University Interscholastic League

Computer Science Competition

2001 Regional Programming Set

I. General Notes

1. Do the problems in any order you like. They do not have to be done in order from 1 to 8.

2. Problems have different point values, depending on their difficulty.

3. Several problems have a reference to columns of an input line, such as column 1 or columns 1-3. In these cases column is referring to the character position on the input line. Column 1 refers to the first character position on the line, while columns 1-3 refer to the first three positions on the line.

4. 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.

5. Your program should not print extraneous output. Follow the form exactly as given in the problem.

II. Point Values and Names of Problems

Number	Name	Point Value
Problem 1	Taxes, Taxes	5
Problem 2	Shuttle Displays	4
Problem 3	Numbers Game	5
Problem 4	Am I Close?	8
Problem 5	Your Move	7
Problem 6	Just the Fax	9
Problem 7	Too Many Acronyms	9
Problem 8	Airline Miles	3
Total		**50**

Problem 1	**Taxes, Taxes**	5 Points

Program Name: taxes.cpp **Input File: taxes.dat**

Have you ever wondered how what you pay in federal taxes compares to the total collected? It is popular for people to assume that they are paying much more than everyone else. So, the government wants to create a web site that compares different people's amounts of tax. (OK, so they probably would not actually build such a web site, but we can pretend.)

Table 1 below shows the graduated income tax schedule for 1999.

Table 1: Federal Income Taxes

Salary Range		Formula to Compute Federal Income Taxes
Minimum	**Maximum**	
$0	$15000	Federal Income Taxes = $0
$15001	$28450	Federal Income Taxes = (Income - $15000) * 0.15
$28451	$54850	Federal Income Taxes = $2017.50 + ((Income - $28450) * 0.28)
$54851	$87250	Federal Income Taxes = $9409.50 + ((Income - $54850) * 0.31)
$87251	$111500	Federal Income Taxes = $19453.50 + ((Income - $87250) * 0.36)
$111501	Unlimited	Federal Income Taxes = $28183.50 + ((Income - $111500) * 0.39)

In addition to the Federal Income Taxes, there are Social Security Income (SSI) and Medicare Taxes structured as described in Table 2.

Table 2: Social Security and Medicare Taxes

Salary Range		Formula to Compute Social Security Income (SSI) and Medicare Taxes
Minimum	**Maximum**	
$0	$83400	SSI and Medicare Taxes = (Income * 0.0765)
$83401	Unlimited	SSI and Medicare Taxes = $5004.00+ (Income * 0.0165)

The "Total Income Taxes" are the sum of the "Federal Income Taxes" and the "SSI and Medicare Taxes". All taxes are computed in dollars and cents. Fractions of pennies are discarded, not rounded. The following table shows some sample incomes and their associated taxes.

Income	Federal Income Taxes	SSI and Medicare Taxes	Total Income Taxes
$45674	($2017.50 + (($45674 - $28450) * 0.28) = **$6840.22**	($45674 * 0.0765) = **$3494.06**	**$10334.28**
$12222	**$0.00**	($12222 * 0.0765) = **$934.98**	**$934.98**
$215426	($28183.50 + (($215426 - $111500) * 0.39) = **$68714.64**	($5004.00 + ($215426 * 0.0165)) = **$8558.52**	**$77273.16**

Your program will be given a series of income values. For each income, your program should compute the Total Income Taxes based on the formulas in Table 1 and Table 2.

Input

Input to your program consists of a series of income values (1 <= Income <= 10000000). Each income is expressed as an integer in dollars on a line by itself. Incomes do not have leading zeroes, nor do they contain anything other than numeric digits. They are not preceded by the "$" symbol.

Output

For each income, your program is to compute the total taxes for the income and print the total taxes on a line by itself. The format of your total taxes output **must** have a "$" in column 1 followed by the number of dollars starting

in column 2, followed by a decimal point, followed by 2 digits of cents. The dollars component of the total taxes must be at least one digit long and have no leading zeroes. Zero dollars is expressed as a single zero digit ($0.00).

NOTE: This problem is representative of a real world problem and the challenges faced when working with dollars and cents. The output from your program may differ from the sample output shown below by a few pennies. The differences may be due to:

- the data types used
- the order in which operations are performed
- the processor
- the compiler (16–bit or 32–bit)

etc.

