Invitation to Computer Science 8th edition Schneider Solutions Manual

SKU: 9781337561914-SOLUTIONS

Solutions to End-of-Chapter Exercises

Chapter 1: An Introduction to Computer Science

1. There is no one correct answer. Common examples are the instructions for using a voice mail system, the instructions for opening a mail box lock, and the instructions for doing laundry.

2. A *heuristic* is a method for finding a reasonably close, "good enough" solution to a problem. It can be viewed as a rule-of-thumb, a method of approximation, an informal technique, or even a way to make an "educated guess." It differs from the concept of an algorithm in that it does not guarantee to produce an optimal solution, just to make a good faith attempt to locate a reasonable one. Heuristics are often used when executing an algorithm might be too time-consuming, and we only need an approximation to the correct answer.

An example of a heuristic for adding two 3-digit numbers, such as 234 + 567, might be:

1. Set the one and tens digit of both operands to 0
2. Increase the hundreds digit of the second operand by 1. These two steps result in changing the problem to the simpler one 200 + 600.
3. Add the hundreds digits, resulting in a final "answer" of 800.

Now, of course, this is not the correct answer, which is 801. But the result we get may be close enough for our needs, and it is certainly a lot easier to add a single column of numbers rather than three columns of numbers.

3. One may argue that the instruction is not well-ordered, since it is unclear whether one should enter the channel first or press CHAN first. Also, it may not be effectively computable if you desire to enter a channel that is out of the DVR's range.

4. (a) Sequential
(b) Conditional
(c) Sequential
(d) Iterative

5. Step 1: *carry* = 0, c_3 = ??, c_2 = ??, c_1 = ??, and c_0 = ??

Step 2: $i = 0$, all others unchanged

Step 4: c_0 = 18, all others unchanged

Step 5: c_0 = 8 and *carry* = 1, all others unchanged

Step 6: i = 1, *carry* = 1, c_3 = ??, c_2 = ??, c_1 = ??, and c_0 = 8

Step 4: c_1 = 7, all others unchanged

Step 5: *carry* = 0, all others unchanged

Step 6: i = 2, *carry* = 0, c_3 = ??, c_2 = ??, c_1 = 7, and c_0 = 8

Step 4: $c_1 = 1$, all others unchanged

Step 5: *carry* = 0, all others unchanged

Step 6: $i = 3$, *carry* = 0, c_3 = ??, $c_2 = 1$, $c_1 = 7$, and $c_0 = 8$

Step 7: $c_3 = 0$, $c_2 = 1$, $c_1 = 7$, and $c_0 = 8$

Step 8: Print out 0178.

6. Replace Step 8 with the following steps:

Step 8: Set the value of i to m

Step 9: Repeat step 10 until either c_i is not equal to 0 or $i < 0$

Step 10: Subtract 1 from i, moving one digit to the right

Step 11: If $i \geq 0$ then print $c_i c_{i-1} \ldots c_0$

7. Assume that a has n digits $a_{n-1}, \ldots, a_0$, and b has m digits, $b_{m-1}, \ldots, b_0$, with n not necessarily equal to m. Add an operation at the beginning of the algorithm that resets the two numbers to the same number of digits by adding non-significant leading zeros to the shorter one. We can then reuse the algorithm of Figure 1.2.

```
If (m > n)  then
        Set i to 0
        While (n+i < m)
                Add a leading zero to the number at position  a(n+i)
                Increment i by 1
        End of the loop
Else
        If (n > m)
                Set i to 0
                While (m+i < n)
                        Add a leading zero to the number at position  b(m+i)
                        Increment i by 1
                End of the loop
```

We have now made the two numbers equal in length. All we need do now is set the variable m to the larger of the two values:

Set m to the larger of m and n.

The addition algorithm in Figure 1.2 will now work correctly. Note that if m and n are equal in value, neither of the Boolean expressions will be true, and neither of the conditional statements will be executed.

8. It is not effectively computable if $b^2 - 4ac < 0$ (since we cannot take the square root of a negative number if we are limited to real numbers) or if $a = 0$ (since we cannot divide by 0).

