Errata for
C# and Algorithmic Thinking for the Complete Beginner
Second Edition

5.2 What is a Constant?

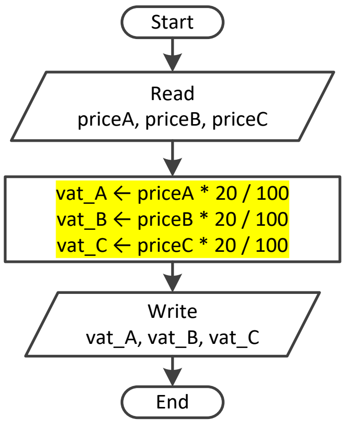

Start
Read
priceA, priceB, priceC
vat_A ← priceA * 20 / 100
vat_B ← priceB * 20 / 100
vat_C ← priceC * 20 / 100
Write
vat_A, vat_B, vat_C
End

Figure 5–1 Calculating the 20% VAT for three products without the use of a constant

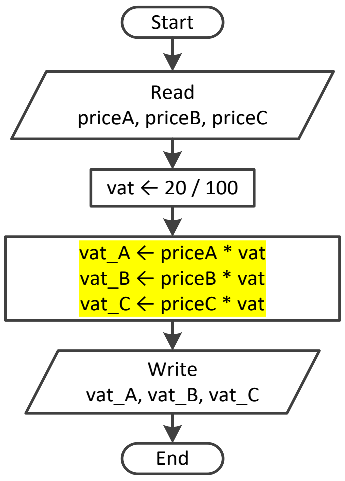


Figure 5–2 Calculating the 20% VAT for three products using a variable, `vat`

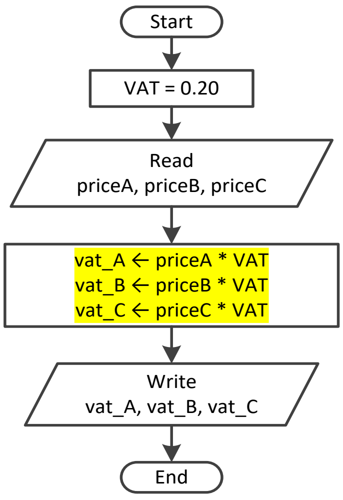


Figure 5-3 Calculating the 20% VAT for three products using a constant, `VAT`

5.10 Review Exercises

3. Complete the following table

Value	Data Type	Declaration and Initialization
The name of my friend	String	`string name = "Mark";`
My address		`string address = "254 Lookout Rd. Wilson, NY 27893";`
The average daily temperature		
A telephone number		`string phone_number = "1-891-764-2410";`
My Social Security Number (SSN)		
The speed of a car		
The number of children in a family		

Exercice 7.4-1 Which C# Statements are Syntactically Correct?

Which of the following C# assignment statements are syntactically correct?

i.	`a = -10;`	*v.*	`a = COWS;`	*ix.*	`a = true;`
ii.	`10 = b;`	*vi.*	`a + b = 40;`	*x.*	`a /= 2;`
iii.	`a_b = a_b + 1;`	*vii.*	`a = 3 b;`	*xi.*	`a += 1;`
iv.	`a = "COWS";`	*viii.*	`a = "true";`	*xii.*	`a =* 2;`

Solution

i. **Correct.** It assigns the integer value −10 to variable `a`.
ii. **Wrong.** On the left side of the value assignment operator, only variables can exist.
iii. **Correct.** It increases variable `a_b` by one.
iv. **Correct.** It assigns the string (the text) *COWS* to variable `a`.
v. **Correct.** It assigns the content of constant (or even variable) `COWS` to variable `a`.
vi. **Wrong.** On the left side of the value assignment operator, only variables (not expressions) can exist.
vii. **Wrong.** It should have been written as `a = 3 * b`.
viii. **Correct.** It assigns the string `true` to variable `a`.
ix. **Correct.** It assigns the value `true` to variable `a`.
x. **Correct.** This is equivalent to `a = a / 2`.
xi. **Correct.** This is equivalent to `a = a + 1`.
xii. **Wrong.** It should have been be written as `a *= 2` (which is equivalent to `a = a * 2`).

