CONSTRUÇÃO DE SOFTWARE: PARTE

Um software não é resultado de um processo espontâneo, mas decorrente de um esforço significativo e articulado, despendido por profissionais com habilidades, dentre as quais aquelas pertinentes à Construção de Software, que inclui codificação, verificação, testes de unidade, testes de integração e depuração (*debugging*).

Uma parte significativa e relevante do esforço de desenvolvimento de um software dedica-se à elaboração de modelos, enquanto a Construção de Software, por outro lado, materializa software a partir de tais modelos. Noutras palavras, são as atividades de construção que produzem um artefato concreto, apto para execução por uma máquina.

Modelos de software são produzidos pelo projeto (*software design*) que, por sua vez, é precedido pela identificação de requisitos de software. Ou seja, primeiro os requisitos são identificados, depois é produzido o projeto de software correspondente a tais requisitos e, só então, é possível a construção. Embora a formalidade e o rigor destas atividades possam variar de forma significativa de um cenário para outro, não existe alternativa para essa sequência: requisitos, projeto, construção. As atividades de desenvolvimento de software estão organizadas em processos, que as ordenam, além de estabelecer artefatos que devem ser produzidos e aqueles a serem consumidos. Podemos continuar fornecendo detalhes até que toda a Engenharia de Software esteja envolvida, mas este texto não tem essa intenção. Aqui, deve ficar claro que a Construção de Software é parte e não se confunde com tudo o que é necessário para a produção de software.

AQUECIMENTO DE CONSTRUÇÃO DE SOFTWARE

Neste documento estão contidos 21 problemas, eles são “pequenos” e já estão resolvidos. Nossa missão é implementá-los.

É inviável lidar com problemas “reais” cujas soluções possam ser apoiadas pelo uso de computadores sem o uso considerável de recursos, principalmente de tempo. Dado que nosso foco de interesse é construção de software, e não há dispomos de “muito” tempo, a estratégia aqui é fazer uso de problemas cuja compreensão seja realizada de forma rápida e já venham acompanhados dos requisitos de software e dos projetos correspondentes.

No restante do documento cada um dos 21 problemas é descrito por um algoritmo. Dado que são algoritmos “simples”, o esforço do desenvolvimento do software correspondente fica praticamente reduzido ao esforço de construção do software para cada algoritmo. Um problema “real” exigiria esforço considerável de projeto (*design*), que só poderia ser realizado após um esforço considerável para caracterizar o software (requisitos) que, por sua vez, dependeria do esforço para compreensão do problema em questão.

Essa abordagem precisa ser utilizada de forma consciente e, dessa forma, evitar uma compreensão simplificada e artificial de desenvolvimento de software. Ou seja, estaremos concentrados nas questões de Construção de Software, mas cientes de que se trata de parte e não se confunde com tudo o que é preciso para desenvolver software.

Dito isso, nossa missão é implementar o código correspondente aos 21 problemas.

PROPRIEDADE 3025

Alguns números apresentam propriedades curiosas. O número 3025, por exemplo, é tal que $30 + 25 = 55$ e, em particular, $55^2 = 3025$. O número 0001 também satisfaz essa propriedade pois $0 + 1 = 1$ e $1^2 = 1$. O algoritmo abaixo retorna verdadeiro se o número fornecido como argumento satisfaz essa propriedade, ou falso, caso contrário.

```
Propriedade3025(n)                                          Algoritmo 1
Exige: 0 ≤ n ≤ 9999
i ← n div 100                                                       1
j ← n mod 100                                                       2
retorne (i + j)² = n                                                3
```

PROPRIEDADE 153

A soma dos cubos dos dígitos do número 153 resulta em 153, $1^3 + 5^3 + 3^3 = 153$. Um algoritmo que retorna verdadeiro se e somente se esta propriedade ocorre para um dado valor é fornecido abaixo.

```
Propriedade153(cdu)                                         Algoritmo 2
Exige: 0 ≤ cdu ≤ 9999
c ← n div 100                                                       1
du ← n mod 100                                                      1
d ← du div 10                                                       2
u ← du mod 10                                                       3
retorne (c³ + d³ + u³) = n                                          4
```

