Chapter 1

Solved Examples

EXAMPLE 1.1: Reading pseudocodes

Examine the following pseudocode segments and determine the values stated by **Write** statements. Assume all variables are integers.

(a)

$kode \leftarrow 0; j \leftarrow 1$
For $[i = -2, 5]$
  $kode \leftarrow kode + 2 * j$
  $num \leftarrow i * kode$
  If $[kode > 6]$ **Then**
    $j \leftarrow -1$
  End If
  Write: num
End For
Write: $kode$

(b)

$kode \leftarrow 0$
For $[i = 0, 9, 3]$
  For $[j = i, 8, 3]$
    $kode \leftarrow kode + j - i$
    Write: $j, kode$
  End For
  Write: $i, kode$
End For
Write: $kode$

(c)

$kode \leftarrow 0$
For $[i = 9, 1, (-2)]$
  For $[j = 4, i, 2]$
    If$[j > 7]$ **Then Exit**
    $kode \leftarrow kode + j$
    Write: $i, j, kode$
  End For
  Write: $i, kode$
End For

(d)

$kode \leftarrow 0$
For $[i = 9, 1, (-2)]$
  For $[j = 4, i, 2]$
    $kode \leftarrow kode + j$
    Write: $i, j, kode$
  End For
  If$[i > 7]$ **Then Exit**
  Write: $i, kode$
End For

SOLUTION:

In (a), the values of 0 and 1 are allocated to the memory location of $kode$ and j, respectively. The variable i is initialized with the **For** loop as $i = -2$. The rhs of the next statement becomes $0+2*1 = 2$, which is assigned to $kode$ ($kode = 2$). The next statement yields $num = (-2) * 2 = -4$. Since $kode < 6$, the **If** construct will be skipped. The value of num in memory (i.e., -4) is written. The loop returns to the top of **For** statement and takes the value of $i = -1$. Then the following two statements

are executed as follows: $kode \leftarrow 2 + 2 * 1 = 4$ and $num \leftarrow (-1)(4) = -4$.

(a)	(b)		(c)			(d)		
-4	0	0	9	4	4	9	4	4
-4	3	3	9	6	10	9	6	10
0	6	9	9	10		9	8	18
8	0	9	7	4	14			
12	3	9	7	6	20			
12	6	12	7	20				
8	3	12	5	4	24			
0	6	12	5	24				
0	6	12	3	24				
	9	12	1	24				

In (b), $kode$ is initialized to 0. The **For** loop over i-variable runs for $i = 0$, 3, 6, and 9. Correspondingly, the **For** loop over j-variable runs for 0, 3, 6, 9 (in case of $i = 0$), 3, 6, 9 (in case of $i = 3$), 6, 9 (in case of $i = 6$), and 9 (in case of $i = 9$).

Discussion: Such exercises, as in this case, that require you to generate output from a code offer the following benefits:

- Reinforces Learning: By actively coding either actual or pseudocodes and generating output, you *do* reinforce your understanding of programming concepts and syntax.
- Debugging Skills: These exercises help you practice debugging. If the output is not what you expected, you will learn to identify and fix errors.
- Problem Solving: You develop critical thinking and problem-solving skills by figuring out how to achieve the desired output.
- Conceptual Understanding: Seeing the output helps solidify concepts like data types, control structures, and algorithms in a concrete way.
- Creativity and Exploration: Generating output can inspire creativity, encouraging you to experiment with different approaches and techniques.
- Confidence Building: Successful generation of outputs boosts confidence in your coding abilities and encourages further exploration.

Overall, this author recommends you tı generate output for the first few steps of the pseudocodes presented in this textbook to make your learning process more interactive and engaging, which will support your deeper understanding and skill development.

EXAMPLE 1.2: Reading pseudocodes

Examine the following pseudocode segments and determine the values stated by **Write** statements. Assume all variables are integers.