7.5 Incrementing/Decrementing Operators

The double slashes (//) ~~after the `Console.WriteLine()` statement~~ indicate that the text that follows is a comment; thus, it is never executed.

Exercise 14.3-5 Finding the Sum of Digits

Write a C# program that prompts the user to enter a three-digit integer and then calculates the sum of its digits. Solve this exercise without using the integer remainder (%) operator.

14.6 Review Exercises

3. Write a C# program that prompts the user to enter his or her name and then creates a secret password consisting of three letters (in lowercase) randomly picked up from his or her name, and a random four-digit number. For example, if the user enters “Vassilis Bouras” a secret password can probably be one of “sar1359” or “vbs7281” or “bor1459”. Space characters are not allowed in the secret password.
4. Write a C# program that prompts the user to enter a three-digit integer and then reverses it. For example, if the user enters the number 375, the number 573 must be displayed. Solve this exercise without using the integer remainder (%) operator.

15.8 How to Negate Boolean Expressions

However, there is a small detail that you should be careful with. If both AND (`&&`) and OR (`||`) operators co-exist in a complex Boolean expression, then the expressions that use the OR (`||`) operators in the negated Boolean expression must be enclosed in parentheses, in order to preserve the initial order of precedence. For example, if the original Boolean expression is

```
x >= 5 && x <= 10 || y == 3
```

20.1 What are Nested Decision Control Structures?

Complex code may lead to invalid results! Try to keep your code as simple as possible by breaking large nested decision control structures into multiple smaller ones, or by using other types of decision control structures.

Obviously, you can nest **any** decision control structure inside **any other** decision control structure as long as you keep them syntactically and logically correct. In the next example, a case decision structure is nested within a dual-alternative decision structure.

24.3 Review Questions: True/False

4. The following C# program

```
static void Main(string[] args) {
  int a, total;
  a = 5;
  total = total + a;
  Console.WriteLine(total);
}
```

satisfies the property of effectiveness.

25.4 Review Questions: True/False

14. The following C# program

```
static void Main(string[] args) {
  int i;

  do {
    Console.WriteLine("Hello");
    i++;
  } while (i <= 10);
}
```

satisfies the property of effectiveness.