9. The first algorithm (Figure 1.3(a)) is a better general purpose algorithm. If you want to shampoo your hair any number *n* times you can change the 2 to *n*. You could even ask the shampooer to input the desired number *n* of washings. For the second algorithm you would have to rewrite the algorithm to repeat steps 4 and 5 998 more times.

10. (a) Trace:

Step 1: $I = 32$, $J = 20$, and R = ??

Step 2: $I = 32$, $J = 20$, and $R = 12$

Step 3: $I = 20$, $J = 12$, and $R = 12$

Step 2: $I = 20$, $J = 12$, and $R = 8$

Step 3: $I = 12$, $J = 8$, and $R = 8$

Step 2: $I = 12$, $J = 8$, and $R = 4$

Step 3: $I = 8$, $J = 4$, and $R = 4$

Step 2: $I = 8$, $J = 4$, and $R = 0$

Step 4: Print $J = 4$

(b) At Step 2 we are asked to divide $I = 32$ by $J = 0$, which cannot be done. We can fix the problem by adding a step between Step 1 and Step 2 that says: If $J = 0$, then print "ERROR: division by 0" and Stop.

11. There are 25! possible paths to be considered. That is approximately 1.5×10^{25} different paths. The computer can analyze 10,000,000, or 10^7, paths per second. The number of seconds required to check all possible paths is about $1.5 \times 10^{25}/10^7$, or about 1.5×10^{18} seconds. That's roughly 10^{12} years: about a trillion years. This would not be a feasible algorithm.

12. A Multiplication Algorithm.

Given: Two positive numbers *a* and *b*

Wanted: A number *c* which contains the result of multiplying *a* and *b*

Step 1: Set the value of *c* equal to 0

Step 2: Set the value of *i* equal to *b*

Step 3: Repeat steps 4 and 5 until the value of *i* is 0

Step 4: Set the value of c to be $c + a$

Step 5: Subtract 1 from i

Step 6: Print out the final answer c

Step 7: Stop

This algorithm assumes that we know how to add two multiple-digit numbers together. We may assume this because we have the algorithm from the book which does exactly that.

13. The algorithm will work correctly only if all three numbers are unique. If two or more numbers are identical, none of the Boolean expressions will be true and nothing will be output. To make this a correct solution you either have to specify in the problem statement that the three numbers provided must all be distinct or (better) change all of the comparison operations to ≥ in place of >.

14. This is an essay question. Students may find excellent resources on the Internet.

15. If this problem is assigned, be sure to coordinate with your computing staff ahead of time for students to get the required information.

16. This is an essay question. Because this is a "hot" topic, a great deal of hype and hyperbole is available, as well as useful information. It might be a good opportunity to teach students about finding *reliable* sources on the Internet, and evaluating online and print source materials.

17. Like question 16 this is an essay question. Students may be familiar with Apple iCloud services for iPhone and iPad devices, so it might be a good opportunity to relate their answers to the services provided by Apple.

18. About 130 feet ((((700,000,000 chars/5 chars per word)/300 words per page)/300 pages per inch)/12 inch per foot)

Discussion of Challenge Work

1. We may perform subtraction, like addition, by subtracting one column at a time, starting with the rightmost column and working to the left. Since we know that the first number is larger than the second one, we know that we can always borrow from columns to the left of the current one. Therefore, if the upper number in the column (a_i) is smaller than the lower, we automatically borrow from the next column. We can do this by subtracting one from the a_{i+1} value of the column to the left. If the a_{i+1} value were already zero, then it would become -1. This automatically causes a borrow to occur on the next step. Here is the algorithm:

Step 1: Set the value of i equal to the value of 0

Step 2: Repeat steps 3 to 6 until the value of i is m

Step 3: If $b_i \leq a_i$ then

Step 4: Set c_i equal to a_i - b_i

Otherwise ($b_i > a_i$)

Step 5: Set c_i equal to (a_i + 10) – b_i and replace a_{i+1} with $a_{i+1} - 1$

(This amounts to a borrow of 1 from a_{i+1} which adds 10 to a_i)

Step 6: Add 1 to i (moving us one column to the left)

Step 7: Print out the final answer $c_{m-1}c_{m-2} \ldots c_0$

Step 8: Stop.

