CS 101 Lab #4 Structured Program Development (Part 2)

Mythili Vutukuru

IIT Bombay

Reference: "C How to Program", Deitel and Deitel, 8th Edition, Chapter 4

GOAL OF LAB #4

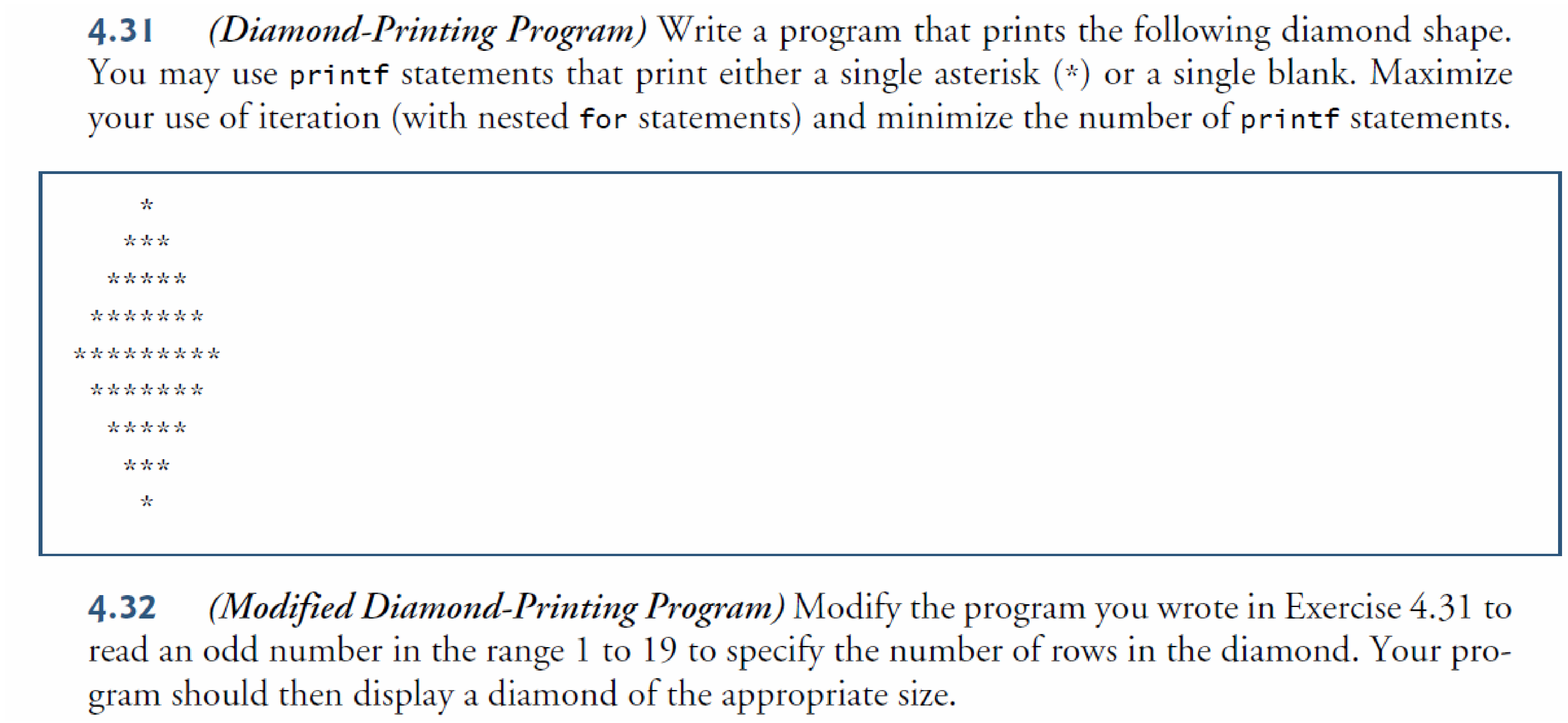
- In this lab, you will write programs to help you understand the following concepts:
  - Practice iteration using for loop
  - Practice programs with switch, break, continue
  - Recap logical operators
  - More practice for structured programming, using all concepts from conditionals, loops, etc.
- Submit any two programs testing any two different concepts
- Please write code neatly, with proper indentations and comments