25.6 Review Exercises

11. Fill in the gaps in the following code fragments so that all loops perform exactly six iterations.

i.
```
int a = 5;
do {
  a--;
} while (a > …… );
```

ii.
```
int a = 12;
do {
  a++;
} while (a < …… );
```

iii.
```
double a = 20;
do {
  a = a + …… ;
} while (a != 23);
```

iv.
```
int a = 100;
do {
  a -= 20;
} while (a != …… );
```

v.
```
int a = 2;
do {
  a = 2 * a;
} while (a != …… );
```

vi.
```
double a = 10;
do {
  a = a + 0.25;
} while (a <= …… );
```

27.5 Review Exercises

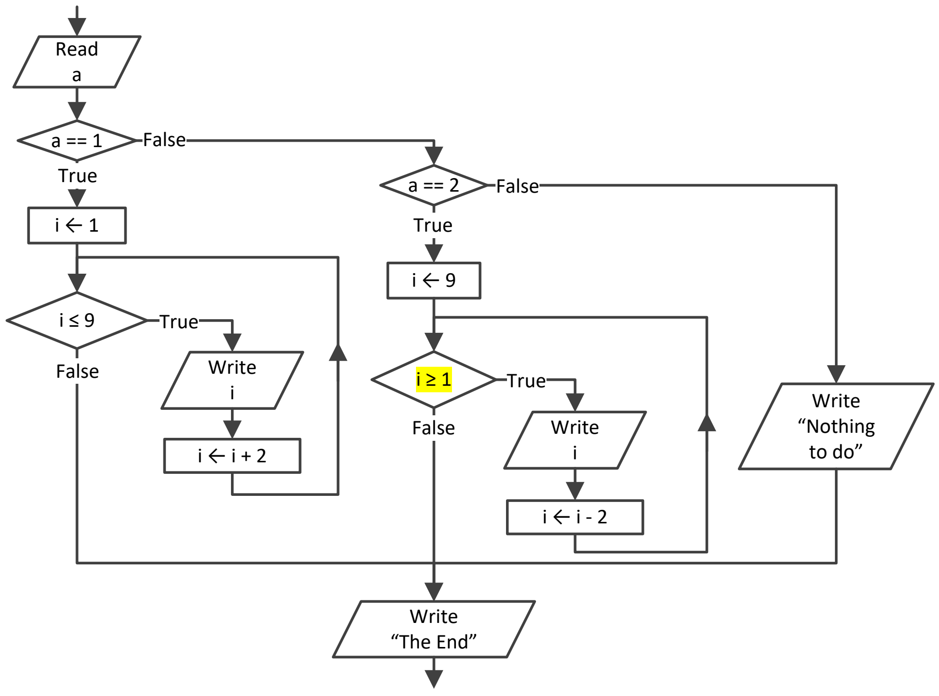
iii.

```
int a;
float b;
for (a = …… ; a >= -15; a -= 2) {
  for (b = 10; b >= 0.5 ; b -= 0.5) {
    Console.WriteLine("Hello Hephaestus");
  }
}
```

28.3 The "Ultimate" Rule

```
int positives_given;
double x;

positives_given = 0;            //Initialization of positives_given
while (positives_given != 3) {  //This is dependent on positives_given
  A statement or block of statements
  if (x > 0) {
    positives_given += 1;       //Update/alteration of positives_given
  }
}
```

Exercise 29.2-4 Designing the Flowchart Fragment



Exercise 30.6-2 Rice on a Chessboard

project_30_6_2

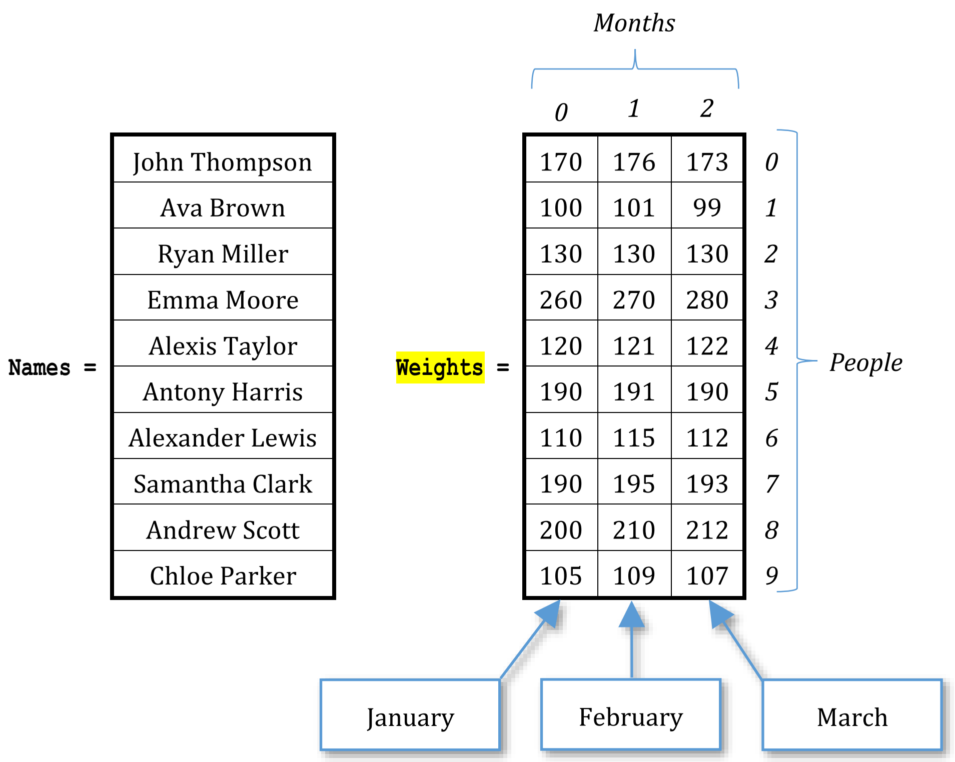
```
static void Main(string[] args) {
  int i;
  ulong grains, total;
  double weight;

  grains = 1;
  total = 1;
  for (i = 2; i <= 64; i++) {
    grains = 2 * grains;
    total = total + grains;
  }
  weight = total / 30000.0;

  Console.WriteLine(total + " " + weight);
}
```

