University Interscholastic League

Computer Science Competition

2000 State Programming Set

I. General Notes

1. Do the problems in any order you like.

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

3. Several problems have a reference to the columns of an input line. In these cases "column" refers to the character position on the input line. That is, “column 1” refers to the first character position on the line, and “columns 1-3” refers to the first three positions on the line.

4. Output should contain exactly what is specified by the problem. There should be no extra blank lines, spaces, or debugging output.

5. Input files will contain exactly what is specified. Specifically, there will be no blank lines or extra spaces in the input files.

II. Point Values and Names of Problems

Number	**Name**	**Point Value**
Problem 1	User Profiling	8
Problem 2	Blockbuster Connections	11
Problem 3	Variable Key String Sort	5
Problem 4	Kangaroo Checks	4
Problem 5	Bouncing Marbles	9
Problem 6	Consonant Blends	5
Problem 7	Change for a Peso?	6
Problem 8	Are we there yet?	3
Total		**51**

Problem 1

User Profiling

8 Points

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

The software development project that you are working on needs someone to write a program that will parse a transaction file and calculate some user profile statistics. Table 1 contains a list of the “activities” that your program will parse and interpret in order to compute the statistics listed in Table 2. Note that all statistics are rounded to the nearest millisecond.

Activities	Meaning	Example
LOGIN	The user is starting a new session.	`00784932 LOGIN izzod`
TRANS	The user is executing a transaction.	`00923523 TRANS izzod`
LOGOUT	The user is terminating a session.	`01230012 LOGOUT izzod`

Table 1: User Activities in the Transactions File

The format for every transaction includes an eight-digit timestamp which represents the number of milliseconds elapsed since midnight. Therefore, the time will range from 00000000 to 86399999. Every transaction also contains the associated user ID which is made up of between 5 and 8 lower case alphabetic characters.

Some rules about the content of the input file.

1. There will be no TRANS activities for any user ID such that there is not a currently active session.
2. There will be no LOGIN activities for any user ID with a currently active session.
3. There will be no LOGOUT activity for any user without a currently active session.
4. Each LOGIN activity will have a corresponding LOGOUT activity.
5. The activities are in chronological order.
6. The file contains exactly one day’s activity such that the times will start as early as 00000000 and will not exceed 86399999.
7. The computer is slow enough that no two activities will occur with the same timestamp.
8. Users may log out and then log into subsequent sessions later in the day.
9. No user will have more than 1 session active at any given time.
10. The input sequence is a list of successful activities by the user and therefore contains no data that violates these input rules.
11. The number of activities in the file is limited only by the number of milliseconds in the day.
12. It is possible that the user simply logs in and then logs out without any transactions.

Statistic	Example Computed Data	How the Statistic is Computed
Average Session Length (**ASL**) (rounded down to nearest integer)		$\dfrac{\sum_{i=1}^{N}(\texttt{LOGOUT\_time}_i - \texttt{LOGIN\_time}_i)}{N}$ Where N is the total number of sessions in the activities file.

Mean Time Between Activities for the same user (**MTBA**) (rounded down to nearest integer)	`01921520 LOGIN izzod`
	`01976089 TRANS izzod`
	`01997011 LOGIN isnot`
	`02193028 TRANS izzod`
	`02243853 TRANS izzod`
	`02353252 LOGOUT izzod`
	`02418238 TRANS isnot`
	`02428132 LOGOUT isnot`
	
	izzod’s session time = 431732
	isnot’s session time = 431121
	
	MTBA=((431732/2)+(431121/1))/ 2 = 323493
		$\dfrac{\sum_{i=1}^{N}(\texttt{LOGOUT\_time}_i - \texttt{LOGIN\_time}_i)/T_i}{N}$
		 Where
		 N is the total number of sessions in the activities file, and
		 T_i is the total number of transactions in the activities file for session i.
		 If the user does not conduct any activities in the session, the session is completely ignored when computing this statistic.

Longest Session Length (**LSL**)		The LSL is the length of the longest duration session.
Shortest Session Length (**SSL**)		The SSL is the length of the shortest duration session.

Table 2: Statistics Your Program is Required to Compute

Input

Your program is to read in all activities from the activities file. The input file consists of a single day of activities with one activity per line in the file. Columns 1-8 contain the time of the activity in milliseconds since midnight (with leading zeroes). Column 9 contains a blank. One of the activities from Table 1 starts in column 10. The activity is followed by a single space and then by the user ID associated with the activity. There will be no extraneous input, blank lines, or spaces in the file.