Example: Input File

```
45674
12222
215426
73468
```

Output to screen

```
$10334.28
$934.98
$77273.16
$20801.38
```

Problem 2	**Shuttle Displays**	4 Points

Program Name: shuttle.cpp **Input File: shuttle.dat**

[This is a real story/situation.] The original displays on board the Space Shuttle (before the recent update) used green phosphorous semi-intelligent displays that displayed characters based on an embedded ROM (read-only memory). This allowed a very simple interface from the computers. It supplied the triplet (x,y,character) describing the position on the screen and the character that should appear there.

One of the maintenance checks on this display required a check of the ROM. A failure in the ROM could cause the distortion of a character (Q appears as an O for example) or the wrong character being displayed. The only way to check the proper operation of the ROM is to manually verify that it can display all characters. (We will limit it to the upper-case alphabetic characters only.) The human folks believe the most effective way to facilitate this is not to force the technician to check each character individually but rather to display a sentence that uses all characters. The technician then intuitively gravitates to spelling errors.

Your company is building these monitors and the test procedures. You need to write a program that, given a sentence of up to 80 characters, will determine if all upper case alphabetic characters are used at least once in the sentence. They can then submit sample sentences to your program to determine which are viable candidates for the test procedure.

For example, you can see that the sentence

THE QUICK BROWN FOX JUMPED OVER THE LAZY DOGS

Contains all of the upper case alphabetic characters at least once and is therefore a viable candidate. You can also see that

THE LAZY DOGS DID NOT SEE THE QUICK BROWN FOX

Does not contain a J, M, P, or V and is therefore not a viable candidate sentence.

Input

Input to your program is a series of sentences consisting of only upper case alphabetic characters and spaces up to 80 characters in length. Each sentence is contained on a line by itself in the input file. Your program should ignore the blank spaces.

Output

For each input sentence, your program should determine if the sentence is a viable candidate. If it is a viable candidate, your program should print exactly the message "Sentence is a candidate" on a line by itself. If it is not a viable candidate, your program should print exactly the message "Sentence is not a candidate" on a line by itself.

Example: Input File

```
THE QUICK BROWN FOX JUMPED OVER THE LAZY DOGS
THE LAZY DOGS DID NOT SEE THE QUICK BROWN FOX
QUEEN JACKIE WAS VERY PERPLEXED WHEN THE COMPUTER FELLOW BEGAN COUNTING AT ZERO
OSCAR THE GROUCH ZIPPED QUICKLY THROUGH THE TRASH LOOKING FOR SOMETHING TO EAT
```

Output to screen

```
Sentence is a candidate
Sentence is not a candidate
Sentence is a candidate
Sentence is not a candidate
```

Problem 3	**Numbers Game**	5 Points

Program Name: numbers.cpp Input File: numbers.dat

A "psychic network" company wants you to write a program that can tell someone if it is "in the stars" that two people marry. Each person is given a score that is computed by summing up the digits of their birthday (formatted as MMDDYYYY). The match measurement between the two people is measured by computing the difference between the two people's scores. The smaller the absolute value of the difference, the better the match. (Okay, so there is nothing scientific about this, but if it will sell calls for $1.99/minute, the company wants to do it.)

For example, for the birth dates 06141965 and 07031969, the match measurement is computed as follows:
Score #1: 06141965 è 0+6+1+4+1+9+6+5 = 32
Score #2: 07031969 è 0+7+0+3+1+9+6+9 = 35
Match measurement è absolute value (32 – 35) = 3

The following rating shows an even closer match.
Score #1: 05031944 è 26
Score #2: 06221943 è 27
Match measurement è absolute value (26-27) = 1

Input

Input to your program will consist of a series of birth date pairs. The first date in the pair is in columns 1-8. There is a blank in column 9. The second date in the pair is in columns 10-17. There is no other input on the input line. **WARNING:** Some compilers will read integers with leading zeroes as octal (base eight) numbers instead of as decimal (base ten) numbers. Be careful.

Output

For each birth date pair, your program is to print the absolute value (a non–negative number) of the match measurement. The match measurement should start in column 1 and be on a line by itself.

Example: Input file

```
06141965 07031969
05031944 06221943
10111999 10112000
```

Output to screen

```
3
1
26
```

Problem 4	Am I Close?	8 Points

Program Name: close.cpp **Input File: close.dat**

If you have ever reconciled a checkbook, you know that it can be very difficult deciding if you have the correct balance compared to the bank. The problem is that at any given moment, your checkbook will probably include checks that have not arrived at the bank. Therefore, you may know the balance of the bank (including all deposits/credits which are approximately instantaneous) and still not know if you and the bank agree on your account. You still have to determine if there is a combination of outstanding checks that can account for the difference between your bank's balance for your account and the balance in your checkbook. For this problem you only need to consider the five checks most recently written. Although checks usually arrive back at the bank in the approximate order that you wrote them, you are not guaranteed that they will arrive in the exact order that you wrote them.