PATTERNS WITH FOR LOOPS

4.16 *(Triangle-Printing Program)* Write a program that prints the following patterns separately, one below the other. Use `for` loops to generate the patterns. All asterisks (*) should be printed by a single `printf` statement of the form `printf("%s", "*");` (this causes the asterisks to print side by side). [*Hint:* The last two patterns require that each line begin with an appropriate number of blanks.]

```
(A)           (B)           (C)            (D)
*             **********     **********             *
**            *********       *********            **
***           ********         ********           ***
****          *******           *******          ****
*****         ******             ******         *****
******        *****               *****        ******
*******       ****                 ****       *******
********      ***                   ***      ********
*********     **                     **     *********
**********    *                       *    **********
```

PATTERNS WITH FOR LOOPS

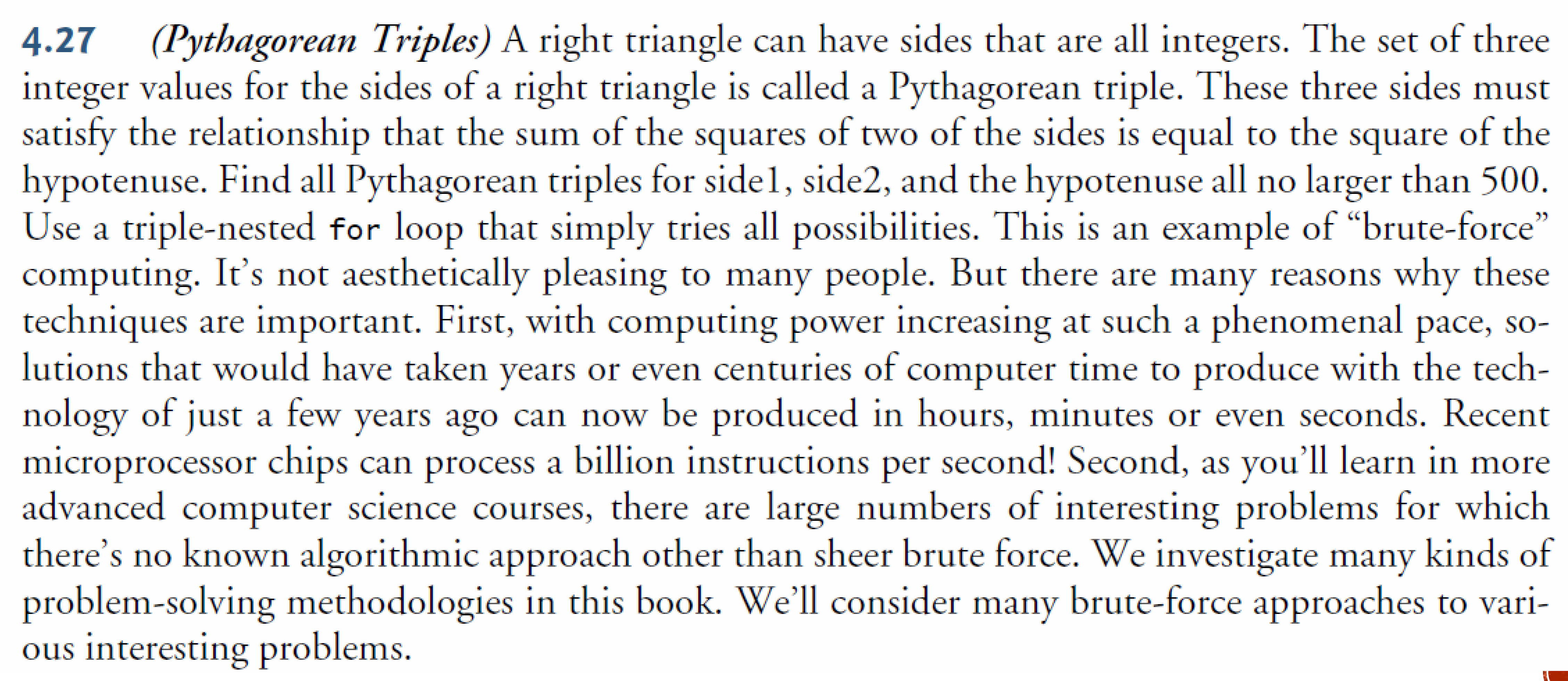


4.31 *(Diamond-Printing Program)* Write a program that prints the following diamond shape. You may use `printf` statements that print either a single asterisk (*) or a single blank. Maximize your use of iteration (with nested `for` statements) and minimize the number of `printf` statements.

```
    *
   ***
  *****
 *******
*********
 *******
  *****
   ***
    *
```