Output

Output from your program consists of exactly four lines of output to the screen. The first line should contain the ASL; the second line should contain the SSL; the third line should contain the LSL; and the fourth line of the output should contain the MTBA. Your program should follow the output format in the Example Output below.

Example: Input File

```
00784932 LOGIN izzod
00923523 TRANS izzod
00939482 TRANS izzod
01023458 LOGIN lizard
01080384 LOGOUT izzod
01103848 LOGIN stevie
01203902 TRANS lizard
01213842 LOGIN izzod
01218293 TRANS izzod
01223483 LOGIN unused
01228293 TRANS stevie
01230012 LOGOUT izzod
02348203 TRANS lizard
03020389 TRANS stevie
03123834 LOGOUT unused
03289893 TRANS stevie
03349583 LOGOUT lizard
04038208 LOGOUT stevie
```

Output to screen

```
ASL=1494491
SSL=16170
LSL=2934360
MTBA=576269
```

Problem 2 | **Blockbuster Connections** | **11 Points**

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

There is a television gameshow called “Blockbusters” in which opposing contestants try to answer questions building a path across a board to win a game. See the Samples of the board in Figure 1 below. A “Family Pair” composed of two relatives must build a contiguous path from left to right before a “Solo Player” builds a contiguous path from top to bottom. The contestants build a path by correctly responding to a question whose answer begins with the letter in the selected box. Every question is open to all players (who buzz in).

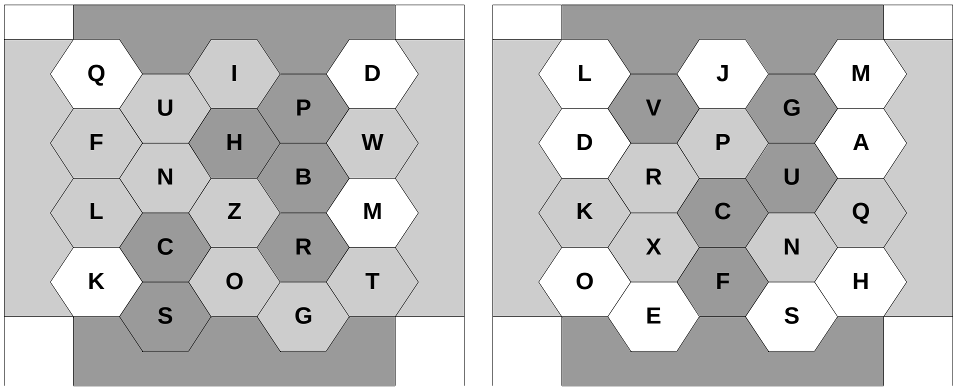

Figure 1: A Family Pair Win (Left) and a Solo Player Win (Right)

Your job is to write software that, given a board layout and a list of letters correctly answered by the contestants, determines if

1. The Solo Player Wins
2. The Family Pair Wins
3. No Winner Yet

You can see from the above examples, that the Family Pair won the game on the left with the Path LNZOGT. (In this case, other winning paths exist for the Family Pair.) The Solo Player won the game on the right with the Path GUCF. Note that not all blocks that are won need be used in the path to win. Further, the nature of the board prevents a path from top-to-bottom and a path from left-to-right from existing at the same time. Therefore, there can be only one winning path.

Your program will read in a board configuration as a single line of 20 characters (upper case alphabetic) where the order of the characters is in the order seen in Figure 2. No character is used twice and (obviously) six of the characters are unused. The board configuration component of the input line is terminated by a single colon.

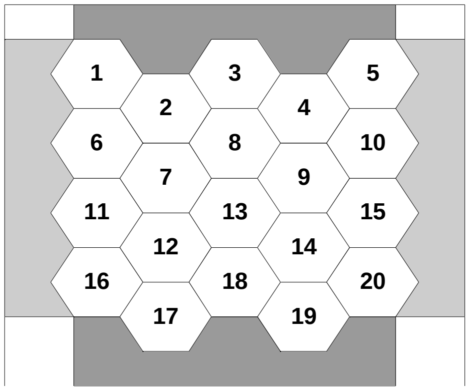

Figure 2: Location of the Input Characters on the Blockbuster Board

You program will then read a series of up to 20 characters which are the results of the contested questions to “test” the board for a winner. The Solo Player’s blocks are in lower case; the Family Pair’s blocks are in upper case. You are guaranteed that no character appears more than once and that all characters that do appear in the contested question results can be found on the board configuration. The end of the “test” will always end with single colon.

