# CHAPTER 1: SOLUTIONS TO REVIEW EXERCISES

**Solution R1.1:** Using a computer does not require any advanced know-how or skill. Most people are capable of turning a computer on and executing a program (such as double clicking on the icon for Microsoft Word to bring up the word processor to type a letter). Programming a computer, however, requires very specific knowledge. You must know the rules of the programming language, and you must know how to construct a program.

As an analogy, using a computer program would be like being able to drive a car, while having the knowledge & skills to be an auto mechanic would be like being able to program a computer.

**Solution R1.2:** When actively being used,both program code and user data are maintained in *primary storage*. Primary storage consists of random access memory (RAM), which hold the instructions of a program for the computer to execute, as well as data (which can be changed by the program, as it is executed). RAM is relatively expensive, and is volatile. Memory being volatile means that if power us turned off, whatever programs or data were being held are then lost.

For long-term storage (such as when the computer is turned off), both program code and data are typically stored ~~long term~~ in a computer’s *secondary storage*, such as a hard disk, or thumb drive. Secondary storage is relatively inexpensive and retains information even when the computer’s power is turned off; this is referred to as non-volatile memory.

~~In addition to a computer’s secondary storage, program code and data can also be stored in a computer’s~~ *~~primary storage~~*~~.~~  ~~Primary storage consists of read-only memory (ROM), which holds programs that must always be present (such as the computer’s operating system), and random access memory (RAM), which can hold changing data and programs that are currently executing.~~ ~~RAM is relatively expensive when compared to secondary storage, and is erased whenever the computer is turned off.~~

**Solution R1.3:** The user of a computer receives information from the computer’s display screen, speakers, and printers (the computer’s output devices). The user can input data using the computer’s keyboard, a pointing device such as a mouse, a microphone, or a webcam (the computer’s input devices).

**Solution R1.4:** A simple cell phone that can only be used to make calls would be a single-function device. If you can download programs (such as media players, web browsers, and games) to the cell phone and execute them, then it would be considered a programmable computer, as it can serve multiple purposes.

**Solution R1.5:** One advantage of C++ over machine code is that C++ statements are independent of the machine (computer) they are being executed on; machine code statements differ from one type of machine to the next. Another advantage of C++ is that it is much more readable and understandable by humans than machine code.

**Solution R1.6:** The hello.cpp file that I created was found in the following folder on my computer. The location will vary from machine to machine.

C:\Documents and Settings\Jonathan\My Documents\Visual Studio 2005\Projects\Hello\Hello

The iostream file was found in the following folder on my computer. The location will vary from machine to machine.

C:\Program Files\Microsoft Visual Studio 8\VC\include

As you can see, the location of the files on my computer shows that the computer I use has the Microsoft Visual Studio development environment installed on it.

**Solution R1.7:** The program prints the following:

6 \* 7 = 42

**Solution R1.8:** The program prints the following:

HelloWorld

**Solution R1.9:** The program prints the following:

Hello

World

**Solution R1.10:** Programs 1 to 3 below each have a different compile-time error (an error that violates the rules of C++). Program 4 has a run-time error (an error that will allow the program to execute, but which fails to provide the correct results).

**Program 1 (compile-time error – no end quote on string, in line 6):**

1 #include <iostream>

2 using namespace std;

3

4 int main()

5 {

6 cout << "Hello, World! << endl;

7 return 0;

8 }

**Program 2 (compile-time error – no semicolon on the cout statement, in line 6):**

1 #include <iostream>

2 using namespace std;

3

4 int main()

5 {

6 cout << "Hello, World!" << endl

7 return 0;

8 }

**Program 3 (compile-time error – no parentheses for main function, in line 4):**

1 #include <iostream>

2 using namespace std;

3

4 int main

5 {

6 cout << "Hello, World!" << endl;

7 return 0;

8 }

**Program 4 (run-time error – “World” is misspelled, in line 6):**

1 #include <iostream>

2 using namespace std;

3

4 int main()

5 {

6 cout << "Hello, Wolrd!" << endl;

7 return 0;

8 }

**Solution R1.11:** A compile-time error is typically found by the compiler during the compilation process. A compile-time error is caused when the source code violates the rules of the programming language being used.

