Java Problems

1. The Fibonacci Series Problem

Find the first term greater than 1000 in the sequence:

1 1 2 3 5 8 13 ...

Also find the sum of all the values up to that term.

2. The Greenfly Problem

Greenfly can reproduce asexually. After one week of life a lone female can produce eight offspring a day. Starting at the beginning of day 1 with a single mature female, how many greenfly could there be by the end of day 28? It may be assumed that:

- There are no deaths
- All offspring are females

Note that at the end of day 1 there will be 9 greenfly (original + 8 offspring). At the end of day 7 there will be 57 greenfly (original + 8×7 offspring). At the end of day 8 there will be 129 greenfly (original + 8×8 offspring + 64 offspring from the daughters produced on day 1).

3. All Prime Numbers less than 600

Write a program to print a table of all prime numbers less than 600. Use the sieve method; take the first 600 integers and cross out all those that are multiples of 2, 3, 5, etc. until only primes remain, then print out the table. The table must be organised so that there are ten prime numbers on each line. The start of the program should resemble the following:

```
public class Primes
 { private static final int SIZE=600, SQRTSIZE=25;

   public static void main(String[] args)
    { int[] primes = new int[SIZE];
      for (int i=2; i<SIZE; i++)
         primes[i] = 1;
      .
      .
```

4. A Sort Problem

Write a program which incorporates a method `sort` to 'sort by exchange' the elements of an array. An appropriate program to sort six numbers of type `int` into ascending order might, in outline, appear thus:

```
public class SortProg
 { public static void main(String[] args)
    { int[] vec = {5, 3, 2, 4, 6};
      sort(vec);
      for (int i=0; i<vec.length; i++)
         System.out.println(vec[i]);
    }

   private static void sort(int[] v)
    { .
      .
      .
    }
 }
```

The call `sort(vec)` should sort the elements `vec[0]`, `vec[1]`, ... `vec[5]` into ascending order.

Roughly speaking the algorithm should be as follows:

- The first pair of elements is dealt with.
- The 2nd then 1st pair of elements are dealt with.
- The 3rd then 2nd then 1st pair of elements are dealt with.
- The 5th, 4rd, ... 1st pair of elements are dealt with.

Here, 'dealing with' means exchanging the elements if they are the wrong way round and abandoning the entire stage if they are all right.

For six elements:	6 2 1 4 3 5
The first stage yields:	2 6 1 4 3 5
The second stage yields:	1 2 6 4 3 5
The third stage yields:	1 2 4 6 3 5
The fourth stage yields:	1 2 3 4 6 5
The fifth stage yields:	1 2 3 4 5 6

5. The Date of Easter Problem

A convenient algorithm for determining the date of Easter in a given year was devised in 1876 and first appeared in *Butcher's Ecclesiastical Handbook*. The algorithm is valid for all years in the Gregorian calendar. Subject to minor adaptations, the algorithm is as follows:

1. Let `y` be the year
2. Set `a` to `y%19`
3. Set `b` to `y/100` and `c` to `y%100`
4. Set `d` to `b/4` and `e` to `b%4`
5. Set `f` to `(b+8)/25`
6. Set `g` to `(b-f+1)/3`
7. Set `h` to `(19*a+b-d-g+15)%30`
8. Set `i` to `c/4` and `k` to `c%4`
9. Set `l` to `(32+2*e+2*i-h-k)%7`
10. Set `m` to `(a+11*h+22*l)/451`
11. Set `n` to `(h+l-7*m+114)/31` and `p` to `(h+l-7*m+114)%31`
12. Determine `10*(p+1)+n`

Note that all identifiers represent integers.
The value of `n` gives the month (3 for March and 4 for April) and the value of `p+1` gives the day of the month. These two values can be combined as `10*(p+1)+n` when 23 April would be given as 234.
Write a method `private static int easter(int y)` which, when presented with a year `y`, returns the date of Easter in the form shown at step 12.
Incorporate this method into a complete test program. Verify that the method gives the correct date of Easter for the current year.

Optional extra: consider the length of the Easter cycle. What is the minimum value of n (where $n>0$) such that for all y (in the Gregorian calendar) the dates of Easter in years y and $y+n$ are the same.
Warning: the length of the Easter cycle can be determined by a pencil and paper analysis of the integer constants given in the algorithm. Verifying the length of the cycle by program can consume a considerable amount of computer time.