(a)

$t \leftarrow 0; n \leftarrow 0$
While $[t < 5]$
  $t \leftarrow t + n$
  $t \leftarrow 0.4 * t^2$
  $n \leftarrow n + 1$
  Write: n, t
End While

(b)

$c \leftarrow 4; k \leftarrow 1; t \leftarrow 0$
Repeat
  $k \leftarrow k * c$
  $c \leftarrow c - 1$
  $k \leftarrow k + 2 * c$
  $t \leftarrow t + k$
  Write: c, t
Until $[c < 0]$

SOLUTION:

In part (a), the variables t and n are initialized to zero in the first line. Since $t = 0$ is less than 5, the logical statement $t < 5$ becomes True and the corresponding block statements are executed. The first statement of the block evaluates $t \leftarrow 0 + 0 = 0$. Subsequently, we obtain $t \leftarrow 0.4 * 0^2 = 0$, $n \leftarrow 0 + 1 = 1$. Write statement prints the values of n (i.e., 1) and t (i.e., 0). The construction goes to the top of the loop and evaluates the logical expression, i.e., $t < 5$, which is True. Then, the code block is executed. Note that with each iteration the value of t increases and will eventually exceed 5. This process is iterated (repeated) as long as t is less than 5. The code output for the rest of the iterations is presented below.

In part (b), the variables t, k, and c are initialized to 0, 1, and 4, respectively, i.e., these are the values stored in the allocated memory space. The block statements are executed until the loop condition $c < 0$ is reached. The first statement evaluates $k \leftarrow 1 * 4 = 4$. The other statements, respectively, give $c \leftarrow 4 - 1 = 3$, $k \leftarrow 4 + 2 * 3 = 10$, and $t \leftarrow 0 + 10 = 10$. Now the loop condition ($c < 0$?) is evaluated. Since $c = 3 > 0$, the code statements will be executed again with $c = 3$, $k = 1$, and $t = 10$. If you perform the rest of the iterations, you should obtain the output values presented below.

CODE OUTPUTs

(a)

```
1   0.0
2   0.4
3   2.304
4   11.252966400000002
```

(b)

```
 3   10
 2   44
 1   114
 0   184
-1   182
```

Discussion: These exercises illustrate the **While** and **Repeat-Until** loops. Note that the **Repeat-Until** loop is not available in every programming language, but using programming language-specific statements, a construction can be established to carry out the same task.

EXAMPLE 1.3: Calculate average and standard error

Consider a DC electric motor to be used to lift a mass m to a height h. The efficiency (e) of the work done by motor to energy delivered to motor:

$$\text{Efficiency, } \eta = \frac{\text{work done by motor}}{\text{energy delivered to motor}} = \frac{mgh}{VI\tau}$$

where g is the acceleration of gravity, V is the applied voltage, I the current, and τ the time for which the motor runs. The relative uncertainties (standard deviations) of m, h, V, I, and τ are given as $\pm 2\%$, $\pm 1\%$, $\pm 1\%$, $\pm 2\%$, and $\pm 4\%$, respectively. Estimate the average and standard error in η.

SOLUTION:

For either case, we assume that g has negligible uncertainty. In other words, the acceleration of gravity can be excluded from the variable list of efficiency:

$$\eta(m, h, V, I, \tau) = \frac{mgh}{VI\tau}$$

(a) The error in some variables can be best described by the *average error*, typically defined as the mean of the absolute differences between predicted values and actual values. This provides a measure of how far predictions are from the true values without considering the direction of the errors. In the case, we will assume that the errors in measured quantities are *average errors* (dm, dh, dV, dI, $d\tau$ in $\pm\%$), which gives rise to $\pm d\eta$. The maximum possible error in each variable, in the most pessimistic scenario, will contribute to the maximum possible error in the efficiency.

The total differential formula for the efficiency is expressed as

$$d\eta = \left|\frac{\partial \eta}{\partial m} dm\right| + \left|\frac{\partial \eta}{\partial h} dh\right| + \left|\frac{\partial \eta}{\partial V} dV\right| + \left|\frac{\partial \eta}{\partial I} dI\right| + \left|\frac{\partial \eta}{\partial \tau} d\tau\right|$$

where dm, dh, dV, dI, and $d\tau$ are differentials that denote absolute errors. Notice that it is quite easy to calculate the partial derivatives in the above expression. Once the partial derivatives are substituted into the above expression and both sides are divided by η, we find

$$\frac{d\eta}{\eta} = \left|\frac{dm}{m}\right| + \left|\frac{dh}{h}\right| + \left|-\frac{dV}{V}\right| + \left|-\frac{dI}{I}\right| + \left|-\frac{d\tau}{\tau}\right|$$

or multiplying both sides by 100 leads to an expression in terms of percentage errors:

$$\eta\% = m\% + h\% + V\% + I\% + \tau\%$$

Finally, the relative error of the estimated efficiency is found as $\pm(2 + 1 + 1 + 2 + 4)\% = \pm 10\%$.