A run-time error cannot be found by the compiler. It is found by testing the program and carefully examining the output or results for errors.**Solution R1.12:** Example algorithm to determine number of years until account depletion:

1. Set the initial account\_value to $10,000.

Set the initial number\_of\_months to 0.

2. Repeat the following while account\_value is greater than $0:

A. Set account\_value equal to account\_value times 1.005.

B. Deduct $500 from account\_value.

C. Increment number\_of\_months by 1.

1. Print the total number\_of\_months, as well as number\_of\_months divided by 12, to show how many years it took until the account was depleted.

The account is depleted after 22 months, or 1.83333 years.

**Solution R1.13:** If the user was allowed to enter the inputs into the algorithm written in Solution R.1.12 above, there are a few number combinations that may not terminate (ie, the program would run forever). For example, if the amount of interest added each month is greater than the amount withdrawn each month, then the algorithm will never terminate. We can fix this by adding some simple changes to the algorithm:

1. User sets account\_value.

User sets interest\_rate.

User sets withdraw\_amount.

1. If the account\_value times the interest rate is greater than or equal to the withdraw\_amount, print “The account never depletes.”, and terminate the program; otherwise:
   1. Repeat the following while account\_value is greater than $0:

i. Set account\_value equal to account\_value times (1 + interest\_rate).

ii. Deduct withdraw\_amount from account\_value.

iii. Increment number\_of\_months by 1.

* 1. Print the total number\_of\_months, as well as number\_of\_months divided by 12, to show how many years it took until the account was depleted.

We would have a similar problem if the user entered a negative value for the withdraw\_amount, which would have the effect of increasing the account instead of decreasing the account. Other problems would occur, if the other user entered values were negative. Input validation is a separate topic, which will be covered later in the text.

For the current problem, we will assume the user will not enter a negative amount.

**Solution R1.14:** Example algorithm to determine the exterior surface area of a house:

1. Calculate the wall\_area of house by using the formula:

wall\_area = (2 \* building\_height \* building\_length) + (2 \* building\_height \* building\_width)

2. Calculate the door\_area by using the formula:

door\_area = number\_of\_doors \* door\_height \* door\_width

3. Calculate the window\_area by using the formula:

window\_area = number\_of\_windows \* window\_height \* window\_width

4. Calculate the paintable\_area by using the formula:

paintable\_area = wall\_area – door\_area – window\_area

**Solution R1.15:** Example algorithm to determine cheapest method of getting to work (“mpg” stands for miles per gallon):

1. Calculate the number of gallons of gas used by using the formula:

gallons\_used = distance / mpg

2. Calculate the cost of the gas used:

cost\_of\_gas = gallons\_used \* $4

3. Calculate total cost of driving including maintenance:

total\_cost = cost\_of\_gas + (distance \* 0.05)

4. If total\_cost of driving is greater than cost of ticket, then print “Taking the train is cheaper,” otherwise print “Driving is as cheap, or cheaper than the train.”

**Solution R1.16: Every day at 5 P.M., please do the following:**

1. **Insert the USB memory stick into the USB port of the computer.**
2. **On the USB memory stick, create a new folder and name it “BACKUP”-date (where date is the date you are backing up files).**
3. **Copy the files from the “My Documents” folder on the computer to the new folder you just created on the USB memory stick.**
4. **Double-check to make sure the new folder contains the files you just copied. If the folder is still empty, something went wrong. It’s possible you backed-up the wrong folder. Please try it again, and be careful which folder you copy from, and to.**
5. **Once you’ve verified the copy is complete, remove the USP memory stick from the computer, and put it back where you got it from.**
6. **Thank you!!!**