2. Students may need assistance finding or understanding other definitions from other sources.

Chapter 2: Algorithm Discovery and Design

1. (a) Set the value of *area* to ½($b \times h$)
(b) Set the value of *interest* to $I \times B$
Set the value of *FinalBalance* to $(1 + I) \times B$
(c) Set the value of *FlyingTime* to *M/AvgSpeed*

2. Algorithm:

Step 1: Get values for *B, I,* and *S*

Step 2: Set the value of *FinalBalance* to $(1 + I/12)^{12}B$

Step 3: Set the value of *Interest* to *FinalBalance – B*

Step 4: Set the value of *FinalBalance* to *FinalBalance – S*

Step 5: Print the message 'Interest Earned: '

Step 6: Print the value of *Interest*

Step 7: Print the message 'Final Balance: '

Step 8: Print the value of *FinalBalance*

3. Algorithm:

Step 1: Get values for *E1, E2, E3 and F*

Step 2: Set the value of *Ave* to $(E1 + E2 + E3 + 2F)/5$

Step 3: Print the value of *Ave*

4. Algorithm:

Step 1: Get values for *P* and *Q*

Step 2: Set the value of *Subtotal* to $P \times Q$

Step 3: Set the value of *TotalCost* to $(1.06) \times$ *Subtotal*

Step 4: Print the value of *TotalCost*

5. (a) If $y \neq 0$ then
Print the value of (x/y)

Else
Print the message 'Unable to perform the division.'

(b) If $r \geq 1.0$, then
Set the value of Area to $\pi \times r^2$
Set the value of *Circum* to $2 \times \pi \times r$

Else
Set the value of *Circum* to $2 \times \pi \times r$

6. Algorithm:

Step 1: Get a value for *B, I,* and *S*

Step 2: Set the value of *FinalBalance* to $(1 + I/12)^{12}B$

Step 3: Set the value of *Interest* to *FinalBalance* – *B*

Step 4: If $B < 1000$ then Set the value of *FinalBalance* to *FinalBalance* – *S*

Step 5: Print the message 'Interest Earned: '

Step 6: Print the value of *Interest*

Step 7: Print the message 'Final Balance: '

Step 8: Print the value of *FinalBalance*

7. Algorithm:

Step 1: Set the value of *i* to 1

Step 2: Set the values of *Won, Lost,* and *Tied* all to 0

Step 3: While $i \leq 10$ do

Step 4: Get the value of CSU_i and OPP_i

Step 5: If $CSU_i > OPP_i$ then

Step 6: Set the value of *Won* to *Won* + 1

Step 7: Else if $CSU_i < OPP_i$ then

Step 8: Set the value of *Lost* to *Lost* + 1

Step 9: Else

Step 10: Set the value of *Tied* to *Tied* + 1

Step 11: Set the value of *i* to *i* + 1

End of the While loop

Step 12: Print the values of *Won*, *Lost*, and *Tied*

Step 13: If *Won* = 10, then

Step 14: Print the message, 'Congratulations on your undefeated season.'

8. Algorithm:

Step 1: Set the value of *i* to 1

Step 2: Set the value of *Total* to 0

Step 3: While $i \leq 14$ do

Step 4: Get the value of E_i

Step 5: Set the value of *Total* to *Total* + E_i

Step 6: Set the value of *i* to *i* + 1

End of While loop

Step 7: Get the value of *F*

Step 8: Set the value of *Total* to *Total* + 2*F*

Step 9: Set the value of *Ave* to *Total* / 16

Step 10: Print the value of *Ave*

9. Algorithm:

Step 1: Set the value of *TotalCost* to 0

Step 2: Do

Step 3: Get values for *P* and *Q*

Step 4: Set the value of *Subtotal* to $P \times Q$

Step 5: Set the value of *TotalCost* to *TotalCost* + $(1.06) \times$ *Subtotal*

While (*TotalCost* < 1000)