(b) In the case of dm, dh, dV, dI, and $d\tau$ being independent of each other, these differentials can be replaced with the standard deviations, i.e., σ_m, σ_h, σ_V, σ_I, and σ_τ. By dividing both sides with η, the quadratic error propagation formula yields

$$\sigma_\eta = \sqrt{\left(\frac{\partial \eta}{\partial m}\sigma_m\right)^2 + \left(\frac{\partial \eta}{\partial h}\sigma_h\right)^2 + \left(\frac{\partial \eta}{\partial V}\sigma_V\right)^2 + \left(\frac{\partial \eta}{\partial I}\sigma_I\right)^2 + \left(\frac{\partial \eta}{\partial \tau}\sigma_\tau\right)^2}$$

or

$$\sigma_\eta = \sqrt{\left(\frac{gh}{VI\tau}\sigma_m\right)^2 + \left(\frac{mg}{VI\tau}\sigma_h\right)^2 + \left(-\frac{mgh}{V^2 I\tau}\sigma_V\right)^2 + \left(-\frac{mgh}{VI^2\tau}\sigma_I\right)^2 + \left(-\frac{mgh}{VI\tau^2}\sigma_\tau\right)^2}$$

or

$$\frac{\sigma_\eta}{\eta} = \sqrt{\left(\frac{\sigma_m}{m}\right)^2 + \left(\frac{\sigma_h}{h}\right)^2 + \left(\frac{\sigma_V}{V}\right)^2 + \left(\frac{\sigma_I}{I}\right)^2 + \left(\frac{\sigma_\tau}{\tau}\right)^2}$$

or

$$\eta\% = \sqrt{(m\%)^2 + (h\%)^2 + (V\%)^2 + (I\%)^2 + (\tau\%)^2}$$

Substituting the numerical values into the above expression, we find

$$\eta\% = \sqrt{2^2 + 1^2 + 1^2 + 2^2 + 4^2} \cong 5.1\%$$

This case yields the relative error of $\approx \pm 5.1\%$, which is about half of the value calculated in Part (a).

Discussion: This example illustrates the application of *error propagation analysis*, which helps quantify uncertainty in measurements and/or calculations. The benefits of uncertainty analysis are

- Quantification of Uncertainty: Error propagation analysis provides a systematic way to calculate the effects of uncertainties in input on the uncertainty in the output result.
- Improved Accuracy: By understanding how errors from each variable propagate, you can identify critical variables that require further attention.
- Risk Assessment: It helps assess the reliability of results, which is vital in decision-making processes, especially in fields like engineering and finance.
- Guidance for Experimental Design: Knowing how errors propagate can inform the design of experiments to minimize uncertainty in key measurements.

Ideally, every experiment should accompany a proper uncertainty analysis.

EXAMPLE 1.4: Converting decimal numbers

Convert 683 and 8.5 (in the decimal system) to equivalent binary, octal, and hexadecimal numbers.

SOLUTION:

Conversion of 587 (an integer) leads to

$$\begin{aligned}(683)_{10} &= 1\times 2^9 + 0\times 2^7 + 0\times 2^5 + 1\times 2^3 + 0\times 2^1 + 1\times 2^0 \\ &= (1010101011)_2 \\ (683)_{10} &= 1\times 8^3 + 2\times 8^2 + 5\times 8^1 + 3\times 8^0 = (1253)_8 \\ (683)_{10} &= 2\times 16^2 + \mathrm{A}\times 16^1 + \mathrm{B}\times 16^0 = (2\mathrm{AB})_{16}\end{aligned}$$

where A and B denote 10 and 11, respectively.