**Solution R1.17:** Example algorithm to determine if a discount card is worthwhile. For this algorithm, the following pieces of information are required:

1. Card\_cost
2. Card\_number\_of\_days
3. Meal\_rate
4. Average\_meal\_cost
5. Meals\_per\_day

Then, perform the following calculation:

1. Calculate the total\_number of meals to be considered as the product of card\_number\_of\_days and meals\_per\_day.
2. Calculate the number\_of\_free\_meals by dividing the total\_number\_of\_meals by (1 + meal\_rate). (You have to pay for the meal\_rate number of meals, and then the next one is free.)
3. Use only the integer part of this result, because you can’t get a partial free meal.
4. If the product ( number\_of\_free\_meals \* average\_meal\_cost ) is greater than the card\_cost, then print: “Get the card and save some money.”

**Solution R1.18:**

From the refrigerator:

Get orange juice

Get eggs

Get bacon

From the pantry:

Get pancake powder

1. Prepare pancake batter.
2. To cook on the griddle for each person eating breakfast:
   1. Crack an egg on the griddle.
   2. Pour 2 pancakes on the griddle.
   3. Place 2 slices of bacon on the griddle.
3. For each place at the breakfast table:
   1. Pour a glass of orange juice.
   2. Serve a plate of griddle-cooked breakfast.
4. Call the family to breakfast!

**Solution R1.19:**  Example algorithm to calculate the square root of a number, to a particular tolerance of accuracy (say, to within 0.000001):

1. Get a number, a, from the user.
2. Calculate guess1 = a / 2.
3. Calculate guess2 = a / guess1.
4. While the absolute value of (guess1 – guess2) is greater than tolerance:
   1. Calculate guess1 = (guess1 + guess2) / 2.
   2. Calculate guess2 = a / guess1.
5. Print “The square root of “ a “ is “ guess1.

**Solution R1.20:**  The requirement for the boundary to be equally spaced from the center cluster means that, in addition to the center cluster being either 2 or 3 tiles wide, it will also either be 1 or 2 tiles high. An example algorithm for placing an array of rows by columns tiles is:

1. If rows is odd, let fill\_rows be (rows – 3) / 2; otherwise, let fill\_rows be (rows – 4) /2.
2. If columns is odd, let fill\_columns be (columns – 5) / 2; otherwise, let fill\_columns be (columns – 4) / 2.
3. For one row, place columns number of black tiles in a row.
4. For fill\_rows number of rows:
   1. Place a black tile.
   2. Place (columns – 2) white tiles to the right of the black tile.
   3. Place a black tile to the right of the white ones, to finish each row.
5. For one row:
   1. Place a black tile.
   2. Place fill\_columns of white tiles to the right of the black tile.
   3. If columns is odd, place 3 black tiles to the right of the white tiles; otherwise place 2 black tiles to the right of the white tiles.
   4. Place fill\_columns of white tiles to the right of the black tiles.
   5. Place a black tile to the right of the white tiles, to finish the row.
6. If rows is even, repeat step 5, once; otherwise skip to step 7.
7. Repeat step 4.
8. Repeat step 3.