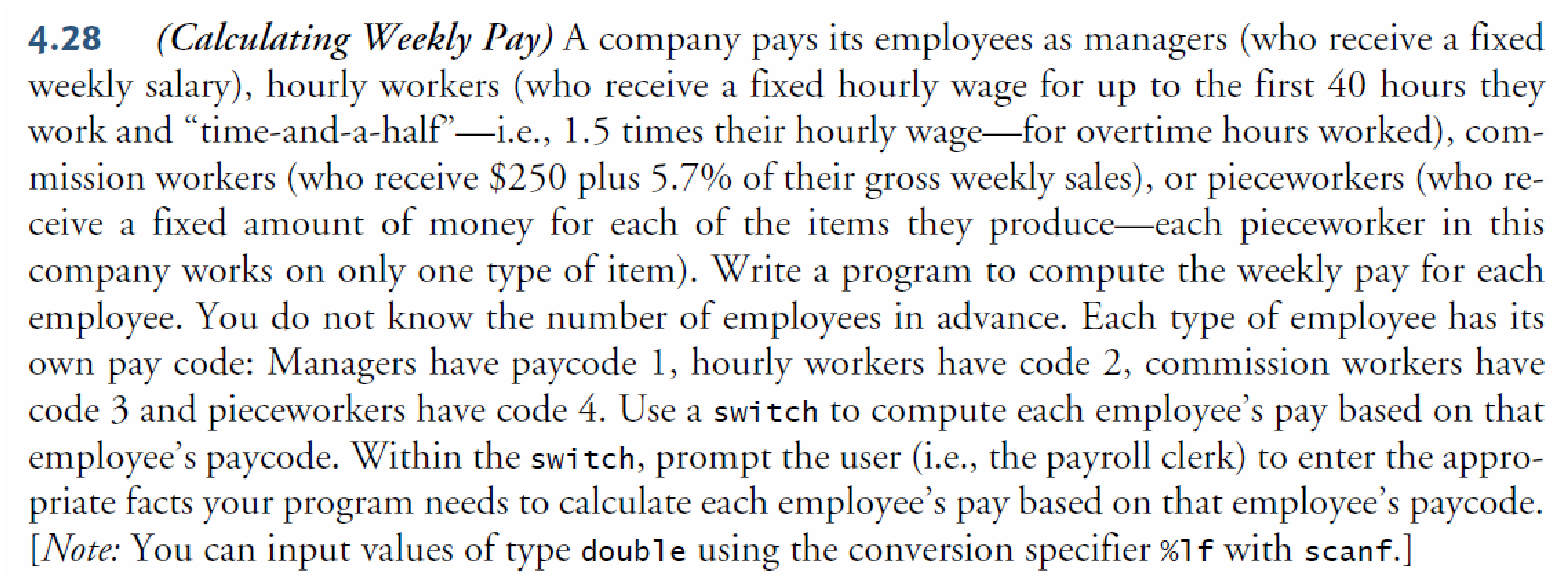
4.32 *(Modified Diamond-Printing Program)* Modify the program you wrote in Exercise 4.31 to read an odd number in the range 1 to 19 to specify the number of rows in the diamond. Your program should then display a diamond of the appropriate size.