Exercise 31.2-3 Designing Arrays

Design the necessary arrays to hold the names of ten people as well as the average weight (in pounds) of each person for January, February, and March. Then add some typical values to the arrays.

Solution

In this exercise, you need a one-dimensional array for names, and a two-dimensional array for people’s weights.



Names
John Thompson
Ava Brown
Ryan Miller
Emma Moore
Alexis Taylor
Antony Harris
Alexander Lewis
Samantha Clark
Andrew Scott
Chloe Parker

	0	1	2
0	170	176	173
1	100	101	99
2	130	130	130
3	260	270	280
4	120	121	122
5	190	191	190
6	110	115	112
7	190	195	193
8	200	210	212
9	105	109	107

31.6 How to Iterate Through a One-Dimensional Array

Second Approach

This approach is very simple but not as flexible as the previous one. There are cases where it cannot be used, as you will see below. Following is a code fragment, written in general form

```
foreach (var element in array_name) {
  process element;
}
```

31.7 How to Add User-Entered Values to a One-Dimensional Array

There is nothing new here. Instead of reading a value from the keyboard and assigning that value to a variable, you can directly assign that value to a specific array element. The next C# program prompts the user to enter the names of four people, and assigns them to the elements at index positions 0, 1, 2, and 3, of the array names.

31.13 Review Questions: True/False

35. If array b contains 30 elements (arithmetic values), the following code fragment doubles the values of all of its elements.

```
for (i = 29; i >= 0; i--) {
  b[i] = b[i] * 2;
}
```

31.14 Review Questions: Multiple Choice

10. If array b contains 30 elements (arithmetic values), the following code fragment

```
for (i = 29; i >= 1; i--) {
  b[i] = b[i] * 2;
}
```

a. doubles the values of some of its elements.
b. doubles the values of all of its elements.
c. none of the above

33.3 Processing Each Column Individually

First Approach – Creating an auxiliary array

```
s = 0;
for (i = 0; i <= ROWS - 1; i++) {
  s += b[i, j];
}
total[j] = s;
```

This program can equivalently be written as

```
total[j] = 0;
for (i = 0; i <= ROWS - 1; i++) {
  total[j] += b[i, j];
}
```

Now, nesting this code fragment in a for-loop that iterates for all columns results in the following.

```
for (j = 0; j <= COLUMNS - 1; j++) {
  total[j] = 0;
  for (i = 0; i <= ROWS - 1; i++) {
    total[j] += b[i, j];
  }
}
```

Second Approach – Just find it and process it.