For example, assume the following:
1. Your bank tells you that your balance is $726.42.
2. Your checkbook tells you that your balance is $347.01.
3. Your last five checks are for $127.43, $16.03, $47.52, $235.95, and $126.54.

You can see that your checkbook can be reconciled with the bank's balance if checks for $127.43, $16.03, and $235.95 have not arrived. Therefore, you would conclude that your checkbook's balance is correct.

However, if you have the following:
1. Your bank tells you that your balance is $9738.42.
2. Your checkbook tells you that your balance is $9584.32.
3. Your last 5 checks that you wrote are for $1284.32, $984.37, $37.29, $349.21, and $198.32.

You can see that your checkbook cannot be reconciled with the bank's balance if you consider only the last five checks. (Of course, you might well go back and recheck your math and more than just the last five checks.)

Input
Input to your program consists of a series of checkbook situations (one situation per input line). Each situation contains exactly 7 integers all in the range (0…999999). The first integer on the line is the balance for the account from the bank (in cents). The second integer on the line is the balance for the account from your checkbook (in cents). The remaining 5 integers on the line are the amounts for the last 5 checks that you wrote. You may assume that all the integers on a line are separated by 1 blank.

Output
For each line of input, your program is to print one of two messages. If the situation can be reconciled, you should print the message "Checkbook reconciles" on a line by itself. Otherwise, you should print the message "Cannot reconcile checkbook" on a line by itself. Your output should contain exactly one message per situation (in the order of the corresponding input situation).

Example: Input File
```
72642 34701 12743 1603 4752 23595 12654
72643 34701 12743 1603 4752 23595 12654
973842 958432 128432 98437 3729 34921 19832
825322 483922 92583 181952 87291 2859 64006
```
Output to screen
```
Checkbook reconciles
Cannot reconcile checkbook
Cannot reconcile checkbook
Checkbook reconciles
```

Problem 5	Your Move	7 Points

Program Name: move.cpp **Input File: move.dat**

In the popular game of solitaire, there are three areas from which cards can be played. The first is the deck from which cards are turned over and played on one of the seven field piles or on one of the four stacks. The field piles are piled such that cards are played in descending rank and alternate between red (hearts and diamonds) and black (spades and clubs). A field pile that has no cards on it can be played upon only with a king. There are four stacks (one for each suit) and cards are played in ascending rank. A stack that has no cards on it can be played upon only with an ace. Aces have the lowest rank, followed in order by the 2-10, jack, queen, and king. Figure 1 shows a solitaire game in progress.