NESTED FOR LOOPS



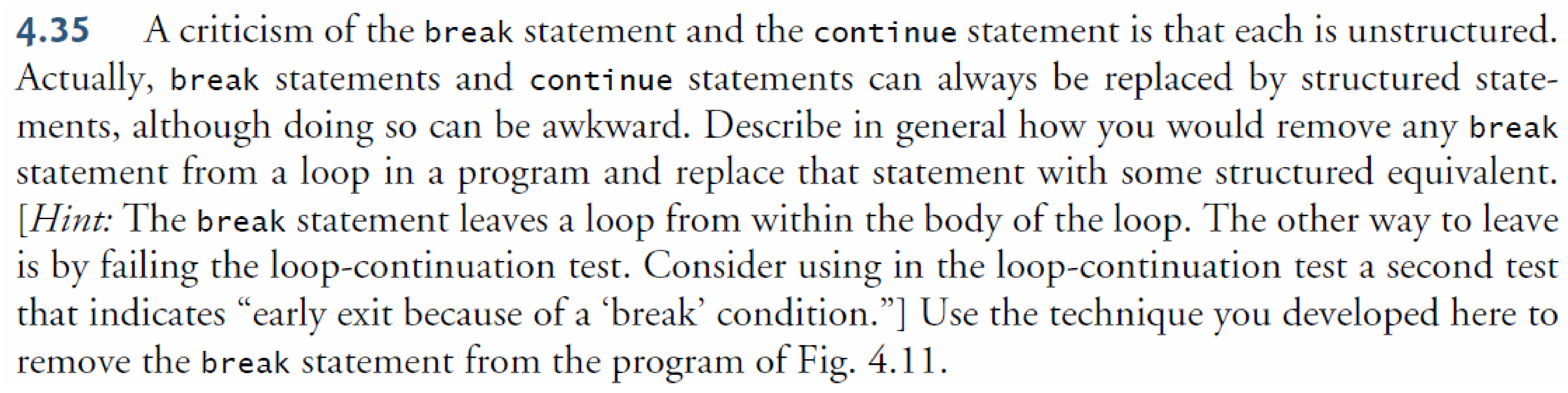
4.27 *(Pythagorean Triples)* A right triangle can have sides that are all integers. The set of three integer values for the sides of a right triangle is called a Pythagorean triple. These three sides must satisfy the relationship that the sum of the squares of two of the sides is equal to the square of the hypotenuse. Find all Pythagorean triples for side1, side2, and the hypotenuse all no larger than 500. Use a triple-nested `for` loop that simply tries all possibilities. This is an example of “brute-force” computing. It’s not aesthetically pleasing to many people. But there are many reasons why these techniques are important. First, with computing power increasing at such a phenomenal pace, solutions that would have taken years or even centuries of computer time to produce with the technology of just a few years ago can now be produced in hours, minutes or even seconds. Recent microprocessor chips can process a billion instructions per second! Second, as you’ll learn in more advanced computer science courses, there are large numbers of interesting problems for which there’s no known algorithmic approach other than sheer brute force. We investigate many kinds of problem-solving methodologies in this book. We’ll consider many brute-force approaches to various interesting problems.

SWITCH STATEMENT

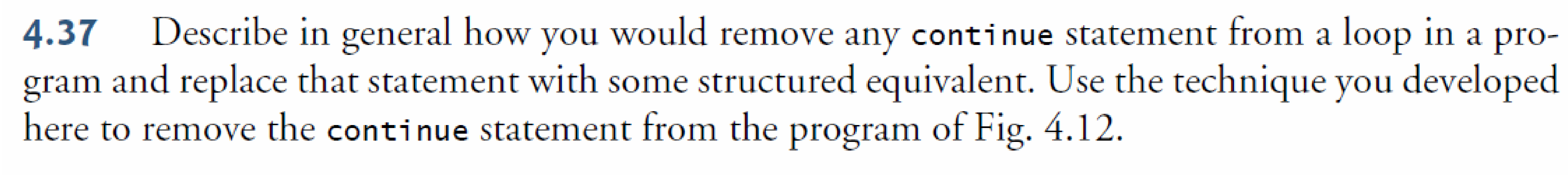


4.28 *(Calculating Weekly Pay)* A company pays its employees as managers (who receive a fixed weekly salary), hourly workers (who receive a fixed hourly wage for up to the first 40 hours they work and "time-and-a-half"—i.e., 1.5 times their hourly wage—for overtime hours worked), commission workers (who receive $250 plus 5.7% of their gross weekly sales), or pieceworkers (who receive a fixed amount of money for each of the items they produce—each pieceworker in this company works on only one type of item). Write a program to compute the weekly pay for each employee. You do not know the number of employees in advance. Each type of employee has its own pay code: Managers have paycode 1, hourly workers have code 2, commission workers have code 3 and pieceworkers have code 4. Use a `switch` to compute each employee's pay based on that employee's paycode. Within the `switch`, prompt the user (i.e., the payroll clerk) to enter the appropriate facts your program needs to calculate each employee's pay based on that employee's paycode. [*Note:* You can input values of type `double` using the conversion specifier `%lf` with `scanf`.]

