica se o número é primo.
   Declare j, NumDiv: inteiro
Início
   NumDiv ← 0
   Para j De 1 Até N Faça
      Se(N Resto j = 0) Então
         NumDiv ← NumDiv + 1
      FimSe
      Se(NumDiv>2) Então
         Abandone  //sai do loop (número certamente NÃO É PRIMO)
      FimSe
   FimPara
   Se(NumDiv=2) Então
      Retorne Verdadeiro
   Senão
       Retorne Falso
   FimSe
FimFunção  //fim da função "FunVerifPrimo"

//=============================================================================
//Programa principal
   //Variáveis globais
   Declare VetCoPrimos: array[1..200] de inteiro
       c, n, p, q, cont, MDC: inteiro
       cond, primos, primoP, primoQ: lógico
Início
   p ← 1
   q ← 1
   primos ← Falso
   cond ← ((p=q) ou (p<2) ou (q<2))
   {Verifica se os dois números são primos válidos}
   Enquanto ((cond) ou (Primos=Falso)) Faça
      EscrevaLn("") //salta linha
      Escreva("Digite o primeiro número primo: ")
      Leia(p)
      Escreva("Digite o segundo número primo: ")
      Leia(q)
      cond ← ((p=q) ou (p<2) ou (q<2))
      primoP ← FunVerifPrimo(p) //verifica se p é primo
      primoQ ← FunVerifPrimo(q) //verifica se q é primo
      Se((primoP) e (primoQ)) Então
         primos ← Verdadeiro  //ambos são primos
      Senão  //algum deles, ou ambos, não são primos
         EscrevaLn("Os dois números NÃO são, ambos, primos. Tente de novo!")
         EscrevaLn("")
      FimSe
   FimEnquanto  //fim do loop dos dados de entrada
   EscrevaLn("")
   {Define os elementos fundamentais da Chave Pública}
   n ← p*q
   c ← FunChavePub(n) //chama função para criar a chave pública
   Escreval("Chave pública (",c,",",n,")")
FimPrograma  //fim do programa "CriaChavePublica"
//=============================================================================
```