Step 6: Print the value of *TotalCost*

10. The tricky part is in steps 6 through 9. If you use no more than 1000 kilowatt hours in the month then you get charged \$.06 for each. If you use more than 1000, then you get charged \$.06 for the first 1000 hours and \$.08 for each of the remaining hours. There are $M_i - 1000$ remaining hours, since M_i is the number of hours in the ith month. Also, remember that $KWBegin_i$ and $KWEnd_i$ are meter readings, so we can determine the total kilowatt-hours used for the whole year by subtracting the first meter reading ($KWBegin_1$) from the last ($KWEnd_{12}$).

Step 1: Set the value of i to 1

Step 2: Set the value of *AnnualCharge* to 0

Step 3: While $i \leq 12$ do

Step 4: Get the value of $KWBegin_i$ and $KWEnd_i$

Step 5: Set the value of M_i to $KWEnd_i - KWBegin_i$

Step 6: If $M_i \leq 1000$ then

Step 7: Set *AnnualCharge* to *AnnualCharge* + $(.06M_i)$

Step 8: Else

Step 9: Set *AnnualCharge* to *AnnualCharge* + (.06)1000

+ $(.08)(M_i - 1000)$

Step 10: Set the value of i to $i + 1$

End of While loop

Step 11: Print the value of *AnnualCharge*

Step 12: If $(KWEnd_{12} - KWBegin_1) < 500$, then

Step 13: Print the message 'Thank you for conserving electricity.'

11. Algorithm:

Step 1: Do

Step 2: Get the values of *HoursWorked* and *PayRate*

Step 3: If *HoursWorked* > 54 then

Step 4: *DT* = *HoursWorked* – 54

Step 5: *TH* = 14

Step 6: *Reg* = 40

Step 7: Else if *HoursWorked* > 40 then

Step 8: *DT* = 0

Step 9: *TH* = *HoursWorked* – 40

Step 10: *Reg* = 40

Step 11: Else (*HoursWorked* ≤ 40)

Step 12: *DT* = 0

Step 13: *TH* = 0

Step 14: *Reg* = *HoursWorked*

Step 15: *GrossPay* = *PayRate* × *Reg*

+ 1.5 × *PayRate* × *TH* + 2 × *PayRate* × *DT*

Step 16: Print the value of *GrossPay*

Step 17: Print the message 'Do you wish to do another computation?'

Step 18: Get the value of *Again*

While (*Again* = yes)

12. Steps 1, 2, 5, 6, 7, and 9 are sequential operations and steps 4 and 8 are conditional operations. After their completion, the algorithm moves on to the step below it, so none of these could cause an infinite loop. Step 3, however, is a while loop, and it could possibly cause an infinite loop. The true/false looping condition is "*Found* = NO and $i \leq 10{,}000$." If NUMBER is ever found in the loop then *Found* gets set to YES, the loop stops, and the algorithm ends after executing steps 8 and 9. If NUMBER is never found, then 1 is added to *i* at each iteration of the loop. Since step 2 initializes *i* to 1, *i* will become 10,001 after the $10{,}000^{th}$ iteration of the loop. At this point the loop will halt, steps 8 and 9 will be executed, and the algorithm will end.

13. Algorithm:

Step 1: Get values for *NUMBER*, $T_1, \ldots . T_{10000}$, and $N_1, \ldots ., N_{10000}$

Step 2: Set the value of *i* to 1 and set the value of *NumberFound* to 0

Answers to Practice Problems

Chapter 2

Section 2.2.2

1.

Step	*Operation*
1	Get values for *x*, *y*, and *z*
2	Set the value of *average* to $(x + y + z)/3$
3	Print the value of *average*
4	Stop

2.

Step	*Operation*
1	Get a value for *r*, the radius of the circle
2	Set the value of *circumference* to $2 \times \pi \times r$
3	Set the value of *area* to $\pi \times r^2$
4	Print the values of *circumference* and *area*
5	Stop

3.