DIA DA SEMANA

Um algoritmo que define o dia da semana para uma data é apresentado por Kim Larsen no artigo *Computing the Day of the Week*, Dr. Dobb´s Journal, em abril de 1995. Este algoritmo é exibido abaixo e faz uso de três parâmetros, d para o dia, m para o mês e a para o ano da data em questão. Por exemplo, para primeiro dia de 2000, os argumentos fornecidos devem ser, respectivamente, 1, 1 e 2000. Observe que o algoritmo estabelece algumas restrições sobre os possíveis argumentos fornecidos. O dia, primeiro dos argumentos, não pode ser menor que 1 nem tampouco maior que 31, por exemplo.

```
DiaDaSemana(d, m, a)                                        Algoritmo 3
Exige: 1 ≤ d ≤ 31, 1 ≤ m ≤ 12, a > 1753
se m = 1 ou m = 2 então                                             1
  m ← m + 12                                                        2
  a ← a − 1                                                         3
fim se                                                              4
s ← d + 2m + 3(m+1) div 5 + a + a div 4 − a div 100 + a div 400     5
retorne s mod 7                                                     6
```

Na primeira linha há uma condição. Caso o argumento fornecido para o mês seja janeiro ou fevereiro, então o valor de m deve ser acrescido de 12 unidades e o ano subtraído de uma. Observe que se o argumento fornecido para o mês for diferente de 1 e também diferente de 2, então nada específico é realizado e a sentença na linha 5 é a próxima a ser executada. A última linha indica o valor produzido pelo algoritmo como sendo o valor do resto da divisão inteira de s por 7. Observe que o valor de s é definido na linha 5. O valor retornado é 0 para segunda-feira, 1 para terça-feira e assim sucessivamente.

RESTO DA DIVISÃO INTEIRA

O resto da divisão inteira de 5 por 3 é 2. O resto da divisão inteira de 3 por 8 é 3. Abaixo segue um algoritmo que executa esta operação.

Mod(x, y)	Algoritmo 4
Exige: 0 ≤ y, 0 < x	
s ← x	1
enquanto y ≤ s **faça**	2
s ← s - y	3
fim enquanto	4
retorne s	5

SOMA DOS PRIMEIROS NATURAIS

A soma dos primeiros naturais, $N = \{0, 1, 2, \ldots\}$, é dada por

$$\sum_{i=1}^{n} i = 1 + 2 + \cdots + n$$

O valor do somatório para os primeiros k naturais é definido por $k\,(k+1)/2$. Em vez de avaliar esta expressão que demanda o uso de um produto e uma divisão, pode-se optar pelo desenvolvimento do somatório, conforme a estratégia adotada pelo algoritmo abaixo.

SomaNaturais(n)	Algoritmo 5
Exige: 1 ≤ n	
i ← 2	1
s ← 1	2
enquanto i ≤ n **faça**	3
s ← s + i	4
i ← i + i	5
fim enquanto	6
retorne s	7

FATORIAL

O fatorial de um número natural n, $n \geq 1$, é definido abaixo.

$$n! = \prod_{i=1}^{n} i = 1 \times 2 \times 3 \cdots (n-1) \times n$$

```
Fatorial(n)                                          Algoritmo 6
Exige: 1 ≤ n
i ← 2                                                        1
f ← 1                                                        2
enquanto i ≤ n faça                                          3
   f ← f * i                                                 4
   i ← i + 1                                                 5
fim enquanto                                                 6
retorne f                                                    7
```

PRODUTO DE INTEIROS USANDO SOMAS

O produto de dois inteiros positivos é dado pela equação abaixo.

$$a * b = \sum_{i=1}^{a} b = \sum_{i=1}^{b} a$$

Um algoritmo que resulta no produto de dois inteiros usando apenas somas segue abaixo.

```
Produto(a, b)                                        Algoritmo 7
Exige: 0 ≤ a, 0 ≤ b
totalParcelas ← a                                            1
parcela ← b                                                  2
se b < a então                                               3
   totalParcelas ← b                                         4
   parcela ← a                                               5
fim se                                                       6
i ← 1                                                        7
s ← 0                                                        8
enquanto i ≤ totalParcelas faça                              9
   s ← s + parcela                                           10
   i ← i + 1                                                 11
fim enquanto                                                 12
retorne s                                                    13
```

POTÊNCIA USANDO SOMAS

O valor de x^y, por exemplo, $3^2 = 3 * 3$, é definido por

$$x^y = \prod_{i=1}^{y} x$$

O algoritmo abaixo implementa potência usando apenas somas, dado que reutiliza o algoritmo Produto, fornecido anteriormente.

Potencia(x, y)	Algoritmo 8
Exige: 0 ≤ x, 0 ≤ y	
potencia ← 1	1
i ← 1	2
enquanto i ≤ y **faça**	3
potencia ← Produto(potencia, x)	4
i ← i + 1	5
fim enquanto	6
retorne potencia	7

VALOR DE PI

O valor de π é definido conforme abaixo.

$$\pi = \sum_{i=0}^{\infty} (-1)^i \frac{4}{2i+1} = \frac{4}{1} - \frac{4}{3} + \frac{4}{5} - \frac{4}{7} + \cdots$$

Para fins práticos um limite deve ser estabelecido para a quantidade de termos do somatório a serem considerados. Para um total de n termos temos o algoritmo abaixo.