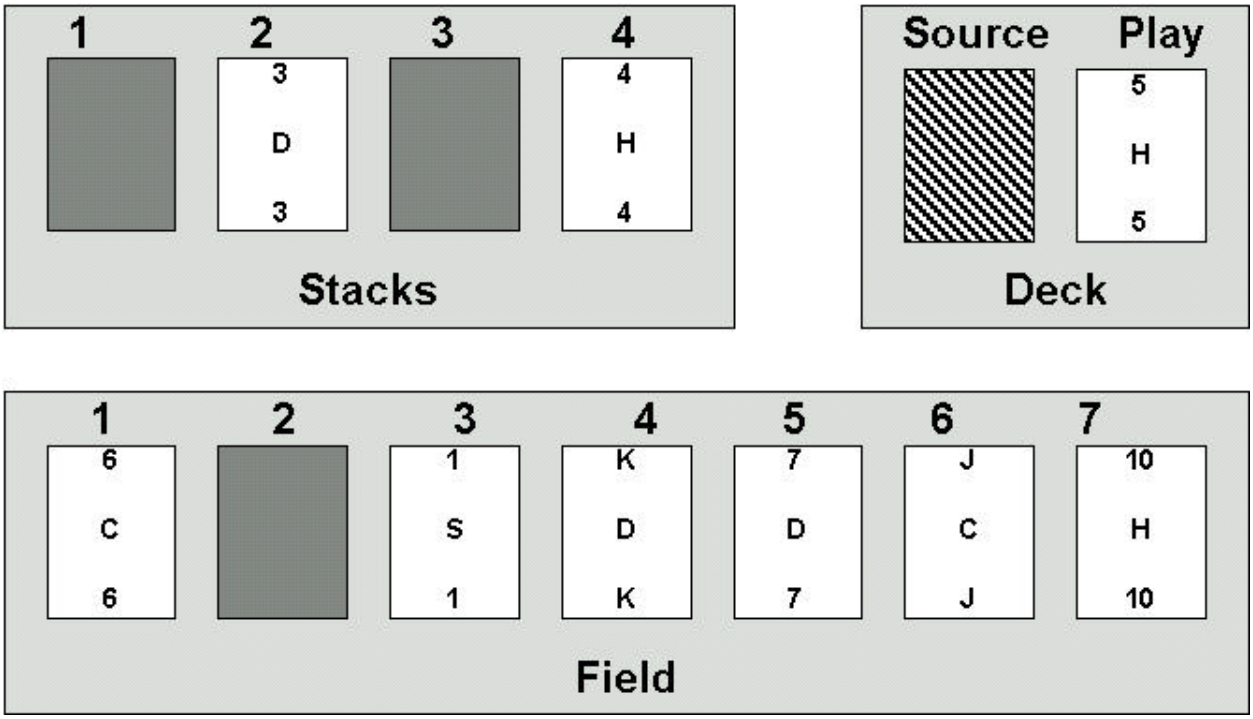

Figure 1: Sample Solitaire Game

Many solitaire card games for computers have an auto-play feature in which the computer will automatically move cards from the field or deck onto the stacks. You are to write an auto-play feature that simply warns the user that a move exists at any given point in the game. Specifically, your program should give an indication (in the order of preference below) if any of the following moves exist. *(Examples for Figure 1 are in italics.)*

1. Move to the stack. A move to the stack exists if:
   - A card is on the play pile of the deck or on any of the field piles such that the rank of the card is exactly one higher than the rank on one of the stacks and the suit is the same for both cards. *5 of hearts can be played on the 4 of hearts on the 2nd stacks pile.*
   - An ace is on the play pile of the deck or on any of the field piles. *Ace of spades from the 3rd field can be played on one of the empty stack piles.*
2. Move to the field. A move to the field exists if:
   - A card is on the play pile of the deck such that the rank of the card is exactly one less than the rank on the target field pile and the suit is of the opposite color (red played on black or black played on red). *5 of hearts can be played on the 6 of clubs in the 1st field pile.*
   - A card is on any of the field piles such that the rank of the card is exactly one less than the rank on the target field pile and the suit is of the opposite color (red played on black or black played on red). *The 10 of hearts in the 7th field pile can be played on the Jack of clubs in the 6th field pile. The 6 of clubs in the 1st field pile can be played on the 7 of diamonds in the 5th field pile.*

- A king is on the play pile of the deck or on any of the field piles and there is an empty field pile. *The king on the 4th field pile can be played on the 2nd field pile.*

Your program is to use the card rank/suit combinations given to it for the top of each of the 12 piles (1 deck, 4 stack, and 7 field) and determine if there is a move to the stack, a move to the field, or no move available. Input to the program will be given with the rank of the card as a one or two digit number from 1 to 13. (Aces are represented by 1, jacks by 11, queens by 12, and kings by 13.) The suit of the card is given as a single letter ("S" for spades, "H" for hearts, "D" for diamonds, and "C" for clubs). Empty piles will be given as "0 N".

Input

Input to your program consists of a series of game snapshots with descriptions of the top card of each of the 12 piles of cards in the game. The first 4 rank/suit pairs are the contents of the 4 piles in the stacks area. The next rank/suit pair is the contents of the play pile of the deck. The last 7 rank/suit pairs are the contents of the 7 piles in the field area. Each rank/suit pair is given as an integer (0-13 for the rank) followed by a single space followed by the suit as a single character (S,H,D,C,N). Each snapshot is on a single line of input.

Output

For each snapshot, your program should determine which move to recommend to the player. If one or more moves to the stacks exist, your program should print the message "Move to Stacks" on a line by itself. If a move to the stacks does not exist, but there are one or more moves to the field that exist, your program should print the message "Move to Field" on a line by itself. If there are no moves to the stacks or to the field, your program should print the message "No Valid Move" on a line by itself.