BREAK AND CONTINUE

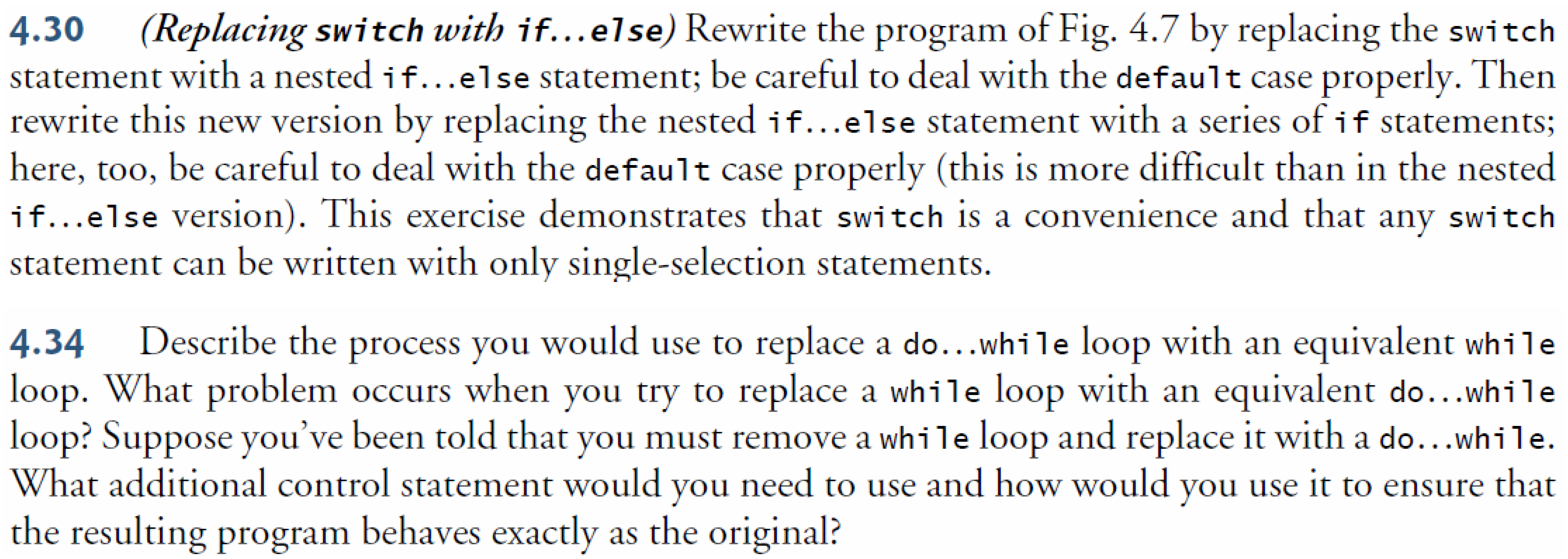


4.35 A criticism of the `break` statement and the `continue` statement is that each is unstructured. Actually, `break` statements and `continue` statements can always be replaced by structured statements, although doing so can be awkward. Describe in general how you would remove any `break` statement from a loop in a program and replace that statement with some structured equivalent. [*Hint:* The `break` statement leaves a loop from within the body of the loop. The other way to leave is by failing the loop-continuation test. Consider using in the loop-continuation test a second test that indicates "early exit because of a 'break' condition."] Use the technique you developed here to remove the `break` statement from the program of Fig. 4.11.



4.37 Describe in general how you would remove any `continue` statement from a loop in a program and replace that statement with some structured equivalent. Use the technique you developed here to remove the `continue` statement from the program of Fig. 4.12.