6. The Friday 13th Problem

Write a program to demonstrate that the 13th of a month is more likely to fall on a Friday than on any other day. Note that:

- `if (n % 4 == 0 && n % 100 != 0 || n % 400 == 0)...`
  year `n` is a leap year.
- the number of days in the leap year cycle of 400 years is an integral multiple of 7 (the program should verify this).
- 1 January 1900 was a Monday.

It is suggested that the program should look at all 4800 thirteenths that lie between 1 January 1900 and 31 December 2299.

7. The Forward and Backward Count Problem

Write a program to sum the series $\sum_{n=0}^{\infty} \frac{1}{1000n+\pi}$ and print the results. The program should compute successive terms starting at $n = 0$ and continue adding terms until the sum ceases to increase. Write out the number of terms computed and the sum of those terms.

The program should then recompute the answer by summing the same terms but in reverse order. This new sum should also be written out.

- Why are the forward and backward sums different?
- What answer would a mathematician give if asked the sum of the series?

8. Accumulating Rounding Errors

Write a program which evaluates $2^n/100$ for each $n = 1, 2, \ldots 16$. Each value should be determined in two different ways. First evaluate `(float)numerator/100.0f` where `numerator` $= 2^n$; this gives a good result. The second way is very naïve: simply add `(float)1/(float)100` to itself 2^n times!

9. Determining a Square Root by Iteration

If x_i is an approximation to $\sqrt{5}$ then x_{i+1} is a better approximation if:

$$x_{i+1} = \frac{1}{2}\left(x_i + \frac{5}{x_i}\right)$$

Prove this and write a program which uses type `double` to determine $\sqrt{5}$ to 10 significant figures.

In writing the program it is probably sensible to have two variables, `x` (which is the latest approximation to $\sqrt{5}$) and `oldx` (the previous approximation to $\sqrt{5}$). Each time round some loop (that is at each *iteration*) consider the following condition:

```
(Math.abs(x-oldx)>1.0e-10d)
```

Note that `Math.abs()` determines the absolute value of its argument and the condition as a whole determines whether `x` and `oldx` differ by more than 10^{-10} (which, when expressed as a `double` constant in Java, is written as `1.0e-10d` where `e` stands for 'times ten to the power of'). This is not strictly what the question requires but will be acceptable.

10. The Recurring Fraction Problem

Consider a function $f(n)$ informally defined thus:

$$f(0) = 2$$

$$f(1) = 1 + \frac{1}{2+1}$$

$$f(2) = 1 + \cfrac{1}{2 + \cfrac{1}{2+1}}$$

$$f(3) = 1 + \cfrac{1}{2 + \cfrac{1}{2 + \cfrac{1}{2+1}}}$$

$$f(4) = 1 + \cfrac{1}{2 + \cfrac{1}{2 + \cfrac{1}{2 + \cfrac{1}{2+1}}}}$$

Note that n is the number of *vincula* in the recurring fraction.
Write a method `private static double f(int n)` which returns the appropriate value.
Incorporate the function into a complete program which will write out the values:

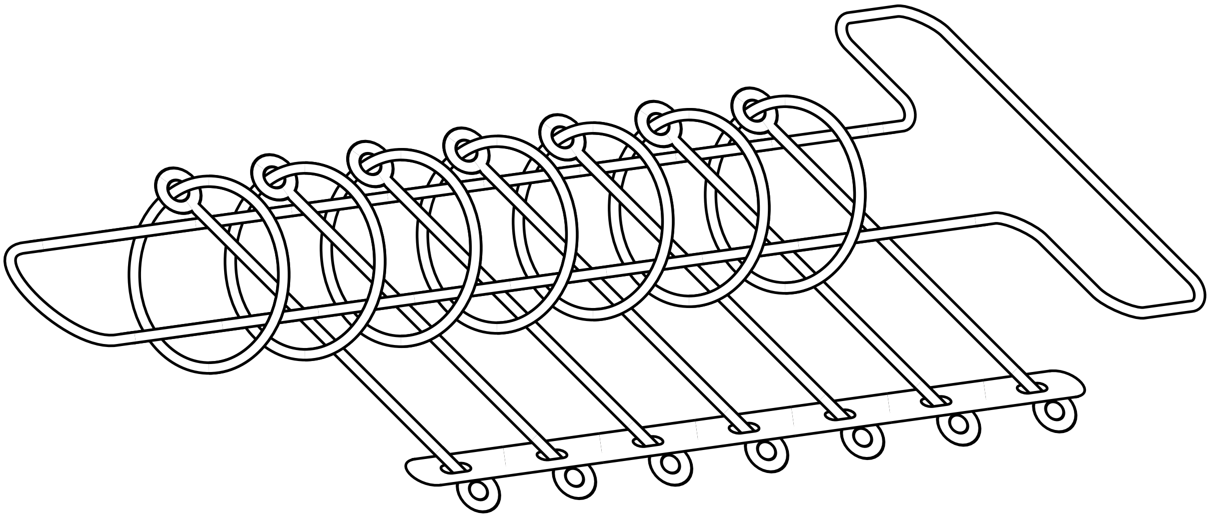
$$f(0), f(1), \ldots f(10).$$

11. Sorting Records

The following is an outline Java program which incorporates some built-in data. The data consist of a array of 8 *records*, where each record is a person's name together with the associated age (in years) and height (in metres). The program should sort the data so that the records are in ascending order of ages. Complete the program.

```
public class PersonalData
 { public static void main(String[] args)
    { Person[] p = {new Person("George",  34, 1.71f),
                    new Person("Betty",   22, 1.76f),
                    new Person("Charles", 24, 1.79f),
                    new Person("Hanna",   29, 1.66f),
                    new Person("Edward",  23, 1.82f),
                    new Person("Frida",   28, 1.77f),
                    new Person("Davina",  33, 1.69f),
                    new Person("Andrew",  25, 1.67f)};
      sort(p);
    }

   private static void sort(Person[] p)
    { .
      .
      for (int i=0; i<p.length; i++)
         System.out.println(p[i]);
    }
 }

class Person
 { private String name;
   private int age;
   private float height;

   public Person(String n, int a, float h)
    { this.name = n;
      this.age = a;
      this.height = h;
    }

   public String toString()
    { return fmtString(this.name,9) + this.age + "  " + this.height;
    }

   .
   .
 }
```

12. The Chinese Rings Puzzle



The mechanics of this wire puzzle are not important. Roughly speaking one has to manipulate the rings until they all come off the T-shaped loop. Write a program which prints out the moves which are required to remove n rings.
For an arbitrary number of rings, the rules are as follows:

a) Each ring can be either on the loop or off it.

b) Only one ring may be moved (from on to off or *vice versa*) at a time.

c) The first ring may be moved at any time.

d) Ring i (where $i > 1$) may be moved if and only if:
   - all rings numbered $i - 2$ or lower are off
   - ring $i - 1$ is on

   [rings higher up (numbered $> n$) the loop may be in any state]

Hence:

To remove the first i rings ...	To replace the first i rings ...
1) Remove the first $i - 2$ rings	1) Replace the first $i - 1$ rings
2) Remove ring i	2) Remove the first $i - 2$ rings
3) Replace the first $i - 2$ rings	3) Replace ring i
4) Remove the first $i - 1$ rings	4) Replace the first $i - 2$ rings

The output for case $n = 4$ should appear thus:

```
 2 off  1 off  4 off  1 on   2 on   1 off  3 off  1 on   2 off  1 off
```

Arrange to have 10 instructions per line. When $n = 4$, precisely one line of output is needed.

13. The Power Problem

The following program makes use of a `class BigNo` which is used to represent large integers and contains methods for operating on these integers.

```
public class Power
 { public static void main(String[] args)
    { BigNo b = power(2,2241);
      System.out.println(b);
    }

   private static BigNo power(int m, int n)
    { BigNo p, s;
      p = n%2 !=0 ? new BigNo(m) : new BigNo(1);
      s = new BigNo(m);
      while (n>1)
       { s = s.multiply(s);
         n /= 2;
         if (n%2 != 0)
            p = p.multiply(s);
       }
      return p;
    }
 }

class BigNo
 { private static final int BASE = 10000;
   private static final int DIGS = 4;

   private int[] value = new int[200];

   private BigNo()
    { this(0);
    }

   public BigNo(int n)       // This converts an ordinary int into a BigNo
    { int i;
      for (i=0; n>0; n /= BASE)
         value[++i] = n%BASE;
      value[0] = i;
    }

   public String toString()  // This converts a BigNo into a String
    { .
      .
    }
```

```
   public BigNo multiply(BigNo that)   // This multiplies one BigNo
    { .                                // by another
      .
      .
    }
 }
```

The method `power` takes two parameters `m` and `n` and raises `m` to the power of `n` by repeatedly using the method `multiply` in `BigNo` objects. As a test case 2 is raised to the power of 2241 and the result is printed out. Complete the program. The method `power` should work for all `m, n > 0` provided m^n is no more than 800 digits long.