Pi(n)	Algoritmo 9
Exige: 1 ≤ n	
i ← 1	1
s ← -1	2
d ← -1	3
p ← 0	4
enquanto i ≤ n **faça**	5
d ← d + 2	6
s ← -1 * s	7
p ← p + 4 * s / d	8
i ← i + 1	9
fim enquanto	10
retorne p	11

Observe que $(-1)^i$ faz com que o sinal dos termos do somatório sejam alternados. Ainda digno de nota é a expressão $2i + 1$ que é geradora de números ímpares e, em vez de fazer uso de uma multiplicação e uma adição, pode-se usar apenas uma soma, conforme o algoritmo.

LOGARITMO NATURAL

A potência e^n onde e é o número de Euler e n um natural positivo é dado pelo somatório abaixo.

$$e^n = \sum_{i=0}^{\infty} \frac{n^i}{i!} = 1 + n + \frac{n^2}{2!} + \frac{n^3}{3!} + \cdots$$

Conforme a definição acima só é possível obter um valor aproximado, e cuja precisão depende da quantidade de termos empregados pelo somatório.

```
LogaritmoNatural(n, k)                                    Algoritmo 10
Exige: 1 ≤ n e 2 ≤ k
i ← 2                                                              1
e ← 1 + n                                                          2
numerador ← n
denominador ← 1
enquanto i ≤ k faça                                                3
    numerador ← numerador * numerador                              4
    denominador ← denominador * i                                  5
    e ← e + numerador / denominador                                6
    i ← i + 1                                                      7
fim enquanto                                                       8
retorne e                                                          9
```

RAZÃO ÁUREA

A razão áurea pode ser obtida através da aplicação de um processo iniciado com dois números inteiros positivos, por exemplo, 1 e 2. O número seguinte é obtido como a soma dos dois anteriores da sequência. Nesse exemplo, seria 3, o seguinte 5, depois 8, depois 13 e assim sucessivamente. Em algum momento esse processo é interrompido e a razão entre o último número da sequência e o anterior define a razão áurea. Um algoritmo correspondente é definido abaixo.

```
RazaoAurea(x, y, k)                                       Algoritmo 11
Exige: 0 ≤ x, x < y, 0 < k
c ← y                                                              1
a ← x                                                              2
i ← 1                                                              3
enquanto i ≤ k faça                                                4
    t ← c                                                          5
    c ← c + a                                                      6
    a ← t                                                          7
    i ← i + 1                                                      8
fim enquanto                                                       9
retorne c / a                                                     10
```

Observe que a razão áurea, empregando o algoritmo acima, depende do valor k, que deve ser compatível com a precisão desejada (quanto maior, mais precisão). Alternativamente a razão pode ser obtida pela expressão $(1 + \sqrt{5})/2$.

QUADRADO PERFEITO

Diz-se que um quadrado perfeito n^2 satisfaz a seguinte equação.

$$n^2 = \sum_{k=1}^{n} (2k - 1) = 1 + 3 + 5 + \cdots + (2n - 3) + (2n - 1)$$

O algoritmo abaixo retorna verdadeiro se o número fornecido é um quadrado perfeito.

```
QuadradoPerfeito(n)                                    Algoritmo 12
Exige: 1 ≤ n
i ← 1                                                          1
s ← 1                                                          2
enquanto s < n faça                                            3
    i ← i + 2                                                  4
    s ← s + i                                                  5
fim enquanto                                                   6
retorne s = n                                                  7
```

RAIZ QUADRADA

O algoritmo abaixo é obtido do Método Babilônico para extração da raiz quadrada. Quanto maior o valor de i mais preciso é o valor da raiz desejada.

```
Raiz(n, i)                                             Algoritmo 13
Exige: 0 < n
r ← 1                                                          1
enquanto 0 ≤ i faça                                            2
    r ← (r + n/r) / 2                                          3
    i ← i - 1                                                  4
fim enquanto                                                   5
retorne r                                                      6
```