Conversion of 18.75 (a real number) gives

$$\begin{aligned}(8.5)_{10} &= 1\times 2^3 + 0\times 2^2 + 0\times 2^1 + 0\times 2^0 + 1\times 2^{-1} = (1000.1)_2 \\ (8.5)_{10} &= 1\times 8^1 + 0\times 8^0 + 4\times 8^{-1} = (10.4)_8 \\ (8.5)_{10} &= 8\times 16^0 + 8\times 16^{-1} = (8.8)_{16}\end{aligned}$$

Discussion: Such numeral systems provide practical benefits in computation, representation, and communication, making them essential tools in technology and programming. Note that certain calculations and algorithms are more naturally expressed in one numeral system than another, making conversions necessary.

Most computers operate using binary systems (0s and 1s) since digital circuits use two states (on and off). The binary systems simplify the design of digital circuits, making it easier to create logic gates and perform computations. The decimals in this example were represented "exactly" in the binary, octal, and hexadecimal number systems. Most numbers are handled by their floating point values, which may lead to loss of precision in computers.

EXAMPLE 1.5: Approximating infinite series

Using the following series, determine the minimum number terms in the approximating series to compute ln(2) correct up to *five decimal places*.

$$\ln x = 2\left\{\left(\frac{x-1}{x+1}\right) + \frac{1}{3}\left(\frac{x-1}{x+1}\right)^3 + \cdots + \frac{1}{2n-1}\left(\frac{x-1}{x+1}\right)^{2n-1} + \cdots\right\}$$
$$= 2\sum_{n=1}^{\infty}\frac{1}{2n-1}\left(\frac{x-1}{x+1}\right)^{2n-1}$$

SOLUTION:

The partial sum (the summation of the first N terms) and the truncation error can be expressed as follows:

$$\ln x \cong S_N(x) = 2\left\{\left(\frac{x-1}{x+1}\right) + \frac{1}{3}\left(\frac{x-1}{x+1}\right)^3 + \cdots + \frac{1}{2N-1}\left(\frac{x-1}{x+1}\right)^{2N-1}\right\}$$

$$R_N = \frac{2}{2N+1}\left(\frac{x-1}{x+1}\right)^{2N+1}$$

For $x = 2$, the N-term partial sum (i.e., approximation) and the truncation error become

$$\ln 2 \cong S_N(2) = 2\left\{\frac{1}{3} + \frac{1}{3\cdot 3^3} + \frac{1}{5\cdot 3^5} + \cdots + \frac{1}{(2N-1)\cdot 3^{2N-1}}\right\}, \quad R_N = \frac{2}{2N+1}\frac{1}{3^{2N+1}}$$

In order for the computed approximation to be correct to at least five decimal places, the truncation error must be less than 0.5×10^{-5}. For various N values, the approximation and corresponding truncation error are tabulated in the table below. It is clear that for five terms we have $R_5 = 1.026 \times 10^{-6} < 0.5 \times 10^{-5}$. That is, the computed sum (=0.693146) is correct to at least five-significant figure.

N	S_N	R_N	True Error
1	0.6666667	0.0246914	0.0264805
2	0.6913580	0.0016461	0.0017892
3	0.6930040	0.0001306	0.0001431
4	0.6931348	0.0000113	0.0000124
5	0.6931460	1.026×10^{-6}	1.13×10^{-6}
6	0.6931471	9.649×10^{-8}	1.07×10^{-8}
7	0.6931472	9.292×10^{-9}	1.03×10^{-9}

Note that when comparing this value with six- and seven-term approximations, we verify that the computed value using the five-term approximation is indeed correct to a five-significant figure.

Discussion: An approximation to an infinite series, $\sum_{n=1}^{\infty} a_n$, can be obtained by truncating the series to a finite sum, i.e., evaluating a *partial sum*:

$$S_N = a_1 + a_2 + \ldots + a_N$$

This is the simplest method and is effective if the series converges very quickly. The more terms we include, the closer we get to the true value sum. But figuring out the number of terms required

to add up to obtain the partial sum with the desired accuracy can be tedious. On the other hand, in programming and computational contexts, the partial sum and its truncation error can be calculated simultaneously using a **While**- or **Repeat-Until**-conditional loop structure. Then, the partial sum can be terminated when the truncation becomes $|R_N| < 0.5 \times 10^{-b}$, where b is the number of accurate digits required.