PClassic 2013 Sample Problems

1. Polygon Containment

Write a program which determines if one polygon is contained within another. We consider polygon A to be contained by polygon B if every vertex of A is within the perimeter and not lying on any edge of B.

Sample Input:

The input will be a colon-delimited pair of polygons, which are semicolon-delimited lists of comma-delimited coordinate pairs. The polygons are formed by connecting each coordinate pair to the next, and the last to the first. You can assume that the polygons are simple (do not intersect themselves) and every coordinate is an integer.

```
(1): "1,1;1,2;2,2;2,1:0,0;0,3;3,3;3,0"
(2): "1,1;3,1;1,3:0,0;0,2;2,2;2,0"
```

Sample Output:

Output a 1 if the first polygon is contained by the second. Otherwise, output a 0.

```
(1): 1
(2): 0
```

2. Cash Register

When you go to the grocery store and pay in cash, you are handed back some change by the cashier. To avoid the possibility of miscounts, Wawa has contracted you to write the software for an automatic change dispenser.

Sample Input:

The function should take two inputs. The first input should be the total price of the goods being purchased, *in cents* (i.e. if you rang up a charge of $14.03 at Wawa, this would be entered as 1403). The second input should be the amount of money you paid with, also *in cents*. For example, if I gave the cashier a twenty-dollar bill, this input would be 2000.

Assume clean input — i.e. the price is never greater than how much the customer paid.

```
(1): (1403, 2000)
(2): (599, 1000)
```

Sample Output:

Assume that Wawa does not give change in bills of value greater than $20. You will return an array of integers where index 0 represents the number of twenty-dollar bills to give back to the customer, index 1 represents the number of ten-dollar bills, 2 the number of five-dollar bills, 3 the number of one-dollar bills, 4 the number of quarters, 5 the number of dimes, 6

the number of nickels, and 7 the number of pennies.

```
(1): [0, 0, 1, 0, 3, 2, 0, 2]
(2): [0, 0, 0, 4, 0, 0, 0, 1]
```

3. Driving in the Snow

A group of friends intend to gather for a programming contest today, but last night a snowstorm closed most of the roads in their town. The snowplows have reopened some roads. Write a program which determines if they can all still get together. This is possible if every friend can, by some series of hops, get to any other friend's house - or, to speak mathematically, if the graph of friends and roads is connected.

Sample Input:

The input will be a semicolon-delimited list. The first element is the number of friends. Each element after that is a comma-delimited pair of two friends who can drive between each other's houses. To think of this as a graph, the first element is the number of vertices, and each element after that is an edge.

```
(1): "5;0,1;0,3;0,4;1,2"
(2): "7;0,1;0,2;0,6;1,2;2,3;3,6;4,5"
```

Sample Output:

Output a 1 if everyone can still gather. Output a 0 if someone can't make it.

```
(1): 1
(2): 0
```

4. Digital Root

The digital root of a positive integer is found by summing the digits of the integer. If the resulting value is a single digit then that digit is the digital root. If the resulting value contains two or more digits, those digits are summed and the process is repeated. This is continued as long as necessary to obtain a single digit.

For example, consider the positive integer 24. Adding the 2 and the 4 yields a value of 6. Since 6 is a single digit, 6 is the digital root of 24. Now consider the positive integer 39. Adding the 3 and the 9 yields 12. Since 12 is not a single digit, the process must be repeated. Adding the 1 and the 2 yields 3, a single digit and also the digital root of 39.

Sample Input:

```
(1): 24
(2): 39
```

Sample Output:

```
(1): 6
(2): 3
```