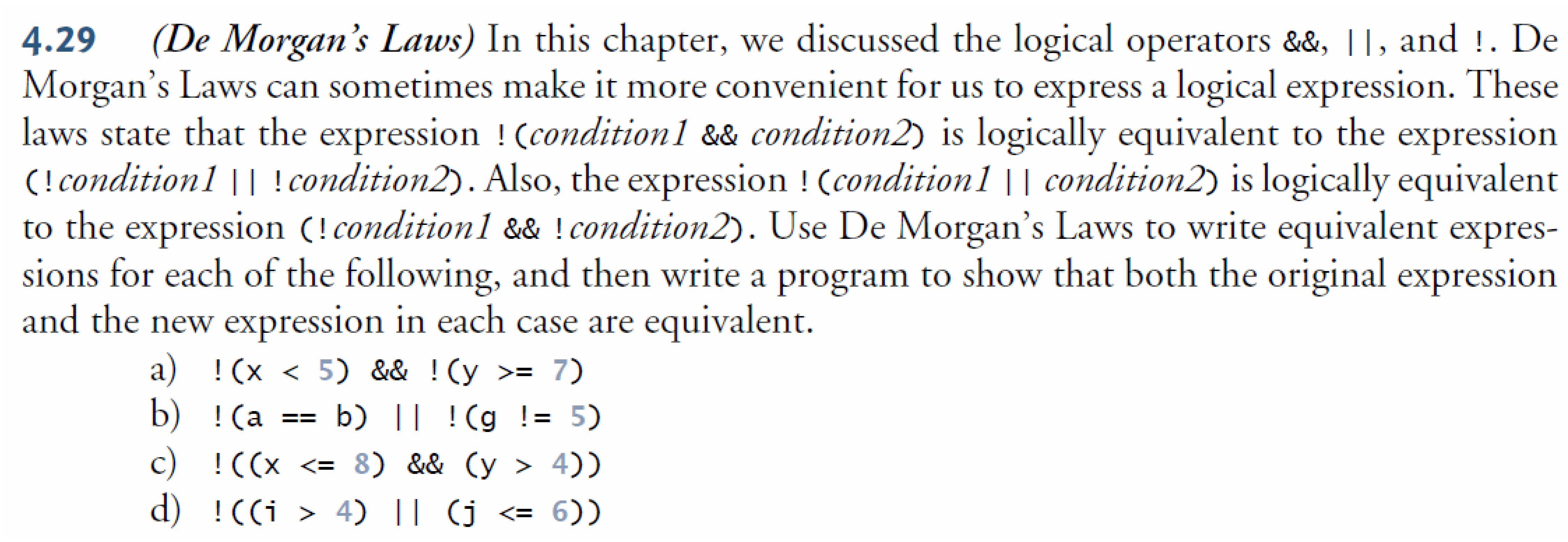
DO-WHILE AND SWITCH



4.30 *(Replacing* **`switch`** *with* **`if...else`***)* Rewrite the program of Fig. 4.7 by replacing the `switch` statement with a nested `if...else` statement; be careful to deal with the `default` case properly. Then rewrite this new version by replacing the nested `if...else` statement with a series of `if` statements; here, too, be careful to deal with the `default` case properly (this is more difficult than in the nested `if...else` version). This exercise demonstrates that `switch` is a convenience and that any `switch` statement can be written with only single-selection statements.

4.34 Describe the process you would use to replace a `do...while` loop with an equivalent `while` loop. What problem occurs when you try to replace a `while` loop with an equivalent `do...while` loop? Suppose you've been told that you must remove a `while` loop and replace it with a `do...while`. What additional control statement would you need to use and how would you use it to ensure that the resulting program behaves exactly as the original?