This approach uses no auxiliary array; it just calculates and directly processes the sum. The code fragment is as follows.

```
for (j = 0; j <= COLUMNS - 1; j++) {
  total = 0;
  for (i = 0; i <= ROWS - 1; i++) {
    total += b[i, j];
  }
  process total;
}
```

Accordingly, the following code fragment calculates and displays the average value of each column.

```
for (j = 0; j <= COLUMNS - 1; j++) {
  total = 0;
  for (i = 0; i <= ROWS - 1; i++) {
    total += b[i, j];
  }
  Console.WriteLine(total / ROWS);
}
```

Exercice 33.4-1 Finding the Average Value of Two Grades

project_33_4_1

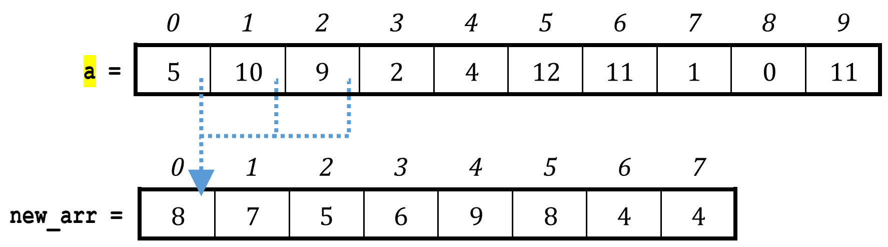
```
const int STUDENTS = 20;

static void Main(string[] args) {
  int i, total;
  double average;

  string[] names = new string[STUDENTS];
  int[] grades_lesson1 = new int[STUDENTS];
  int[] grades_lesson2 = new int[STUDENTS];

  for (i = 0; i <= STUDENTS - 1; i++) {
    Console.Write("Enter student name No" +(i + 1) + ": ");
    names[i] = Console.ReadLine();

    Console.Write("Enter grade for lesson 1: ");
    grades_lesson1[i] = Int32.Parse(Console.ReadLine());

    Console.Write("Enter grade for lesson 2: ");
    grades_lesson2[i] = Int32.Parse(Console.ReadLine());
  }

  //Calculate the average grade for each student
  //and display the names of those who are greater than 89
  for (i = 0; i <= STUDENTS - 1; i++) {
    total = grades_lesson1[i] + grades_lesson2[i];
    average = total / 2.0;
    if (average > 89) {
      Console.WriteLine(names[i]);
    }
  }
}
```

Exercise 34.1-1 Creating an Array that Contains the Average Values of its Neighboring Elements

Write a C# program that lets the user enter 100 positive numerical values into an array. Then, the program must create a new array of 98 elements. This new array must contain, in each position the average value of the three elements that exist in the current and the next two positions of the given array.

Solution

Let's try to understand this exercise through an example using 10 elements.



Array `new_arr` is the new array that is created. In array `new_arr`, the element at position 0 is the average value of the elements in the current and the next two positions of array `a`; that is, (5 + 10 + 9) / 3 = 8. The element at position 1 is the average value of the elements in the current and the next two positions of array `a`; that is, (10 + 9 + 2) / 3 = 7, and so on.

Exercise 34.1-4 Merging Two-Dimensional Arrays

project_34_1_4

```
const int COLUMNS = 4;

static void Main(string[] args) {
  int i, j;

  int[,] a = {
    {10, 11, 12, 85},
    {3, 1, 5, 10},
    {-1, 2, -5, -10}
  };

  int[,] b = {
    {10, 11, 16, 33},
    {11, 13, 5, 55},
    {-1, -2, -4, 44},
    {55, 33, 77, 12},
    {-110, 120, 132, 43}
  };

  int rows_of_a = a.Length / COLUMNS;
  int rows_of_b = b.Length / COLUMNS;

  //Create array new_arr
  int[,] new_arr = new int[rows_of_a + rows_of_b, COLUMNS];
  for (i = 0; i <= rows_of_a - 1; i++) {
    for (j = 0; j <= COLUMNS - 1; j++) {
      new_arr[i, j] = a[i, j];
    }
  }
  for (i = 0; i <= rows_of_b - 1; i++) {
    for (j = 0; j <= COLUMNS - 1; j++) {
      new_arr[rows_of_a + i, j] = b[i, j];
    }
  }

  //Display array new_arr
  for (i = 0; i <= rows_of_a + rows_of_b - 1; i++) {
    for (j = 0; j <= COLUMNS - 1; j++) {
      Console.Write(new_arr[i, j] + "\t");
    }
    Console.WriteLine();
  }
}
```

