Problem 5: Brick House

Points: 10

Author: Holly Norton, Fort Worth, Texas, United States

Problem Background

We want to build a row of bricks for our brick house that is a certain number of inches long, and we have a number of small bricks and large bricks with which to do it. You need to write an application that will decide if its is possible to build this row of bricks using some or all of the given bricks. You do not need to use all of the given bricks!

Problem Description

Your program will be given a goal length for the brick wall and the number of small and large bricks available. Small bricks are each 1 inch long. Large bricks are 5 inches long. You will need to determine if it is possible to build a row of bricks exactly as long as the goal using only the available bricks.

Sample Input

The first line of your program's input, **received from the standard input channel**, will contain a positive integer representing the number of test cases. Each test case will consist of a single line, including three non-negative integers separated by spaces:

- The first integer represents the number of small, one-inch-long bricks
- The second integer represents the number of large, five-inch-long bricks
- The third integer represents the target length of the wall, **X**, in inches

```
3
3 1 8
3 1 9
3 2 10
```

Sample Output

For each test case, your program must print a single line with the word "true" if it is possible to build a wall of exactly **X** inches using only the bricks available. Otherwise, it should print "false".

```
true
false
true
```

Problem 6: Around and Around

Points: 15

Author: Chris Mason, Sunnyvale, California, United States

Problem Background

In 1962, John Glenn completed a historic spaceflight, orbiting the Earth three times in a small spacecraft. This flight was one of many that paved the way for an era of space exploration, eventually leading to the moon landings just seven years later. Despite the seemingly simple nature of Glenn's flight, it still required very precise calculations to ensure that he remained in orbit and didn't either fly off into space or come crashing back to Earth.

Objects in orbit don't remain in space simply because they've left Earth's atmosphere; they're still constantly falling towards Earth. The reason they stay in space is because they're moving so fast that they continually "miss" the Earth as they fall. In the case of John Glenn's historic flight, he was moving at an orbital speed of 17,544 miles per hour (28,234.8 kilometers per hour). This is fast enough to travel from New York to London in less than 12 minutes. During his entire flight, which lasted just short of five hours, Glenn travelled a total distance of 75,679.3 miles (121,794 kilometers).

Your task today is to determine how far an object in orbit at a particular height will travel during a single orbit of Earth.

Problem Description

Your program will be given the altitude of an object orbiting around Earth at the equator. Using this information, your program must calculate the total distance travelled by that object during a single orbit. It will help you to know that the circumference of the Earth at the equator is 40,075 kilometers.

Sample Input

The first line of your program's input, **received from the standard input channel**, will contain a positive integer representing the number of test cases. Each test case will consist of a single line, including an integer representing the object's altitude above the Earth's sea level in kilometers. Altitudes will be greater than or equal to 160 (the lowest possible orbital height for Earth).

```
3
160
200
265
```

Sample Output

For each test case, your program must output the distance travelled by an object orbiting around the equator at the given altitude in kilometers. Each value should be rounded to the nearest tenth of a kilometer (one decimal place).

```
41080.3
41331.6
41740.0
```

Problem 7: Image Compression

Points: 15

Author: Steve Brailsford, Marietta, Georgia, United States

Problem Background

Images can be saved onto a computer in many different types of file formats, each with its own advantages and disadvantages. JPEG (or JPG) images are commonly used for photography, because their format allows the image information to be compressed, reducing the size of the file and allowing you to take more pictures. The downside to this is that repeatedly editing a JPEG image causes the quality of the image to gradually get worse over time; each time the file is saved, the existing image data is compressed further and further, losing fine details.

The process of compressing a JPEG image is complicated but can be broken down into several individual steps. One of these steps is called quantization, which takes a wide range of numbers created by a previous step in the process and converts them to a smaller, more manageable scale. This results in some loss of detail as previously mentioned; two different but close numbers may be converted to the same result number. However, the human eye often cannot discern very high-frequency changes, so this loss is usually not noticeable.

Problem Description

Your program will need to implement an example quantization algorithm that accepts perceived brightness values and converts them to an integer value between 0 and 255 inclusive. Your program will be given a list of decimal values representing brightness values (such as might be read by a scanner). Your program must identify the highest (max) value and the lowest (min) value from the list of values, then convert all values in the list to the target scale using this formula:

$$Output = \frac{Input - Min}{Max - Min} * 255$$

All results should be rounded to the nearest integer.

Sample Input

The first line of your program's input, **received from the standard input channel**, will contain a positive integer representing the number of test cases. Each test case will include the following lines of input:

- A positive integer, **X**, representing the number of values in the list
- **X** lines, each containing a decimal number to be converted

```
2
5
0.0
25.0
50.0
75.0
100.0
6
12.3
-67.1
122.8
428.4
-15.9
221.0
```

Sample Output

For each test case, your program must output the list of converted numbers, maintaining the same order. Print one number per line, and round all results to the nearest integer.

```
0
64
128
191
255
41
0
98
255
26
148
```