HOMEWORK PROBLEMS 4

1) Which of the `if` statements below is legal? Run a test program to check your answers.

```
if (count <= 10)
{
}
else
   System.out.println("hello");


if (count <= 10)

else
   System.out.println("hello");


if (count <= 10)
   ;
else
   System.out.println("hello");
```

A semicolon by itself is the **null statement**. It is a legal statement in Java. When executed, it does nothing. Braces that do not contain any statements act like a null statement.

2) Write a multi-branch `if` statement that displays the English word for the integer value in the variable `x` for any value between 0 and 5. For other integer values, your `if` statement should display `other`. For example, if `x` is `2`, then your `if` statement should display `two`. If `x` is `-20`, your `if` statement should display `other`.

3) Write the statement that reads in an integer from the keyboard using the `Scanner` object pointed to by `p`, and assigns the value read in to an `int` variable `x`.

4) Write a `while` loop that computes the sum of the odd integers from 1 to 999 inclusive. Increment the variable that is acting as a counter by 2 each time the loop body is executed.

5) List the first 10 operations that occur when the code in Fig. 4.7 is executed. Do the same for the code in Fig. 4.9. How do the two sequences compare?

6) Run the following program (it contains an infinite loop):

```
class C4h6
{
   public static void main(String[] args)
   {
      int x = 1;
      while (x == 1)
         System.out.println("Infinite loop");
   }
}
```

Determine the keyboard sequence for your computer system that will end the infinite loop. Try ctrl-c, ctrl-d, ctrl-z, and ctrl-break. You can also terminate the program by closing the terminal/command-mode session running the program.

7) What is the effect of the statement below. Try it for values of `x` equal to 0 to 10. `n` has type `int`.

```
switch(n)    // n is the controlling expression
{
   case 0:
   case 1:
   case 7:      System.out.println("dog");
                break;
   case 3:      System.out.println("cat");
   case 4:      System.out.println("bird"):
                break;
   case 6:      System.out.println("elephant");
                break;
   default:     System.out.println("default");
}
```

The `break` statement has two uses: to break out of a loop and to break out of a `switch` statement. The constants used for each case should have an integer type or a type that is represented with integers (for example, `char`). The type of the controlling expression should match the type of the case constants. The `default` case is optional. The controlling expression can be an expression. For example, we could have used `n + 1` in place of `n` in the example above.

Remove the `break` statements in the code above. With this change, what is displayed for each value of `n` from 0 to 10. In what sequence are the instructions executed without the `break` statements?

8) Test the statement below in a program. Try it for `y` equals `1` and `z` equals `2`. Also try it for `y` equals `10` and `z` equals `8`. What is its effect?

```
x =      y < z ? 10 : 20;
         ---------------
                ↑
```

this is a **conditional expression**

The `?:` pair is called the **conditional operator**. The conditional operator has three operands:

operand1 `?` *operand2* `:` *operand3*

operand1 is a true/false expression. If *operand1* is `true`, the value of the entire expression is given by *operand2*. If *operand1* is `false`, the value of the entire expression is given by *operand3*. *operand2* and *operand3* can be expressions. In the example above, `y < z`, `10`, and `20` are *operand1*, *operand2*, and *operand3*, respectively. If `y < z` is `true`, `10` is assigned to `x`. Otherwise, `20` is assigned to `x.`

9) Write a program that reads in two integers and displays their sum. *Hint*: Invoke `nextInt` twice.

10) Write a program that displays all the integers from 1 to 1000 that are not evenly divisible by 13. *Hint*: `x` is not evenly divisible by 13 if the expression `x%13 != 0` is `true`. Recall that `%` is the remainder operator.

11) Run the following program:

```
class C4h11
{
   public static void main(String[] args)
   {
      int i = 1;
      while (i < 5) ;    // put semicolon after parentheses
      {
         System.out.println(i);
         i++;            // adds 1 to i
      }
   }
}
```

What happens? A lone semicolon is a valid statement in Java. It is called the **null statement**. A null statement does nothing when executed. The semicolon after the right parenthesis in the `while` statement functions as the *entire* body of the `while` statement. Now remove this semicolon, compile and run. What happens?

12) Write a program that computes the value of 7 factorial. 7 factorial is equal to 7× 6 × 5× ... × 1. Initialize a variable named `factorial` to 7. Then using a loop, multiply `factorial` by the integers 6 down to 2. Use type `long` for your computations. What is the largest factorial a variable of type `long` can hold?

