**ACM High School Programming Contest**

**2020 Programming Problem Set**

**DO NOT OPEN THIS PACKET UNTIL INSTRUCTED TO BEGIN!**

1. **General Notes**
   1. Do the problems in any order you like. They do not have to be done in order from 1 to 12.
   2. All problems have a value of 60 points. Incorrect submissions receive a deduction of 5 points but may be reworked and resubmitted. Deductions are only included in the team score for problems that are ultimately solved correctly.
   3. There is no extraneous input. All input is exactly as specified in the problem. Unless specified by the problem, integer inputs will not have leading zeros. Unless otherwise specified, your program should read to the end of file.
   4. Your program should not print extraneous output. Follow the form exactly as given in the problem.
2. **Names of Problems**

|  |  |
| --- | --- |
| **Number** | **Name** |
| Problem 1 | Isabelle |
| Problem 2 | Frustrating Frustums |
| Problem 3 | Feeding Bevo |
| Problem 4 |  |
| Problem 5 |  |
| Problem 6 |  |
| Problem 7 |  |
| Problem 8 | Pyramint |
| Problem 9 | What’s for Dinner? |
| Problem 10 |  |
| Problem 11 | Rosetta StOwOne |
| Problem 12 |  |

1. **Isabelle**

**Program Name: Isabelle.java**

here

**Constraints**

**Input**

**Output**

**Example Input File**

**Example Output to Screen**

1. **Frustrating Frustums**

**Program Name: Frustums.java Input File: frustums.dat**

here

**Constraints**

**Input**

**Output**

**Example Input File**

**Example Output to Screen**

1. **Feeding Bevo**

**Program Name: FeedingBevo.java Input File: feedingbevo.dat**

A dog looking at the camera

Description automatically generated

Rick and Lillian recently took their dog, Bevo, to the vet and were told that Bevo needs to be on a more regular feeding schedule to lose weight. However, Rick and Lillian are very forgetful and will sometimes feed Bevo more than they should, at the wrong time, or even forget to feed him at all. Rick has put together a button device that Bevo has been trained to press when he wants food. This button will send the date and time to a device that will determine if it is time to feed Bevo. Bevo is only allowed to eat once per day between 4pm and 7 pm.

**Input**

The first line of input will contain a single integer t that indicates the number of days of data. The first line of each day will contain a single integer n that denotes the number of times Bevo has pressed the button that day. Each of the following n lines will contain a date and time in the format example 09/27/1995 13:52.

**Output**

For each time Bevo presses the button:

* Print “It is not time to feed Bevo!” if it outside of Bevo’s feeding time.
* Print “Feeding Bevo at [time].” if it is within Bevo’s feeding time, where [time] denotes when Bevo pressed the button.
* Print “Bevo has already eaten!” if Bevo presses the button more than once during his feeding time.

At the end of each day, print the date followed by either “Bevo ate today.” or “Bevo did not eat today.” Print “=====” after each day.

**Example Input File**

2

1

02/26/2020 19:00

3

03/18/2019 01:14

03/18/2019 16:00

03/18/2019 16:12

**Example Output to Screen**

It is not time to feed Bevo!

02/26/2020 Bevo did not eat today.

=====

It is not time to feed Bevo!

Feeding Bevo at 16:00.

Bevo has already eaten!

03/18/2019 Bevo ate today.

=====

1. **Name**

**Program Name: .java Input File: Name.dat**

here

**Constraints**

**Input**

**Output**

**Example Input File**

**Example Output to Screen**

1. **Name**

**Program Name: .java Input File: Name.dat**

here

**Constraints**

**Input**

**Output**

**Example Input File**

**Example Output to Screen**

1. **Name**

**Program Name: .java Input File: Name.dat**

here

**Constraints**

**Input**

**Output**

**Example Input File**

**Example Output to Screen**

1. **Name**

**Program Name: .java Input File: Name.dat**

here

**Constraints**

**Input**

**Output**

**Example Input File**

**Example Output to Screen**

1. **Pyramint**

**Program Name: Pyramint.java Input File: pyramint.dat**

You’ve obtained a large supply of mints from your…well, it doesn’t really matter where they came from. Don’t worry about it. They’re here. Your task? Assemble them in the perfect shape: a pyramid. You’ll need to think through just how many mints will be required to complete this engineering feat. To make your calculations easier, this pyramid will have a square base, and a height of n. More formally, a pyramint is defined as a three-dimensional structure with the following recursive definition:

A pyramint of height 1 (a 1-pyramint) is just a single mint.

For n > 1, an n-pyramint has a base of an nxn square of mints, directly above which is an (n-1)-pyramint.

**Constraints**1 ≤ t ≤ 10

1 ≤ n ≤ 1900

**Input**

The first line of input will contain a single integer t that indicates the number of test cases to follow. The first and only line of each test will contain a single integer n that denotes the height of the requested pyramint.

**Output**

For each test case, output the number of mints needed for the n-pyramint.

**Example Input File**

4

1

2

4

13

**Example Output to Screen**

1

5

30

819

1. **What’s for Dinner?**

**Program Name: Dinner.java Input File: dinner.dat**

Flavius and his friends are hungry. However, no one ever seems to agree on where to eat dinner together. His friend Josephus creates the following process to pick who decides what’s for dinner.

Start with n people standing in a circle. One person is designated Person 1 and then they are numbered (clockwise) through to Person n. Person 1 then removes Person 2 from the circle. Continuing around the circle, Person 3 removes Person 4 from the circle, Person 5 removes Person 6 from the circle, and so on. This procedure continues with the next person removing the closest person next to them clockwise. They go on until only one person remains in the circle, and this person must choose what’s for dinner.

Flavius, however, only wants to eat at one restaurant… Olive Garden. Instead of standing at a random position in Josephus’s circle and hoping for the best, Flavius wants you to write a program to determine where he should stand in the circle so that he will always be able to pick Olive Garden for dinner.

**Constraints**1 ≤ t ≤ 12

2 ≤ n ≤ 5000

**Input**

The first line of input will contain a single integer t that indicates the number of test cases to follow. The first and only line of each test case will contain a single integer n that denotes the number of people standing in the circle.

**Output**

For each test case, output “Person” followed by the position number Flavius should stand at in the circle.

**Example Input File**

3

2

8

12

**Example Output to Screen**

Person 1

Person 1

Person 9

1. **Name**

**Program Name: .java Input File: Name.dat**

here

**Constraints**

**Input**

**Output**

**Example Input File**

**Example Output to Screen**

1. **Rosetta StOwOne**

**Program Name: Rosetta.java Input File: rosetta.dat**

here

**Constraints**

**Input**

**Output**

**Example Input File**

**Example Output to Screen**

1. **Name**

**Program Name: .java Input File: Name.dat**

here

**Constraints**

**Input**

**Output**

**Example Input File**

**Example Output to Screen**