Step	*Operation*
1	Get values for *amount*, the amount of electricity used, and for *cost*, the cost per kilowatt-hour
2	Set the value of *subtotal* to *amount* × *cost*
3	Set the value of *tax* to 0.08 × *subtotal*
4	Set the value of *total* to *subtotal* + *tax*
5	Print the value of *total*
6	Stop

4.

Step	*Operation*
1	Get values for *balance*, the current credit card balance, for *purchases*, the total dollar amount of new purchases, and for *payment*, the total dollar amount of all payments
2	Set the value of *unpaid* to *balance* + *purchases* − *payment*
3	Set the value of *interest* to *unpaid* × 0.12
4	Set the value of *newbalance* to *unpaid* + *interest*
5	Print the value of *newbalance*
6	Stop

5.

Step	*Operation*
1	Get values for *length* and *width* in feet
2	Set the value of *length-in-yards* to *length*/3

3	Set the value of *width-in-yards* to *width*/3
4	Set the value of *area* to *length-in-yards* × *width-in-yards*
5	Set the value of *cost* to *area* × 23
6	Print the value of *cost*
7	Stop

6.

Step	***Operation***
1	Get values for *first*, *second*, and *third*
2	Set the value of *points earned* to (5 × *first*) + (3 × *second*) + (1 × *third*)
3	Print the value of *points earned*
4	Stop

Section 2.2.3

1. If $x \geq 0$ then

    Set the value of *y* to 1

Else

    Set the value of *y* to 2

2. Get values for *x*, *y*, and *z*

If $x > 0$ then

    Set the value of *average* to (*x* + *y* + *z*)/3

    Print the value of *average*

Else

    Print the message 'Bad Data'

Stop

3. Get values for *balance*, *purchases*, *payment*

Set the value of *unpaid* to *balance* + *purchases* − *payment*

If *unpaid* < 100 then

    Set the value of *interest* to *unpaid* × 0.08

Else

    If *unpaid* ≤ 500 then

        Set the value of *interest* to *unpaid* × 0.12

    Else

        Set the value of *interest* to *unpaid* × 0.16

Set the value of *newbalance* to *unpaid* + *interest*

Print the value of *newbalance*

Stop

4. Get a value for x

While $x \neq 999$ do

Set the value of a to x^2

Set the value of b to sin(x)

Set the value of c to $1/x$

Print the values of a, b, and c

Get a value for x

End of the loop

Stop

5. Get values for *length*, *width*, *price*

Set the value of *length-in-yards* to *length*/3

Set the value of *width-in-yards* to *width*/3

Set the value of *area* to *length-in-yards* × *width-in-yards*

Set the value of *cost* to *area* × *price*

If *cost* ≤ 500 then

Print the message 'You can afford this carpet'

Else

Print the message 'This carpet is too expensive'

Stop

6. Add the following statement to the end of the solution, just before the Stop statement:

(Assume that we have stored the cost of the carpet in the variable *cost*.)

If (*cost* ≤ 250) then

Print the message 'This is a particularly good deal.'

7. Get a value for x

While ($x \neq 999$) do

Set the value of a to x^2

Print the value of a

Set the value of b to sin(x)

Print the value of b

If $x = 0$ then

Print the error message 'Cannot compute 1/x'

Else

Set the value of *c* to 1/*x*

Print the value of *c*

Get a value for *x*

End of the loop

Stop

Section 2.3.1

1. Initial values
   $a = 2$ $b = 4$ *count* $= 0$ *product* $= 0$
   After pass 1
   $a = 2$ $b = 4$ *count* $= 1$ *product* $= 2$
   After pass 2
   $a = 2$ $b = 4$ *count* $= 2$ *product* $= 4$
   After pass 3
   $a = 2$ $b = 4$ *count* $= 3$ *product* $= 6$
   After pass 4
   $a = 2$ $b = 4$ *count* $= 4$ *product* $= 8$

2. Yes, the algorithm still works correctly. On Line 1, we input the two values $a = 0$, $b = 0$. On Line 2, the Boolean expression is true, so we set *product* $= 0$ and skip the entire else clause. The next line executed is "print the value of product" which outputs a 0, the correct answer to the problem 0×0.