14. The Eight Queens Problem

Supply the missing code from the following program so that it prints out the number of ways in which 8 queens can be placed on a chess board such that no queen is under attack from any other. Count all solutions, even those which are rotations or reflections of others. The correct answer is 92.

```
public class EightQueens
 { private static int count=0;

   public static void main(String[] args)
    { tryit(0,0,0);
      System.out.println("There are " + count + " solutions");
    }

   private static void tryit(int left, int above, int right)
    { if (above==0xFF)
          count++;
      else
       { int poss = ~(left | above | right) & 0xFF;
         while (poss != 0)
          { int place = ...;
            tryit(..., ..., ...);
            poss = poss & (~place);
          }
       }
      return;
    }
 }
```

15. Reversing a List

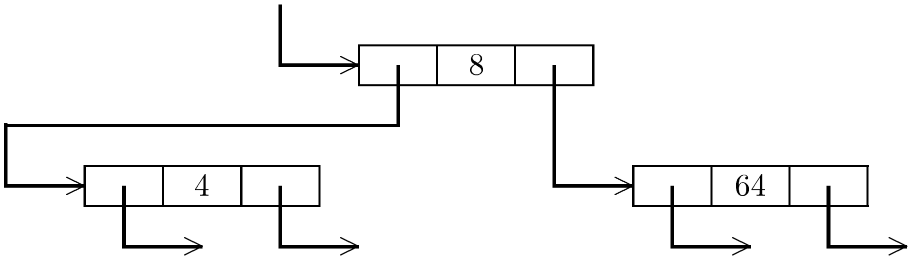
Consider a `class Link` whose definition begins thus:

```
class Link
 { private int val;
   private Link next;

   public Link(int n)
    { this.val = n;
      this.next = null;
    }

   .
   .
```

Any given `Link` can point to a sequence of others to form a list. Incorporate into `Link` a method `public Link reverse()` which returns a new list which is the reverse of that which begins with the given `Link`. It may be profitable for the `reverse` method to make use of a `put` method which places a new element at the end of an existing list.
Write a convincing test program to exercise the `reverse` method.

16. The Tree Sort Problem

In the program outlined below, `class Node` is has three data fields. The middle one, `val`, holds a value if type `int`. The outer ones point to further nodes so that a tree structure is formed:



The tree is built up one node at a time. A new node is put on the tree in such a way that the tree is ordered. Thus $4 < 8 < 64$. If the next new node is to incorporate 32, it will hang from the left of the 64 node.
Complete the methods in `class Node`.

```
public class TreeSort
 { public static void main(String[] args)
    { Node tree = new Node(16);
      tree.put(8);
      tree.put(4);
      tree.put(64);
      tree.put(32);
      System.out.println("Tree elements:  " + tree);
      System.out.println("Element sum:    " + tree.sum());
    }
 }

class Node
 { private Node left;
   private int val;
   private Node right;

   public Node(int n)
    { this.left = null;
      this.val = n;
      this.right = null;
    }

   public void put(int k)
    { .
      .
      .
    }

   public String toString()
    { .
      .
      .
    }

   public int sum()
    { .
      .
      .
    }
 }
```

17. What does this do?

The following is a complete java program. The problem is to analyse the program and determine what it writes out *without* keying the program in and trying it.

```
class K
 { public static int k = 0;
 }

abstract class JJ
 { public abstract void upk();
 }

class Jack extends JJ
 { public void upk()
    { K.k += 10;
    }
 }

class Jill extends JJ
 { public void upk()
    { Jack ink = new Jack();
      ink.upk();
      K.k += 200;
    }
 }

public class UpkEtc
 { public static void main(String[] args)
    { Jack ink = new Jack();
      fred(ink, 3000);
      System.out.println("Value is " + K.k);
    }

   private static void fred(JJ uk, int n)
    { int a = 10*n;
      uk.upk();
      if (n > 1000)
       { K.k++;
         Jill ink = new Jill();
         fred(ink, n-1000);
         a += n;
       }
      K.k += a;
    }
 }
```

18. The Word Count Problem

Write a program whose source code is to be in `WordCount.java` which, when supplied with a file of ordinary text, will count the number of words in the text. For the purposes of this program, a word is any sequence of characters delimited by spaces, tabs and newlines. The characters tab and newline in Java are `'\t'` and `'\n'` respectively.
Here is an example run:

```
$ java WordCount
The quick
brown fox
jumps over
the lazy dog.
^d
```