Exercise 34.1-5 Creating Two Arrays – Separating Positive from Negative Values

Note that the arrays `pos` *and* `neg` *contain a total number of* `pos_index` *and* `neg_index` *elements respectively. This is why the two last loop control structures iterate until variable* `i` *reaches values* `pos_index - 1` *and* `neg_index - 1`*, respectively, and not until* `ELEMENTS - 1`*, as you may mistakenly expect.* ~~*Obviously the sum of* `pos_index + neg_index` *equals to* `ELEMENTS`*.*~~

Exercise 34.4-7 The Five Best Scorers

Now, in order to sort all rows, you need to nest this code fragment in a for-loop that iterates for all of them, as shown next.

```
for (i = 0; i <= TEAMS - 1; i++) {
  swaps = false;
  for (m = 1; m <= PLAYERS - 1; m++) {
    for (n = PLAYERS - 1; n >= m; n--) {
      if (g[i, n] < g[i, n - 1]) {
        temp = g[i, n];
        g[i, n] = g[i, n - 1];
        g[i, n - 1] = temp;

        temp_str = p[i, n];
        p[i, n] = p[i, n - 1];
        p[i, n - 1] = temp_str;
      }
    }
    if (!swaps) break;
  }
}
```

Exercise 34.4-9 Sorting One-Dimensional Arrays While Preserving the Relationship with a Second Array

Write a C# program that prompts the user to enter the total number of kWh consumed each month for a period of one year. It then displays each number of KWh consumed (in descending order) along with the name of the corresponding month. Use the selection sort algorithm.

36.2 How to Make a Call to a Method

Every call to a method is as follows: you write the name of the method followed by a list of arguments (if required), either within a statement that assigns the method's returned value to a variable or directly within an expression.

Let's see some examples. The following method accepts an argument (a numeric value) and returns the result of that value raised to the power of three.

```
static double cube(double num) {
  double result;

  result = num * num * num;
  return result;
}
```

Now, suppose that you want to calculate a result using the following expression

$$y = x^3 + \frac{1}{x}$$

37.10 Review Questions: True/False

17. Optional arguments must be on the left side of any non-optional arguments.

Exercise 38.1-5 Finding the Average Values of Positive Integers

Note the last single-alternative decision structure, `if (count > 0)`*. It is necessary in order for the program to satisfy the property of definiteness. Think about it! If the user enters a real (float) right from the beginning, the variable* `count`*, in the end, will contain a value of zero.*

Exercise 38.2-3 Progressive Rates and Electricity Consumption

The LAV Electricity Company charges subscribers for their electricity consumption according to the following table (monthly rates for domestic accounts).

Kilowatt-hours (kWh)	USD per kWh
kWh ≤ 400	$0.08
401 ≤ kWh ≤ 1500	$0.22
1501 ≤ kWh ≤ 2000	$0.35
2001 ≤ kWh	$0.50

39.9 Review Exercises

10. During the Cold War after World War II, messages were encrypted so that if the enemies intercepted them, they could not decrypt them without the decryption key. A very simple encryption algorithm is alphabetic rotation. The algorithm moves all letters N steps "up" in the alphabet, where N is the encryption key. For example, if the encryption key is 2, you can encrypt a message by replacing the letter A with the letter C, the letter B with the letter D, the letter C with the letter E, and so on. Do the following:
    i. Write a class named `EncryptDecrypt` that includes
        a. private integer field named `_encr_decr_key`.
        b. a property named `Encr_decr_key`. It will be used to get and set the value of the field `_encr_decr_key`. The getter must throw an error when the field has not yet been set, and the setter must throw an error when the field is not set to a value between 1 and 26.