Example: Input File

```
0 N 3 D 0 N 4 H 5 H 6 C 0 N 1 S 13 D 7 D 11 C 10 H
1 S 0 N 2 C 8 D 11 D 0 N 7 C 9 S 12 H 3 H 5 S 9 C
0 N 0 N 0 N 0 N 9 S 10 H 4 H 2 C 7 C 8 S 12 S 1 D
1 D 0 N 0 N 0 N 9 S 10 H 4 H 2 C 7 C 8 S 12 S 9 C
```

Output to screen

```
Move to Stacks
No Valid Move
Move to Stacks
Move to Field
```

Problem 6	Just the Fax	9 Points

Program Name: fax.cpp **Input File: fax.dat**

The secretarial staff of a particularly large office building has to physically deliver received faxes to the recipients' offices from a central communications room. The secretaries would prefer to have the faxes arranged in a stack so that they can do the least amount of walking. Fortunately, the building's offices are numbered such that if the faxes are sorted by room number, the secretaries can save the wear and tear on their shoes.

Your program is to read in a directory for the building. The directory lists the names of the people in the building (in alphabetical order by last name and then first) along with their office number. To make things easier, names are given with the underscore character substituted for any spaces. Your program should then read the names of the fax recipients and print the order of the stacked faxes (ascending by office number of the recipient). Note that if there is more than one fax for a recipient, you will only print the recipient's name once.

For example, if you were given the building directory:

Office Occupants	Office Number
LUCILLE_BALL	7245
JAN_BRADY	7327
GOOFY_DRIVER	1646
DONALD_DUCK	934
PORKY_FRIEND	6453
WEASER_MANN	6069
MINNIE_MOUSE	3852
ALFALFA_NADER	17
STIMEY_RASCAL	4583
DARLA_ROSE	9275
FROGGY_SINGER	23812
BUCK_WHEAT	582

And the list of recipients was:

WEASER_MANN, BUCK_WHEAT, GOOFY_DRIVER, LUCILLE_BALL, DARLA_ROSE, BUCK_WHEAT

Then your program would print the names of the recipients in the following order:

BUCK_WHEAT
GOOFY_DRIVER
WEASER_MANN
LUCILLE_BALL
DARLA_ROSE

Input

Input to your program starts with the number of occupants (1<=NUM_OCCUPANTS<=100) that indicates the number of occupants in the building directory. The next NUM_OCCUPANTS lines of input each contain a single occupant with the name starting in column 1 of the line, followed by a single blank character, followed by the office number (1<=OFFICE_NUMBER<=999999). There is only one occupant per office number and that occupant's name will appear only once in the directory. Line number (NUM_OCCUPANTS+2) contains a single integer (1<=NUM_FAXES<=100). The next NUM_FAXES lines of input will each contain exactly 1 recipient name on the line by itself starting in column 1. All names consist of up to 30 upper case alphabetic characters. All recipients will have an office in the building directory.

Output

Output from your program consists of a list of the recipient names (each occurring only once) for all recipients of one or more faxes. The list of names should be sorted in ascending order by the office number in which that recipient can be found. Each recipient's name should each be on a line by itself starting in column 1.

Example: Input File

```
12
LUCILLE_BALL 7245
JAN_BRADY 7327
GOOFY_DRIVER 1646
DONALD_DUCK 934
PORKY_FRIEND 6453
WEASER_MANN 6069
MINNIE_MOUSE 3852
ALFALFA_NADER 17
STIMEY_RASCAL 4583
DARLA_ROSE 9275
FROGGY_SINGER 23812
BUCK_WHEAT 582
6
WEASER_MANN
BUCK_WHEAT
GOOFY_DRIVER
LUCILLE_BALL
DARLA_ROSE
BUCK_WHEAT
```

Output to screen

```
BUCK_WHEAT
GOOFY_DRIVER
WEASER_MANN
LUCILLE_BALL
DARLA_ROSE
```

Problem 7	**Too Many Acronyms**	9 Points

Program Name: acronym.cpp Input File: acronym.dat

Personal story from the author

When I first joined International Business Machines Incorporated, I was working on the Space Station Freedom On-Board software. International Business Machines Incorporated was subcontracted by McDonnell Douglas Aerospace Corporation on a contract to the National Aeronautics and Space Administration. I was the technical lead for the System Management Computer Software Configuration Item, which included Fault Detection, Isolation, and Recovery.

Unfortunately, the heavy use of acronyms made it nearly impossible to understand anything I read for the first three months. The following paragraph is a restatement of the above paragraph using the acronyms.