This should produce the output:

```
Total number of words: 9
```

Note that it would be more usual to present the data as a file. For example, if the fox story were in the file `fox` the following command would produce the same output:

```
$ java WordCount <fox
```

19. The Croquet Fixtures Problem

The program below attempts to count the number of ways in which 8 croquet players can compete by pairs on 4 lawns over 7 rounds.
The only restrictions are that in each round all 8 players must take part and after 7 rounds each player must have played every other player. No restrictions are places on which lawn a given player may play on.
Note that 28 games are played and this is ${}^8C_2$, the number of ways of choosing 2 players from 8. The program behaves rather like the 8-queens problem in that it counts the number of ways in which the 28 pairings can be placed.
The call `System.exit(0);` causes the program to terminate abruptly after printing the very first solution found. There are so many solutions that the program would take a long time to run if it were allowed to complete.
The first schedule counted happens to be:

```
12  13  14  15  16  17  18
34  24  23  26  25  28  27
56  57  58  37  38  35  36
78  68  67  48  47  46  45
```

Consider how the program works and supply the missing body of the method `printOut`.

```
public class CroquetA
 { private static int count=0;
   private static int[] plan = new int[28];
   private static boolean[] alreadyPlayed = new boolean[193];
   private static final int p1 = 1, p2 = 2, p3 = 4, p4 =  8,
                            p5 =16, p6 =32, p7 =64, p8 =128;
   private static final int maxgame = 28;

   public static void main(String[] args)
    { for (int i=0; i<=192; i++)
          alreadyPlayed[i] = false;
      tryit(0);
      System.out.println("There are " + count + " solutions");
    }

   private static void tryit(int game)
    { if (game == maxgame)
       { count++;
         printOut();
         System.exit(0);    //  Abort after printing first solution
         return;
       }
      int imposs = 0;
```

```
      for (int i = game & 0x1C; i<game; i++)
         imposs |= plan[i];
      int poss = ~imposs & 0xFF;
      int[] player = new int[8];
      int k = -1;
      while (poss!=0)
       { player[++k] = poss & -poss;
         poss &= ~player[k];
       }
      for (int i=0; i<k; i++)
         for (int j=i+1; j<=k; j++)
          { int pi = player[i];
            int pj = player[j];
            int pair = pi | pj;
            if (!(alreadyPlayed[pair]))
             { plan[game] = pair;
               alreadyPlayed[pair] = true;
               tryit(game+1);
               alreadyPlayed[pair] = false;
             }
          }
    }

   private static void printOut()
      .
}
```

20. The Refined Croquet Fixtures Problem

The first solution counted by the Croquet Fixtures Program would not be acceptable to the organiser of a croquet tournament. Player 1 always plays on lawn 1 and thereby has an unfair advantage.

Modify the program so as to impose two extra restrictions. First, every player must play twice on three of the lawns and once on the remaining lawn and, secondly, a player may not play two consecutive games on the same lawn. Print out the first solution found in the following form:

```
12 35 14 27 48 57 68   2 2 1 2 2 1 2 2
34 17 26 18 67 38 25   2 2 2 1 1 2 2 2
56 28 37 45 23 16 47   1 2 2 2 2 2 2 1
78 46 58 36 15 24 13   2 1 2 2 2 2 1 2
```

The last eight columns show the lawn counts for each player. The first row indicates that players 3 and 6 play only once on the first lawn but the other players each play twice on that lawn.

21. The Triangular Solitaire Problem

Write a program to determine the number of ways of successfully concluding a game of Triangular Solitaire. This game is played on a pegboard which consists of a triangular array of 15 holes which may be numbered thus:

```
        1
      2   3
    4   5   6
  7   8   9   10
11  12  13  14  15
```

The board has 18 triplets of adjacent holes such as 4-5-6 and 3-5-8. [Note that 1-5-13 is *not* a triplet; the holes are not deemed adjacent.]

At the start, all holes are occupied by pegs except hole 1. The pegs are then removed one at a time according to certain rules. Play ends when no move is possible. If at the time play ends there is exactly one peg remaining and that peg is in hole 1 the game was a success and counts as a solution.

A turn may be executed if any triplet of holes has a peg in the middle hole and one of the other holes. The turn consists of the outside peg jumping over the middle peg to the vacant hole and the removal of the middle peg. Thus, if there is a peg in holes 4 and 5 but not 6 then the peg in hole 4 may be moved to hole 6 and the peg in hole 5 is removed from the board.

