```
(number = 5, number² < 50 is true, Loop continues)    5
(number = 6, number² < 50 is true, Loop continues)    6
(number = 7, number² < 50 is true, Loop continues)    7
(number = 8, number² < 50 is false, Loop exits)
```

g. Question:

```
int j = -10
while (!(j > 10))
{
    System.out.println (j);
    j += 3;
} // while
```

Answer:

```
(j = -10, !(j > 10) is true, Loop continues)      -10
(j = -7, !(j > 10) is true, Loop continues)       -7
(j = -4, !(j > 10) is true, Loop continues)       -4
(j = -1, !(j > 10) is true, Loop continues)       -1
(j = 2, !(j > 10) is true, Loop continues)        2
(j = 5, !(j > 10) is true, Loop continues)        5
(j = 8, !(j > 10) is true, Loop continues)        8
(j = 11, !(j > 10) is false, Loop exits)
```

8.11 Exercises

1. Create a program called `CubeRoots` that outputs a table of numbers and their cube roots from 10 to 50. The table should line up and the cube roots should be output to 4 decimal places. [8.3.1]

2. Create a program called `SumSequence` that obtains a number from the user and then outputs the sum of 1 through the number (1 + 2 + 3 + … + number) using a **`for`** loop. [8.3.1]

3. Create a program called `Factorial` that obtains a number from the user and then outputs the product of 1 through the number (1 * 2 * 3 * … * number) using a **`for`** loop. What happens if the result is larger than can be held in an **`int`**. [8.3.1]

4. Create a program called `Fibonacci` that obtains a number `n` from the user and then outputs the first `n` numbers in the Fibonacci numbers. The Fibonacci sequence starts with 1 and 1. Each additional element in the sequence is derived by adding the previous two elements in the sequence. Here are the first 10 numbers. [8.3.1]

```
1 1 2 3 5 8 13 21 34 55
```

5. Create a program called *Triangle* that creates a triangle made of asterisks using a loop. [8.3.1]

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

6. Create a program called *Diamond* that obtains creates a diamond made of asterisks using two loops. [8.3.1]

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

7. Create a program called *BetterBox* that uses the *hsa.PaintBug* class. The program should draw a box using a **for** loop. [8.3.1]

8. Create a program called *SpiralLoop* that uses the *hsa.PaintBug* class. The program should draw a square spiral using a **for** loop. Each edge it draws should be slightly shorter than the one before it. [8.3.1]

9. Create a program called *ManyV* that uses the *hsa.PaintBug* class. The program starts a *PaintBug* object near the upper-left corner of the window and draws 5 large 'V's, connected to each other, making a back and forth pattern. [8.3.1]

10. Create a program called *SquareTable* that outputs numbers and their squares from 1 to 80 in a table as follows. [8.3.1]

```
Num    Square     Num    Square     Num    Square     Num    Square
  1         1      21       441      41      1681      61      3721
  2         4      22       484      42      1764      62      3864
...
 20       400      40      1600      60      3600      80      6400
```

11. Create a program called *By3* that outputs multiples of 3s from 30 to 60 using a **for** loop. [8.3.3]

12. Create a program called *PopulationGrowth* that calculates how many years it will be until the population of a city exceeds 1,000,000 people. The current population of the city is 150,000 and it is growing at 5% a year. [8.4]

13. Create a program called *CountDown* that obtains a number from the user and counts backwards from 100 by 5s, outputting "I stopped." when the count would be less than the number specified by the user. The numbers should line up. [8.4]

```
What number do I stop at? 82
100
 95
 90
 85
I stopped.
```

14. Create a program called *Point* that simulates the playing of a simple game. Roll a single die to get a value from 1 to 6. This is called your *point*. Now keep rolling until you roll the same value again (your point). Output what the point was, the list of rolls that were made until the point was rolled, and a count of how many times the die was rolled until the point was rolled. [8.4]

15. Create a program called *SumDigits* that obtains an integer from the user and then sums the digits. The program should use the remainder and integer division operators. [8.4]

16. Create a program called *CalendarMonth* that obtains the day of the week that a month starts in and the number of days in the month. It then outputs a well-formatted calendar for the month. [8.4]

```
Day the month starts on (1 for Sunday): 3
Number of days in month: 30

Sun Mon Tue Wed Thu Fri Sat
          1   2   3   4   5
  6   7   8   9  10  11  12
 13  14  15  16  17  18  19
 20  21  22  23  24  25  26
 27  28  29  30
```

17. Create a program called *Factors* that obtains an integer from the user and outputs all of its divisors on one line. [8.4]

```
Enter a number: 12
1 2 3 4 6 12
```

18. Create a program called *PerfectNumbers* that outputs the first three perfect numbers. A perfect number is a number that is equal to the sum of all of its divisors (excluding itself). For example 6 is a perfect number because 6 = 1 + 2 + 3. [8.4]

19. Create a program called *simpleCalc* that reads in a simple expression in the form

```
[number] [operator] [number]
```

and outputs the result. The program should halt if both numbers are 0, otherwise it should output the result. The numbers are floating-point (use the *Stdin.readDouble* method to read them). The operator is either +, -, * or / (use