3. case 1 ($a = -2$, $b = 4$):
   The value of *product* will be -8, which is correct.

   case 2 ($a = 2$, $b = -4$):
   The value of *product* will be 0 (the while loop does not execute at all), which is incorrect.

4. The original algorithm fails when $b < 0$ because *count* is never less than *b*. If $b < 0$, change the value of *b* to $-b$, but set the value of a variable called *bnegative* to YES to remember that *b* was negative. After the product is computed, the sign of *product* will be incorrect if *b* was negative, so change the sign. Here is a pseudocode version that works for all integer values of *a* and *b*:

   Get values for *a* and *b*

   Set the value of *bnegative* to NO

   If (either *a* = 0 or *b* = 0) then

      Set the value of *product to* 0

   Else

If $b < 0$ then
  Set the value of b to $-b$
  Set the value of *bnegative* to YES
Set the value of *count* to 0
Set the value of *product* to 0
While (*count* $<$ b) do
  Set the value of *product* to (*product* + a)
  Set the value of *count* to (*count* + 1)
End of the loop
If (*bnegative* = YES) then
  Print the value of $-$*product*
Else
  Print the value of *product*
Stop

5. It is a highly inefficient algorithm because of the number of steps required to find the answer. For example, to add 234 + 567, we will have to repeat the addition of the value 234 to a running sum 567 separate times. When we do multiplication the "traditional" way that we learned in grade school, we have to carry out far fewer operations. (We will learn much more about how to evaluate the efficiency of algorithms in Chapter 3.)

6. Get values for a and b
If (either $a > 10{,}000$ or $b > 10{,}000$) then
  Print 'Please use a more efficient multiplication algorithm'
Else
  If (either $a = 0$ or $b = 0$) then
    Set the value of *product* to 0
  Else
    Set the value of *count* to 0
    Set the value of *product* to 0
    While (*count* $<$ b) do
      Set the value of *product* to (*product* + a)
      Set the value of *count* to (*count* + 1)
    End of the loop
  Print the value of *product*
Stop

Section 2.3.3

1. We would need to change Line 1 so that the algorithm would input 1 million numbers and names rather than 10,000. We would need to change Line 3 so that the loop would repeat a maximum of 1 million times rather than a maximum of 10,000. Actually, that is not a lot of change given that the problem is 100 times bigger. We only needed to change two lines. In fact, if the phone book were 1 billion numbers in length, then we would only need to change the same two lines.

2.

Step	*Operation*
1	Get values for *NUMBER*, T_1, … , $T_{10,000}$ and N_1, … , $N_{10,000}$
2	Set the value of *i* to 1 and the value of *Found* to NO
3	Do
4	If *NUMBER* is equal to the *i*th number on the list, T_i , then
5	Print the name of the corresponding person, N_i
6	Set the value of *Found* to YES
	Else
7	Add 1 to the value of *i*
8	While (*Found* = NO) and ($i \leq 10{,}000$)
9	If (*Found* = NO) then
10	Print the message 'Sorry, this number is not in the directory'
11	Stop

3. You must change the operation on Line 7 from a greater-than (>) to a less-than (<) sign. That line will now read as follows:

   If $A_i <$ *largest so far then* ...

   That is the only required change. However, to avoid confusion about what the algorithm is doing, you probably should also change the name of the variable *largest so far* to something like *smallest so far* on Lines 3, 7, 8, and 12. Otherwise, a casual reading of the algorithm might lead someone to think incorrectly that it is still an algorithm to find the largest value rather than the smallest.

4. If $n = 0$ (the list is empty), then there are no values for A_1, A_2, ..., A_n. In particular, setting the value of *largest so far* to A_1 gives a meaningless value to *largest so far*. The while loop will not execute at all because *i* has the value 2 and *n* has the value 0, so the condition $i \leq n$ is false. The algorithm will print out nonsense values for *largest so far* and *location*.

   The algorithm can be fixed by putting a conditional statement after Line 1. If $n = 0$, then the algorithm should print a message that says the list is empty. The "else" case will be the rest of the current algorithm.