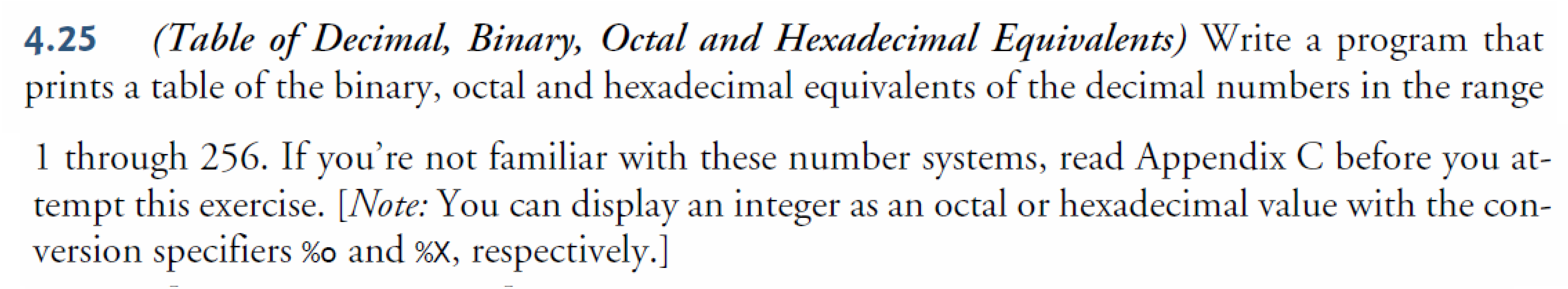
DE MORGAN'S LAWS



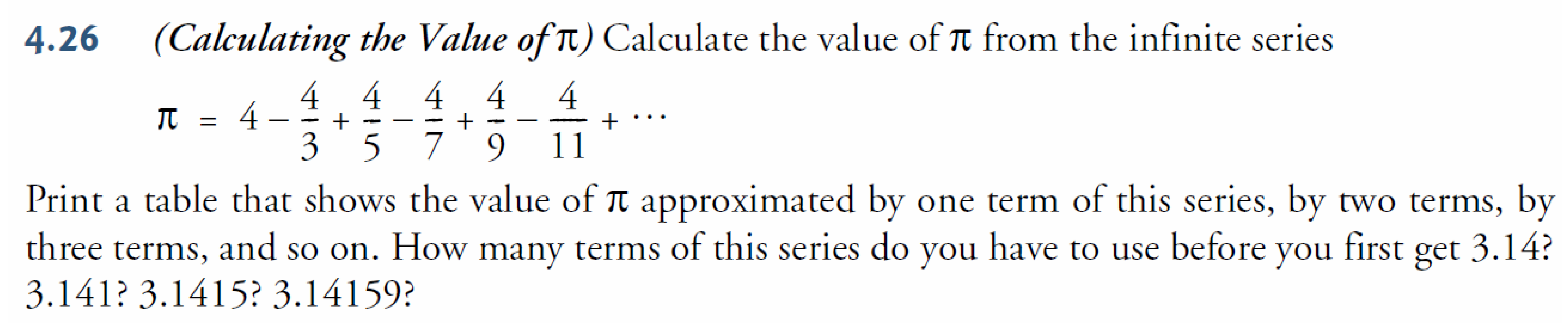
4.29 *(De Morgan's Laws)* In this chapter, we discussed the logical operators `&&`, `||`, and `!`. De Morgan's Laws can sometimes make it more convenient for us to express a logical expression. These laws state that the expression `!(`*condition1* `&&` *condition2*`)` is logically equivalent to the expression `(!`*condition1* `||` `!`*condition2*`)`. Also, the expression `!(`*condition1* `||` *condition2*`)` is logically equivalent to the expression `(!`*condition1* `&&` `!`*condition2*`)`. Use De Morgan's Laws to write equivalent expressions for each of the following, and then write a program to show that both the original expression and the new expression in each case are equivalent.

a) `!(x < 5) && !(y >= 7)`
b) `!(a == b) || !(g != 5)`
c) `!((x <= 8) && (y > 4))`
d) `!((i > 4) || (j <= 6))`

MORE STRUCTURED PROGRAMMING



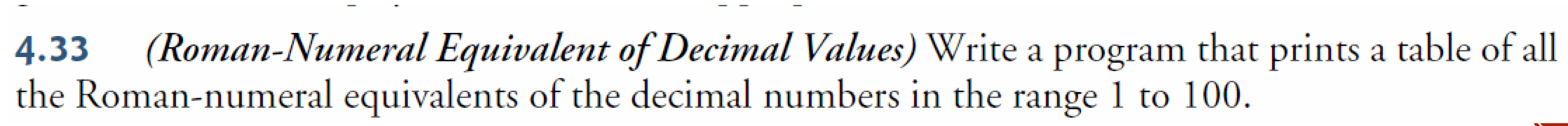
4.25 *(Table of Decimal, Binary, Octal and Hexadecimal Equivalents)* Write a program that prints a table of the binary, octal and hexadecimal equivalents of the decimal numbers in the range 1 through 256. If you're not familiar with these number systems, read Appendix C before you attempt this exercise. [*Note:* You can display an integer as an octal or hexadecimal value with the conversion specifiers `%o` and `%X`, respectively.]



4.26 *(Calculating the Value of π)* Calculate the value of π from the infinite series

$$\pi = 4 - \frac{4}{3} + \frac{4}{5} - \frac{4}{7} + \frac{4}{9} - \frac{4}{11} + \cdots$$

Print a table that shows the value of π approximated by one term of this series, by two terms, by three terms, and so on. How many terms of this series do you have to use before you first get 3.14? 3.141? 3.1415? 3.14159?



4.33 *(Roman-Numeral Equivalent of Decimal Values)* Write a program that prints a table of all the Roman-numeral equivalents of the decimal numbers in the range 1 to 100.

MORE STRUCTURED PROGRAMMING

- Write a program to print out all the unique prime factors of a number. You must print out each prime factor only one. For example, if your input number is 360, you must print out the numbers 2 3 5.
- Write a program to print out the complete prime factorization of a number. That is, you must list out all the prime factors as many times as they appear in the prime factorization. For example, if your input is 360, you must output 2 2 2 3 3 5