When I first joined IBM, I was working on the SSF On-Board software. IBM was subcontracted by MDAC on a contract to the NASA. I was the technical lead for the SM CSCI which included FDIR.

Clearly the use of acronyms allows written prose to use much less space. But, the saving of space is a moot point if the prose cannot be understood. It is your job to write a program that will read a file of sentences with embedded acronyms and produce updated sentences that substitute the expanded meanings of the acronyms. The following table contains the list of acronyms that your program should recognize.

Acronym	Meaning
NASA	National Aeronautics and Space Administration
IBM	International Business Machines Incorporated
FDIR	Fault Detection, Isolation, and Recovery
CSCI	Computer Software Configuration Item
N2	Nitrogen
O2	Oxygen
CW	Caution and Warning
SDT	Smoke/Fire, Rapid Depression, and Toxic Atmosphere
FSS	Fire Suppression System
SSF	Space Station Freedom
DMC	Daily Maintenance Cycle
FSD	Fire/Smoke Detected
AR	Affected Region
LCV	Loss of Crew and Vehicle
SM	System Management

Your program will read a series of lines of input that are each up to 20 characters in length. Your program is to:

1) Read each line of input.
2) Scan it for acronyms defined in the above table. Acronyms should appear exactly as in the above table and can be delimited by non-alphanumeric characters, the beginning of the line, and the end of the line. For example, "N2 is inert" contains the acronym N2 because it is delimited by the beginning of the line and a space. "H2O2 is a medicine" does not contain an acronym because O2 is delimited by the numeric character "2" in front.
3) Substitute the acronym's meaning into the string. Replace **only** the acronym and leave all other characters, spaces, punctuation, etc.
4) Write the updated string to the screen.

Input

Input to your program consists of a series of lines of input that your program should process using the steps above. You may assume that no line of input contains more than 20 characters but there is no limit to the number of lines of input.

Output

For each line of input, your program should print the updated input line using the above steps. You may assume that no output line will contain more than 80 characters.

Example: Input File

```
The SSF FSS system
must be purged with
N2 during a DMC.
Seepage past the N2
dispersal nozzle
allows a buildup of
O2 in the FSS feeder
lines.  When the SDT
detection system
issues a FSD CW to
FDIR, a command is
issued to the FSS
to disperse N2 to
the AR.  If the FSS
lines have not been
purged, the fire
will be fanned with
O2 instead of being
extinguished with N2
leading to a possible
LCV event.
```

Output to screen

```
The Space Station Freedom Fire Suppression System system
must be purged with
Nitrogen during a Daily Maintenance Cycle.
Seepage past the Nitrogen
dispersal nozzle
allows a buildup of
Oxygen in the Fire Suppression System feeder
lines.  When the Smoke/Fire, Rapid Depression, and Toxic Atmosphere
detection system
issues a Fire/Smoke Detected Caution and Warning to
Fault Detection, Isolation, and Recovery, a command is
issued to the Fire Suppression System
to disperse Nitrogen to
the Affected Region.  If the Fire Suppression System
lines have not been
purged, the fire
will be fanned with
Oxygen instead of being
extinguished with Nitrogen
leading to a possible
Loss of Crew and Vehicle event.
```

Problem 8	**Airline Miles**	3 Points

Program Name: airline.cpp **Input File: airline.dat**

When flying frequently on most airlines, a passenger can join a frequent flier club. Benefits are based on the number of segments and miles that a passenger flies. A segment is a one-way trip regardless of the number of stops on the route. A round trip is two segments. Members are classified as Platinum, Gold or Silver Elite Status when they attain a threshold of segments or miles. If these thresholds convey different status, a member is awarded the higher one. The table below shows the threshold number of segments or miles that a member must have to attain the named Elite Status.

Elite Status	Number of Segments Flown	Number of Miles Flown	Notes
Platinum	120 and more	75000 and more	
Gold	80 - 119	50000 – 74999	
Silver	40 - 79	25000 – 49999	
Member	0 – 39	0 – 24999	

Input

Input to your program consists of a series of input lines with segments/miles pairs. The segments (0<S<=1000) will start in column 1 of the input line. The segments will be followed by a single space followed by the miles (0<M<=1000000).

Output

For each segments/miles pair, your program will print the Elite Status of the person with those segments/miles. Based on the above table, your program will print only the status (Platinum, Gold, Silver, Member) starting in column 1 on a line by itself.

Example: Input File

```
23 44235
17 24342
120 53234
118 74853
```

Output to screen

```
Silver
Member
Platinum
Gold
```