NÚMERO PRIMO

Um número n maior que 1 é dito primo quando os únicos divisores desse número são 1 e o próprio n. Uma solução para verificar se n é primo é verificar se qualquer um dos números de 2 até $n - 1$ é divisor de n, conforme o algoritmo abaixo.

```
Primo(n)                                               Algoritmo 14
Exige: n > 1
i ← 2                                                          1
enquanto i < n faça                                            2
```

```
   se n mod i = 0 então                                    3
       retorne falso                                       4
   fim se                                                  5
   i ← i + 1                                               6
fim enquanto                                               7
retorne verdadeiro                                         8
```

Convém destacar que o algoritmo acima realiza várias operações desnecessárias. Por exemplo, não é necessário fazer a verificação a partir de $n/2$. Adicionalmente, se não é divisível por 2, também não será por 4, 6 e assim sucessivamente. O Crivo de Eratóstenes, apresentado na seção seguinte, oferece uma estratégia alternativa ao algoritmo acima.

CRIVO DE ERATÓSTENES

Eratóstenes de Cyrene (Líbia) é tido como o primeiro homem a medir a circunferência da terra. Além de astrônomo e poeta, é mais conhecido na matemática pelo método de determinação de números primos, conhecido por Crivo de Eratóstenes. Nesse método, os números primos até um dado valor são identificados. Por exemplo, para encontrar todos os primos de 1 até 100, os números múltiplos de 2, 3, 4 e assim por diante, são progressivamente eliminados dessa faixa de valores. Iniciamos por 2 e para cada múltiplo de 2 até 100, simplesmente retiramos esses do conjunto. Depois o 3 é considerado e, todo múltiplo de 3 maior que 3 é eliminado do conjunto. O processo se repete para 4 e assim por diante, desde que não tenha sido eliminado e até que o próximo valor a ser considerado não seja maior que $\sqrt{100} = 10$. Neste ponto o processo pode ser interrompido e todos os números não eliminados são primos.

O algoritmo abaixo emprega o método descrito acima. O vetor fornecido deve possuir, pelo menos, n posições e inicialmente todas com o valor 0, indicando que o número do índice em questão não foi eliminado. Ao final, todos aqueles índices que permanecerem com o valor 0 são números primos. Ou seja, para verificar se 100, por exemplo, é um número primo, então devemos criar um vetor com 100 posições, de índices 1, 2, até 100 e executar o algoritmo abaixo com este vetor e o valor 100 como argumentos. Ao final, teremos que consultar o valor na posição de índice 100. Neste caso sabemos que o valor deve ser 1, indicando que este número foi eliminado ao longo da execução do algoritmo e, portanto, não se trata de um primo.

```
CrivoEratostenes(a[], n)                                   Algoritmo 15
Exige: n > 1 e a[i] = 0 para todo 2 ≤ i ≤ n
i ← 2                                                      1
limite ← ⌊√n⌋                                              2
enquanto i ≤ limite faça                                   3
   se a[i] = 0 então                                       4
       multiplo ← i + i                                    5
       enquanto multiplo ≤ n faça                          6
           a[multiplomult] ← 1                             7
```

```
            multiplo ← multiplo + i                                    8
         fim enquanto                                                  9
   fim se                                                              10
   i ← i + 1                                                           11
fim enquanto                                                           12
```

MAIOR DIVISOR COMUM

O maior divisor comum de dois inteiros positivos e não nulos é o maior valor dentre os divisores comuns entre eles. Por exemplo, o maior divisor comum entre 10 e 5 é 5. Entre 30 e 28 é 2. Por volta do ano 200 antes de Cristo, Euclides apresentou um algoritmo para identificar o maior divisor comum entre dois inteiros baseado na seguinte definição.

$$MDC(a,b) = \begin{cases} a, & se\ b = 0 \\ MDC(b, a\ mod\ b), & \text{caso contrário} \end{cases}$$

Em conformidade com esta definição, $MDC(81,54) = MDC(54,27) = MDC(27,0) = 27$. Esta definição é empregada pelo algoritmo abaixo, sem o uso de recursão.

```
MDC(a, b)                                                    Algoritmo 16
Exige: b ≤ a, 0 < b
enquanto b ≠ 0 faça                                                    1
    m ← a mod b                                                        2
    a ← b                                                              3
    b ← m                                                              4
fim enquanto                                                           5
retorne a                                                              6
```

O algoritmo abaixo é uma variante daquele acima e se limita ao uso de subtrações (não faz uso do resto da divisão inteira).

```
MDC2(a[], n)                                                 Algoritmo 17
Exige: b ≤ a, 0 < b
enquanto a ≠ b faça                                                    1
    se a > b então                                                     2
        a ← a – b                                                      3
    senão                                                              4
        b ← b – a                                                      5
    fim se                                                             6
fim enquanto                                                           7
retorne a                                                              8
```

É atribuído a Euclides a coleção de 13 livros conhecida por *Elementos*, que só perde para a Bíblia em número de edições, o que o torna uma das publicações mais bem-sucedidas.