**Solution R1.21:**  Example algorithm to create a concentric tile pattern of alternating white and black squares:

1. From left to right,
   1. place eleven black tiles in a row.
2. Move down one row.
3. Start over, right below the left-most tile.
4. From left to right,
   1. place a black tile,
   2. then nine white tiles,
   3. then a black tile.
5. Move down one row.
6. Start over, right below the left-most tile.
7. From left to right,
   1. place a black tile,
   2. then a white tile,
   3. then seven black tiles,
   4. then a white tile,
   5. then a black tile.
8. Move down one row.
9. Start over, right below the left-most tile.
10. From left to right,
    1. place a black tile,
    2. then a white tile,
    3. then a black tile,
    4. then five white tiles,
    5. then a black tile,
    6. then a white tile,
    7. then a black tile.
11. Move down one row.
12. Start over, right below the left-most tile.
13. From left to right,
    1. place a black tile,
    2. then a white tile,
    3. then a black tile,
    4. then a white tile,
    5. then three black tiles,
    6. then a white tile,
    7. then a black tile,
    8. then a white tile,
    9. then a black tile.
14. Move down one row.
15. Start over, right below the left-most tile.
16. From left to right,
    1. place a black tile,
    2. then a white tile,
    3. then a black tile,
    4. then a white tile,
    5. then a black tile,
    6. then a white tile,
    7. then a black tile,
    8. then a white tile,
    9. then a black tile,
    10. then a white tile,
    11. then a black tile.
17. Repeat steps 11 – 13.
18. Repeat steps 8 – 10.
19. Repeat steps 5 – 7.
20. Repeat steps 2 – 4.
21. Repeat step 1.
22. Stop.

Solution to R1.22: Example algorithm to instruct a robot to mow a rectangular lawn:

1. Use a ‘turn’ variable, and set its’ value to “right”.
2. To start, check the color of the unit in front.
   1. If the color is red,
      1. turn right 90 degrees,
      2. repeat Step 2;
   2. otherwise continue to Step 3.
3. Move forward one unit.
4. Check the color of the unit in front.
   1. If the color is red
      1. If the ‘turn’ value is “right”
         1. Turn right 90 degrees.
         2. Check the color of the unit in front
            1. If the color is red, Stop Mowing.
            2. Otherwise, continue with Step 4)a.i.3.
         3. Move forward one unit.
         4. Turn right 90 degrees.
         5. Move forward one unit.
         6. Change ‘turn’ to “left”.
      2. Otherwise the ‘turn’ value is “left”
         1. Turn left 90 degrees.
         2. Check the color of the unit in front.
            1. If the color is red, Stop Mowing.
            2. Otherwise, continue with Step 4)a.ii.3.
         3. Move forward one unit.
         4. Turn left 90 degrees.
         5. Move forward one unit.
         6. Change ‘turn’ to “right”.
   2. Otherwise the color is green, so repeat Step 3.

Solution to R1.23: Example algorithm to count the number of windows in a room, starting from any position in the room: (we will count the number of walls, to keep track of what we’re doing {there are four walls}, and count the number of windows, to provide the solution to this problem). Also, because the size of a window is unknown, an additional variable, ‘window’, will be used to keep track of what the robot is sensing (‘window’ = 1 means there is a window, while ‘window’ = 0 means there is a wall).

Main algorithm:

1. Set ‘number\_of\_walls’, ‘number\_of\_windows’, and ‘window’ all to zero.
2. Find a corner, and set the starting position.
3. For each wall, examine the wall and count the number of windows.
4. When finished, display the number of windows.
5. Stop.

Finding a corner:

1. Check what is in front of the robot.
2. If neither a wall or window,
   1. move forward,
   2. repeat Step 1 of ‘finding a corner’.
3. otherwise,
   1. add 1 to ‘number\_of\_walls’, then
   2. turn right.
   3. If ‘number\_of\_walls’ is 2,
      1. Set ‘number\_of\_walls’ to 0.
      2. Step 2 of the main algorithm is done, continue with Step 3 of the main algorithm.
   4. Otherwise, repeat Step 1 of ‘finding a corner’.