13) What does the program below display? Run it to check your answer.

```
class C4h13
{
   public static void main(String[] args)
   {
      int i, j;
      i = 1;
      while (i < 10)
      {
         j = 1;
         while (j < 5)
         {
            System.out.print(i + "," + j + "  ");
            j++;
         }
         System.out.println();     // go to next line
         i++;
      }
   }
}
```

This program has a loop within a loop. This structure is called a **nested loop**.

14) Suppose you are performing calculations on a calculator that displays only three digits. If you enter `12.3`, `+`, `0.04`, and `=` , the calculator will add `12.3` and `0.04`. But because the calculator maintains only three digits, it gets the result `12.3` (which is `12.34` truncated to three digits). *Adding 0.04 has no effect*. Now suppose you enter `0.01`, `+`, `0.04`, and `=`. The calculator in this case displays `0.05`, the correct answer. The problem in the first case is **roundoff error** that results because of the finite precision with which numbers are represented within a calculator. This error can be significant if the difference in value of the numbers to be added is large. In the extreme case, adding the smaller number to the larger number has no effect on the larger number, as illustrated

by our first case above. An important observation: Computations on a calculator or a computer do not necessarily yield the correct answers, *even if the correct operations are performed.*

In the following program, the two loops perform the same computation but in a different order. Why do they yield different answers? If so, which one yields the better answer?

```
class C4h14
{
   public static void main(String[] args)
   {
      float sum1 = 0.0f, sum2 = 0.0f;
      int i;

      i = 1;
      while (i <= 100)   // sum from large to small
      {
          sum1 = sum1 + 1.0f/i;
          i++;
      }
      System.out.println("sum1 = " + sum1);

      i = 100;
      while (i >= 1)    // sum from small to large
      {
          sum2 = sum2 + 1.0f/i;
          i--;
      }
      System.out.println("sum2 = " + sum2);
   }
}
```

15) What is the effect of the `continue` statement in the following loop:

```
int i = 1;
while (i <= 10)
{
   System.out.println("before continue i = " + i);
   if (i%3 == 0) continue; // execute continue when i multiple of 3
   System.out.println("after continue i = " + i);
}
```

16) Write a program that displays the cubes of all the integers that are greater than `n` but less than `m`, where `n` and `m` are values entered via the keyboard. Test your program for `n` = 10, `m` = 10000.

17) Write a program that repeatedly reads in an integer and displays its square. Your program should terminate only if the user enters 0.

18) Write a program that computes and displays the value of

```
0.1 + (0.1)² + (0.1)³ + ... + (0.1)ⁿ
```

Your program should prompt for and read in from the keyboard the value of `n`. For this program, you should use a variable `sum` to accumulate the sum. Also use a variable `term`. Initialize `term` to 0.1 (which is the first term in the summation). Each time through the loop, execute

```
sum = sum + term;
```

Follow this statement with a statement that changes the value in the variable `term` to the value of the next term in the summation. Get the next term from the current term by multiplying the current term by 0.1. Use type `double` for `sum` and `term`. Test your program for `n` equals 1, 15, 50, and 100.

19) Write a program that computes the value of

```
1.0  + 1.0/(1!) + 1.0/(2!) + 1.0/(3!) + ... + 1.0/(n!)
```

Your program should prompt for and read in from the keyboard the value of `n`. Use the technique described in homework problem 18 (i.e., compute each term from the previous term). Also include the following statement in your program:

```
System.out.println("e = " + Math.E);
```

Use type `double` for your computations. Run your program for n = 1, 10, 20, 50, and 100.

20) Display all the values in the following sequence that are less than `max:`

```
1, 2×1, 3×2×1, 4×3×2×1, ...
```

Your program should prompt for and read in the value of `max`. Use a loop with an interior exit test constructed with an `if` statement and a `break` statement. Obtain each value to be displayed from the previous value displayed by multiplying the previous value by the appropriate factor. For example, get the second value by multiplying the first value by 2; get the third value by multiplying the second value by 3. In general, get the `i`th value by multiplying the `(i-1)`st value by `i`. Test your program with values for `max` equal to -1, 1, 2, 3, 10, 100, and 1000.

21) Same as homework problem 20 but use the structure illustrated in Fig. 4.8.

22) Same as homework problem but 20 use the structure illustrated in Fig. 4.9.