The state of the 15 holes can be represented by the bottom 15 bits of a word with the bits numbered 1 to 15 from left to right. A one indicates a peg and a zero indicates that the hole is empty. Using this scheme, the first and last stages in a game are represented by the bit patterns 011111111111111 and 100000000000000 respectively. In octal, these values are 037777 and 040000.

It is convenient to begin the program as follows:

```
public class SolitaireA
 { private static int count=0;

   private static int[] triple = {007000, 000700, 000340, 000034, 000016,
                                  000007, 001104, 012200, 002210, 064000,
                                  024400, 004420, 004204, 002102, 022100,
                                  001041, 011040, 051000};

   private static int[] ta =     {006000, 000600, 000300, 000030, 000014,
                                  000006, 001100, 012000, 002200, 060000,
                                  024000, 004400, 000204, 000102, 002100,
                                  000041, 001040, 011000};

   private static int[] tb =     {003000, 000300, 000140, 000014, 000006,
                                  000003, 000104, 002200, 000210, 024000,
                                  004400, 000420, 004200, 002100, 022000,
                                  001040, 011000, 050000};
```

```
private static final int first = 037777, last = 040000;

public static void main(String[] args)
 { tryit(first);
   System.out.println("There are " + count + " solutions");
 }
```

The 18 elements in `triple` represent the 18 triplets and for each element there are corresponding elements in `ta` and `tb` which give the two arrangements of the three bits in a triplet which permit a valid move.
All the work takes place in the method `tryit` and a correct program should show that there are 6816 solutions.

22. Hash Problem

An array of data contains not more than 500 positive numbers of type `double` which are reasonably well scattered in the range 0.0 to 10000.0 and a program is required which will determine whether all the numbers in the data are different.
The program should inspect each number in turn and if the number has been met before a variable `duplicates` should be incremented by 1. In detail:

1. Declare an array `double[] table = new double[630]`
2. Set all 630 elements to `-1.0d`
3. Assign the first number in data to a `double` variable `x`
4. Assign to an `int` variable `n` thus: `int n = (int)x % 630;`
5. Set: `table[n] = x`
6. Assign the next number to `x`
7. Repeat from 4 but note that for each `x` there are three possibilities:
   a) `table[n]` is empty ($= -1.0$) so fill with `x`
   b) `table[n]` holds `x` so increment `duplicates`
   c) `table[n]` holds another value `x'` which by bad luck is such that:
   `(int)x' % 630 == (int)x % 630`
8. In case 7c increment `n` and repeat from 7a.

Invent some suitable data. Just a few entries in an array `data` will suffice for test purposes, for example:

```
private static double[] data = {637.42d, 6300.95d, 7.81d, 6300.95d,
                                712.72d, 4325.22d, 2.79d, 3125.77d,
                                813.02d, 3125.77d, 6.42d, 1234.56d};
```

Write a program to process the data in the manner outlined. This program should not incorporate any bug which may be present in the above description!

23. The Quadratic Equation Problem

Write a program which solves quadratic equations of the form:

$$ax^2 + bx + c = 0$$

Data should be organised into lines with a set of three test coefficients on each line. If a set of coefficients is such as to lead to complex roots, then the program should put out a message to this effect and pass on to the next set of coefficients without further calculation. The program should aim to produce results to the greatest accuracy warranted by the input, and deal sensibly with extreme coefficient values and other awkward cases. The following test cases should be used:

a	b	c
1	5	6
0.1	0.5	0.6
0	0	1
0	1	0
0	1	1
0	10000	1
0	10^{-34}	-10^{34}
0	10^{34}	-10^{-34}
1	-20000	10^{8}
0.1	-2000	10^{7}
10^{-34}	-2	10^{34}
1	0	0
1	1	0
1	10^{-34}	0
1	0	1
1	1	1
1	10	1
1	100	1
1	1000	1
1	10000	1
1	10^{34}	1
10^{34}	0	-10^{34}
10^{-34}	0	-10^{-34}
1	10^{-34}	-10^{34}
1	10^{34}	10^{-34}
10^{-10}	-10^{30}	10^{-10}

In each of the above cases, it is trivial to determine the correct (or nearly correct) results using pencil and paper. Annotate the output, by hand, writing the correct results against those determined by the program; comment on any gross discrepancies.