Examining the wall:

1. Turn left.
2. Check what is in front of the robot.
3. If a window is sensed,
   1. If 'window’ is 0,
      1. Change ‘window’ to 1.
      2. Add 1 to ‘number\_of\_windows’
   2. Otherwise make no changes, it’s just a big window.
4. Otherwise,
   1. Set ‘window’ to 0.
5. Turn right.
6. Check what is in front of the robot.
7. If neither a wall or window,
   1. Move forward.
   2. Begin again at Step 1 of ‘examining the wall’
8. Otherwise
   1. Add 1 to ‘number\_of\_walls’
   2. If ‘number\_of\_walls’ is 4,
      1. Stop examining walls, and
      2. Return to the main algorithm, Step 4.
   3. Otherwise, of a window is sensed,
      1. Change ‘window’ to 1.
      2. Add 1 to ‘number\_of\_windows’
   4. Otherwise,
      1. Change ‘window’ to 0.
   5. Turn right.
   6. Move forward.
   7. Begin again at Step 1 of ‘examining the wall’

Solution to R1.24: Example algorithm for using the right-hand rule to escape a maze:

1. Find a wall.
   1. Sense what is in front of the robot.
   2. If neither a wall or exit
      1. Move forward.
      2. Repeat Step 1)a.
   3. Otherwise, if the exit,
      1. Move forward.
      2. Stop, the maze has been exited.
   4. Otherwise,
      1. Turn left. (the wall will now always be on the right of the robot.)
      2. Continue with step 2).
2. Execute the right-hand rule.
   1. Sense what is in front of the robot.
   2. If a wall, (left-turn corner)
      1. Turn left.
      2. Continue with Step 2)a.
   3. Otherwise, if an exit,
      1. Move forward.
      2. Stop, the maze has been exited.
   4. Otherwise, neither a wall or exit,
      1. Move forward.
      2. Turn right.
      3. Sense what is in front of the robot.
      4. If a wall,
         1. Turn left.
         2. Move forward.
         3. Continue with step 2)a.
      5. Otherwise, if an exit,
         1. Move forward.
         2. Stop, the maze has been exited.
      6. Otherwise, neither a wall or exit,(a right-turn corner)
         1. Move forward.
         2. Continue with step 2)a.

Supplemental Business Exercise Solutions

Solution to BR1.25: Example algorithm for finding the best options from the loyalty promotions online catalog. To assess the online catalog, we will keep track of the current ‘best\_price’, and a list of items which cost that much.

1. Set ‘best\_price’ to zero, and clear the ‘list\_of\_items’.
2. Check the next catalog item:
   1. If there are no more catalog items:
      1. Display the ‘list\_of\_items’, and the ‘best\_price’.
      2. Stop.
   2. If the price is greater than $100, or less than the ‘best\_price’
      1. Skip the item.
      2. Continue with Step 2).
   3. Otherwise, if the price is greater than ‘best\_price’,
      1. Set ‘best\_price’ to price.
      2. Clear the ‘list\_of\_items’.
      3. Add the item to the ‘list\_of\_items’.
      4. Continue with Step 2).
   4. Otherwise, the price matches ‘best\_price’.
      1. Add the item to the ‘list\_of\_items’.
      2. Continue with Step 2).

**Supplemental Engineering Exercise Solutions**

**Solution to ER1.26:** Algorithm to find television dimensions:

Using the aspect ratio of 16:9, the diagonal is 18.36.

Prompt the user for the ‘diagonal’ measurement of a television.

Divide ‘diagonal’ by 18.36, and name it ‘hypoteneuse’.

i.e.: hypoteneuse = diagonal / 18.36

To find the ‘width’, multiply ‘hypoteneuse’ by 16.

i.e.: width = hypoteneuse \* 16

To find the ‘height’, multiply ‘hypoteneuse’ by 9.

i.e.: height = hypoteneuse \* 9

Display the ‘height’ and ‘width’, with appropriate lables.