23) Write a program that displays

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
1 2 3 4 5 6
1 2 3 4 5 6 7
1 2 3 4 5 6 7 8
1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9 10
```

Use nested loops (i.e., a loop within a loop). The outer loop body should execute 10 times. Each time the inner loop is activated, it should display one line of numbers. *Hint*: Use the `print` statement to display each number. Use the `println` statement with no arguments to go to the next line.

24) Write a program reads in the number of terms to sum and then sums that number of terms in the series

    4 – 4/3 + 4/5 – 4/7 + 4/9 + …

    For example, if the user enters 100, then your program should sum the first 100 terms of the series. Use type `double` for all your computations. Run you program for the following inputs: 1, 10, 100, and 1000.
    Your program should display the computed value and the value of `Math.PI`.

25) Write a program that contains a loop that executes 20 times. Each time the loop body executes, it should compute the square root of `x` and assign the result back to `x`. Initialize `x` to `5.0`. Do the square roots seem to converge on a value. Use `Math.sqrt` (a predefined method in the `Math` class) to compute square roots. Run your program a second time using an initial value of `0.5` for `x`.

26) Redo homework problem 25 from Chapter 1, but use a loop.

27) Write a program that reads in a positive integer into `n`. Your program should then display `n` rows. Each row should have consecutive integers starting from 1 and have one more integer than the previous row. The first row should contain only 1. For example, if `n` is 3, then your program should display

```
1
1 2
1 2 3
```

    See homework problem 23.

28) Write a program that reads in a positive integer into `n`. It should then compute and display `n` factorial, when is the program of the integers from `n` down to 1.

29) Write a program that reads in a positive integer into `n`. Your program should then display the `n`th Fibonacci number. The first two Fibonacci numbers are 1 and 1. Thereafter, each number is the sum of the two numbers immediately preceding it. Thus, the sequence is 1, 1, 2, 3, 5, 8, 13, 21, …Run your program for `n` = 10, 15, and 20.

30) Write a program that reads in a positive integer into `n`. You program should determine and display the `n`th Fibonacci number divided by the (`n`-1)th Fibonacci number. Run your program for `n` = 10, 15, and 20.

31) Write a program that displays the integers 1 to 10 in 1-to-10 order, each on a separate line.

32) Write a program that displays the integers 10 down to 1 in 10 to 1 order, each on a separate line.

33) Write a program that reads in reads integers. When the sum is greater than or equal to 100, your program should display this fact and stop.

    Sample session:

```
Enter integer
50
Enter integer
40
Enter integer
9
```

```
Enter integer
5
Sum is now greater than or equal to 100
```

34) Write a program that prompts for and reads in integers. Your program should stop when a number is entered which is equal to the first number entered.

Sample session:

```
Enter integer
5
Enter integer
4
Enter integer
-30
Enter integer
5
Goodbye
```

35) Write a program that reads and sums positive integers. Your program should display this sum when a negative number is entered and then stop. Don't include the negative number in the sum.

Sample session

```
Enter integer
3
Enter integer
10
Enter integer
15
Enter integer
-1
Sum = 28
```

36) Write a program that reads in an integer grade. If the grade greater than or equal to 80, your program should display "`good`". Otherwise, your program should display "`bad`".

37) Write a program that prompts the user for a positive double value. Read it into the variable `x`. Set a variable `g` to 10.5. Then execute the following statement 50 times:

```
g = (g + x/g)*0.5
```

Finally, display the final value of `g`. What is the relationship between the value entered for `x` and the final value of `g`?

38) Write a program that prompts the user for a count. Then it reads that number of integers. Your program should display the largest and smallest integers among the numbers following the count. Test your program with the following input: 5, 7 -20, 100, -7, 0.

39) Write a program that prompts for and reads in a positive integer into a variable `n`. Your program should then sum the first `n` ODD integers and display the sum. For example, if 3 is entered for `n`, your program should display the sum 1 + 3 + 5. If 5 is entered, your program should display the sum 1 + 3 + 5 + 7 + 9. What is the relation between the number entered and the number displayed.

40) Is the following loop an infinite loop, or will a run-time error occur when `i` exceeds the range of an `int`?

```
int i = 0;
while (true)
   i = i + 1;
```

41) Write a program that reads in a file and then displays the number of occurrences of each vowel (a, e, i, o, or u). Your program should be case insensitive. For example, it should treat `a` and `A` as the same letter. Test your program with the file `gettingstarted.txt`. Your display should look like this:

```
a: 23
e: 37
.
.
.
```

where the counts for each letter are the counts for the file `gettingstarted.txt`.

42) Write a program that includes a method `wellFormed` that is passed a string consisting of left and right parentheses. Your `wellFormed` method should return true if the parentheses are well formed, and false otherwise. Test your method with following strings:

```
()
(())
()()
()(())
((()()()))
(
)
)(
(()
())
```