REGRA DE HORNER PARA AVALIAÇÃO DE POLINÔMIO

Observe o polinômio abaixo.

$$p(x) = \sum_{i=0}^{n} a_i x^i = a_o + a_1 x + a_2 x^2 + a_3 x^3 + \cdots + a_n x^n$$

A avaliação desse polinômio para um valor específico de x e onde $a_0, a_1, \ldots, a_n$ são números reais, pode obtida por meio de somas, produtos e potências. Em um algoritmo onde cada parcela do somatório é obtida e acumulada, teríamos algo como o cômputo de $n-1$ potências, n produtos e outras n somas. A Regra de Horner, contudo, é uma alternativa mais eficiente, conforme o algoritmo abaixo.

Horner(x, g, a_g, a_{g-1}, ..., a_0)	Algoritmo 18
Exige: 1 ≤ g	
p ← a_g	1
i ← g - 1	2
enquanto 0 ≤ i **faça**	3
p ← p * x + a_i	4
i ← i - 1	5
fim enquanto	6
retorne p	7

FIBONACCI

Por definição, o n-ésimo número de Fibonacci, $F(n)$, $n \geq 0$, é tal que

$$F(n) = \begin{cases} 0 & se\ n = 0 \\ 1 & se\ n = 1 \\ F(n-1) + F(n-2) & se\ n \geq 2 \end{cases}$$

O algoritmo abaixo é uma proposta de obtenção do n-ésimo número de Fibonacci (versão não recursiva).

Fibonacci(n)	Algoritmo 19
Exige: 0 ≤ n	
a ← 0	1
c ← 1	2
se n = 0 **ou** n = 1 **então**	3
retorne n	4
fim se	5
i ← 2	6
enquanto i ≤ n **faça**	7
t ← c	8
c ← c + a	9
a ← t	10
i ← i + 1	11
fim enquanto	12
retorne c	13

CADASTRO DE PESSOAS FÍSICAS (CPF)

O CPF é uma sequência de 11 dígitos, sendo dois deles denominados de dígitos verificadores. Abaixo segue um algoritmo que verifica os dígitos de um CPF. Se a regra de formação dos dígitos é atendida para um dado CPF de entrada, então o algoritmo retorna verdadeiro, caso contrário, retorna falso.

O formato de um CPF é definido por abc.def.ghi-jk, onde os dois pontos e o hífen são usados apenas por legibilidade. Os oito primeiros dígitos formam o número-base, enquanto o nono dígito, i, representa a região fiscal (local de emissão do CPF). O valor 1 para esse dígito é para emissões no DF, GO, MS, MT e TO. O valor 2 para AC, AM, AP, PA, RO e RR. O valor 3 para CE, MA e PI. O dígito 4 para AL, PB, PE e RN. O dígito 6 para MG, 7 para ES e RJ, 8 para SP, 9 para PR e SC e, por último, o dígito 0 para RS.

O algoritmo abaixo assume que o argumento (vetor) define os dígitos de um CPF, desde d[1] até d[11]. Ou seja, os dígitos verificadores são d[10] e d[11], conforme essa convenção.

CPF(d[]) — Algoritmo 20

```
Exige: 11 dígitos em d, de d[1] até d[11]
j ← d[1] + 2d[2] + 3d[3] + 4d[4] + 5d[5] + 6d[6] + 7d[7] + 8d[8] + 9d[9]          1
k ← d[2] + 2d[3] + 3d[4] + 4d[5] + 5d[6] + 6d[7] + 7d[8] + 8d[9] + 9d[10]        2
dj ← Mod(Mod(j,11), 10)                                                          3
dk ← Mod(Mod(k, 11), 10)                                                         4
retorne dj = d[10] e dk = d[11]                                                  5
```

Uma variante desse algoritmo observando uma estratégia similar àquela empregada para avaliação de um polinômio pelo Método de Horner segue abaixo.

CPF2(d[]) — Algoritmo 21

```
Exige: 11 dígitos em d
c ← 8                                  1
p ← d[9]                               2
s ← d[9]                               3
enquanto 1 ≤ c faça                    4
   p ← p + d[c]                        5
   s ← s + p                           6
   c ← c - 1                           7
fim enquanto                           8
j ← Mod(Mod(s, 11), 10)                9
k ← Mod(Mod(s – p + 9d[10], 11), 10)   10
retorne j = d[10] e k = d[11]          11
```