**Solution to ER1.27: Algorithm to determine if “red eye” is present:**

**Given there are ‘rows’ number of rows, and ‘columns’ number of columns,**

**Find the centers of both the black elements, and the red elements.**

**If the centers coincide, indicate that “red eye” is present.**

* + - 1. **Set ‘black\_row’ and ‘black\_column’ to 0.**
      2. **Set ‘red\_row’ and ‘red\_column’ to 0.**
      3. **Set ‘number\_red\_pixels’ and ‘number\_black\_pixels’ to 0.**
      4. **Step through each ‘row’, one at a time.**
      5. **For each ‘row’, step through each ‘column’, one at a time.**
      6. **At each (‘row’, ‘column’) combination, assess the (‘row’, ‘column’) pixel’s color:**
      7. **If the pixel is:**
         1. **White, skip to step 8.**
         2. **Red:**

**add ‘row’ to ‘red\_row’**

**add ‘column’ to ‘red\_column’**

**add 1 to ‘number\_red\_pixels’**

**Skip to step 8.**

* + - * 1. **Black:**

**Add ‘row’ to ‘black\_row’**

**Add ‘column’ to ‘black\_column’**

**Add 1 to ‘number\_black\_pixels’**

* + - 1. **Advance:**
         1. **If the last row is complete, skip to step 9.**
         2. **If the row is complete, advance to the next row, first column, then return to step 7.**
         3. **to the next column, then return to step 7.**

1. **Once all of the (‘row’, ‘column’) combinations have been checked:**
   1. **Divide ‘red\_row’ by ‘number\_red\_pixels’, and assign the value to ‘red\_center\_row’**
   2. **Divide ‘red\_columns’ by ‘number\_red\_pixels’, and assign the value to ‘red\_center\_column’**
   3. **Divide ‘black\_row’ by ‘number\_black\_pixels’, and assign the value to ‘black\_center\_row’**
   4. **Divide ‘black\_column’ by ‘number\_black\_pixels’, and assign the value to ‘black\_center\_column’**
2. **If ‘red\_center\_row’ is equal to ‘black\_center\_row’, and ‘red\_center\_column’ is equal to ‘black\_center\_column’, display “yes”, otherwise, display “no”.**

**Solution to ER1.28: Algorithm to calculate taxi fare:**

**Given a ‘distance’ (in miles), and ‘travel\_time’ (in minutes):**

* + - 1. **If ‘distance’ is less than 0.2 miles**
         1. **Set ‘mileage\_cost’ = $3.50**

**Otherwise,**

* 1. **set ‘mileage\_cost’ = $3.50 + [ int( 5 \* ‘distance’ – 1 )] \* $0.55.**
  2. **Furthermore, if ( 5 \* ‘distance’ ) is greater than ( int( 5 \* ‘distance’ ), then add $0.55 more to ‘mileage\_cost’.**

2. ‘delay\_cost’ = ( time – 5 \* distance ) \* $0.55.

3. ‘fare’ = ‘mileage\_cost’ + ‘delay\_cost’.

**Solution to ER1.29: Algorithm to calculate travel time with acceleration:**

**Given a ‘distance’ (in miles), and ‘velocity’ (in mph), and a constant acceleration is given by how long it takes to reach 60 mph, t60 (in seconds):**

* + - 1. The time it takes to accelerate up to ‘velocity’ is: ta = ( ‘velocity’ / 60 ) \* t60.
      2. The distance travelled (in miles) while accelerating is: Da = (1/2)\*(t60/60mph)\*(velocity2)/3600. (note: there are both hours & seconds being used. Dividing by 3600 compensates for the mismatch in units.)
      3. The time it takes to finish the trip at the desired ‘velocity’ is: tv = (‘distance’ – Da) / ‘velocity’.
      4. The total time to complete the trip, ttotal = ta + tv.

**Solution to ER1.30: Algorithm to calculate the tire radius desired to cheat riders:**

**Given the true distance, x, (in miles), the desired cheat distance, y, (also in miles), and the proper tire radius, R, (in inches):**

1. First, find the false number of rotations (with units, miles per inch): faux\_rotations = y / (2 \* pi \* R).
2. The cheat circumference would then be: cheat\_circumference = x / faux\_rotations.
3. Which leads directly to the cheat radius: cheat\_radius = cheat\_circumference / (2 \* pi)