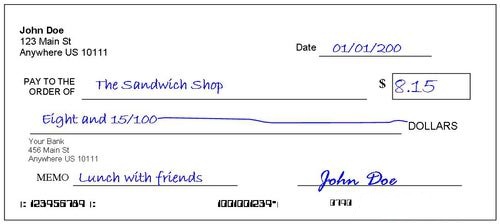
**Java program:** Prob09.java

**Input File:** Prob09.in.txt

**Output:** Your output needs to be directed to stdout (i.e., using System.out.println())

**Introduction**

OK, OK, we know everyone pays their bills online these days but there are still some occasions where a check is called for. To help with these few occasions, write a program that converts dollar amounts from their numeric form into their English equivalent, as would be seen on a check.

Guidelines:

1. Not all input numbers will have a decimal. Numbers with no decimal should be assumed to have 0 Cents.
2. There will be no more than 2 numbers after the decimal point.
3. Capitalize each English word except the word “and”.
4. All inputs will be less than one million.
5. You should use the singular for Dollar and Cent when appropriate.

**Program Input**

The file Prob09.in.txt will contain a list of numbers, one per line.

**Example Input:**

143

2.34

1.01

1234.56

**Program Output**

Your program should output the English version of each number in the input file in the order it was encountered in the file.

**Example Output:**

One Hundred Forty Three Dollars and 0 Cents

Two Dollars and 34 Cents

One Dollar and 1 Cent

One Thousand Two Hundred Thirty Four Dollars and 56 Cents

**Reference**

For your reference, here are the first 100 integers written in English:

|  |  |  |  |
| --- | --- | --- | --- |
| Zero | Twenty Five | Fifty | Seventy Five |
| One | Twenty Six | Fifty One | Seventy Six |
| Two | Twenty Seven | Fifty Two | Seventy Seven |
| Three | Twenty Eight | Fifty Three | Seventy Eight |
| Four | Twenty Nine | Fifty Four | Seventy Nine |
| Five | Thirty | Fifty Five | Eighty |
| Six | Thirty One | Fifty Six | Eighty One |
| Seven | Thirty Two | Fifty Seven | Eighty Two |
| Eight | Thirty Three | Fifty Eight | Eighty Three |
| Nine | Thirty Four | Fifty Nine | Eighty Four |
| Ten | Thirty Five | Sixty | Eighty Five |
| Eleven | Thirty Six | Sixty One | Eighty Six |
| Twelve | Thirty Seven | Sixty Two | Eighty Seven |
| Thirteen | Thirty Eight | Sixty Three | Eighty Eight |
| Fourteen | Thirty Nine | Sixty Four | Eighty Nine |
| Fifteen | Forty | Sixty Five | Ninety |
| Sixteen | Forty One | Sixty Six | Ninety One |
| Seventeen | Forty Two | Sixty Seven | Ninety Two |
| Eighteen | Forty Three | Sixty Eight | Ninety Three |
| Nineteen | Forty Four | Sixty Nine | Ninety Four |
| Twenty | Forty Five | Seventy | Ninety Five |
| Twenty One | Forty Six | Seventy One | Ninety Six |
| Twenty Two | Forty Seven | Seventy Two | Ninety Seven |
| Twenty Three | Forty Eight | Seventy Three | Ninety Eight |
| Twenty Four | Forty Nine | Seventy Four | Ninety Nine |

**Java program:** Prob10.java

**Input File:** Prob10.in.txt

**Output:** Your output needs to be directed to stdout (i.e., using System.out.println())

**Introduction**

Since their invention over 200 years ago, trains have provided a safe and efficient method of transportation. Passenger trains carry hundreds of travelers from station to station, requiring only track to be laid between them. In the beginning, train stations had dedicated tracks which linked them. This proved to be a waste of infrastructure as much track was going unused the majority of the time. Railroad switches were designed to solve this problem. Switches allow trains to quickly cross to an adjacent track. In this manner, trains were no longer bound to the track they began their journey on. Your task will be to determine, for each starting station, what the end station is given a layout of interconnected tracks.

**Program Input**

The file Prob10.in.txt will contain a set of train station names, tracks, barriers, and switches.

* Train tracks are denoted by an equal sign (=). Each track has a start station and an end station. Trains move from left to right.
* Station names will always be three characters and will appear at the beginning and end of a track. Station names are not necessarily unique.
* Between two adjacent tracks will be barriers and switches. Barriers are denoted by a minus sign (-), and switches are denoted by a vertical bar (|). When a train reaches a switch, it will always crossover to the neighboring track. For example:

AAA======================BBB

------|---------|---

AAB====================CCC

--------|---------|----------

AAC=============================DDD

A train leaving station AAA would cross to the middle track, then to the bottom track, and then back to the middle track to arrive at station CCC.

* Switches will never connect directly to a station – they can only connect two track pieces.
* If the situation occurs where it’s possible for a train to switch to either of the two neighboring tracks, the train will always move in the “upward” direction. Also, switches can be used to cross multiple tracks if they line up correctly. For example:

AAA=======BBB AAA=======BBB

---|--- ---|---

AAB=======BBA AAB=======BBA

---|--- ---|---

AAC=======BBC AAC=======BBC

In the example on the left, a train leaving station AAB would arrive at the switches in the middle and would choose to move up instead of down. The train would end up at station BBB. In the example on the right, a train leaving station AAA would cross over the middle track because the switches are aligned. The train would arrive at station BBC.

**Example Input:**

AAA========================================================GGG

-|-----|------|--------|-----------|--------------------

BBB=======================================HHH

--------|----------------|-----------|----------

CCC================================================III

-----|--------------|--------------------|-----------------

DDD===========================================================JJJ

-------|-------------|-------------------------------------

EEE============================KKK

------|----------------|---------------

FFF=======================================LLL

**Program Output**

Your program should list out the starting stations in the order they were encountered in the input file. For each starting station, list the ending station for a train leaving from that starting station in the following form:

Start: StartStation, End: EndStation

**Example Output:**

Start: AAA, End: JJJ

Start: BBB, End: LLL

Start: CCC, End: III

Start: DDD, End: HHH

Start: EEE, End: KKK

Start: FFF, End: GGG