From Figure 1, the two input lines would be

```
QUIPDFNHBWLCZRMKSOGT:scNLFUhIWpbrGTOZ:
LVJGMDRPUAKXCNQOEFSH:vKXQfcPRNug:
```

And these two input lines would result in the output
```
“Family Pair Wins”
“Solo Player Wins”
```

But it is possible that your program will find that no one has won yet as is the case with the line
```
QUIPDFNHBWLCZRMKSOGT:scNLFUhIWpbrGT:
```

In which case your program would print the output
```
“No Winner Yet”
```

Input
Input to your program consists of a series of lines each of which contain a single test. Each input line contains 20 alphabetic characters prescribing the board configuration as described above. Column 21 contains a single colon. The line then contains between zero and 20 characters prescribing the results of the contested questions. The last column on the line contains a single colon.

Your program should continue reading test cases until it reaches the end-of-file. The file will not contain any extraneous or erroneous input (such as blank lines, embedded spaces, invalid characters).

Output
For each line of input, your program should print one of the following lines of output on a line by itself to the screen: `“Family Pair Wins”, “Solo Player Wins”, or “No Winner Yet”`. Your output should not print the quotations nor any other extraneous output and the output results should start in column 1.

Example: Input File
```
QUIPDFNHBWLCZRMKSOGT:scNLFUhIWpbrGTOZ:
LVJGMDRPUAKXCNQOEFSH:vKXQfcPRNug:
QUIPDFNHBWLCZRMKSOGT:scNLFUhIWpbrGT:
```
Output to screen
```
Family Pair Wins
Solo Player Wins
No Winner Yet
```

Problem 3 | **Variable Key String Sort** | **5 Points**

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

Although most string sorts are done based on the order assigned to characters in the English language, it is possible that other languages or codes assign a different order to the characters. For this problem, you are to write a program that will sort up to 30 strings based on a different ordering of the alphabetic characters.

Your program is given the new ordering of the characters in the alphabet as a single string of the characters in the order (ascending -- from first to last) being prescribed. Your program is then given up to 30 strings which are up to 10 characters in length each. Your program is required to sort the strings in ascending order.

For example, if your program is given the following alphabetic ordering
QWERTYUIOPASDFGHJKLZXCVBNM

Then it would sort the words “HONEYBEE”, “HONEY”, “SPIDER”, “SPITE”, “WEB”, and “RAIDERS” as follows.
WEB
RAIDERS
SPITE
SPIDER
HONEY
HONEYBEE

Note that the words “HONEY” and “HONEYBEE” are sorted such that two words which are equal up to the length of the shorter word will be ordered such that the shorter word will be considered to have the lesser value of the two.

Input
The first line of input to your program consists of exactly 26 alphabetic characters. The characters on this line define the proper ordering of the character set you should use as a basis for sorting. All characters of the English alphabet appear exactly once and are in upper case. The remaining lines in the input file each contain exactly one word which should be sorted according to the ordering of the first line of the file. The words to be sorted are all in upper case and are between 1 and 10 alphabetic characters in length. There are between 1 and 30 words to be sorted and there are no embedded lines or blanks anywhere in the input file.

Output
Output from your file will consist of the words on the lines after the first line of input in the input file. These words should be sorted in ascending order per the rules described above. Your output should not contain any extraneous output including blank lines or embedded spaces.

Example: Input file
```
LAKSJDHFGPQOWIEURYTMZNXBCV
LAKESIDE
RED
EDUCATION
I
FORTUNE
SORTING
VLASIC
VLADIMIR
SCHOLZBOT
MORK
GROOVY
FONZIE
ORSON
ORK
COMPULSORY
```

Output to screen
```
LAKESIDE
SORTING
SCHOLZBOT
FORTUNE
FONZIE
GROOVY
ORK
ORSON
I
EDUCATION
RED
MORK
COMPULSORY
VLASIC
VLADIMIR
```

Problem 4

Kangaroo Checks

4 Points

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

At the end of each day, all banks must reconcile each account in its books. Reconciliation of an account is done by adding all the deposits to the starting balance and then subtracting all the debits to the account. Since the bank does not record the time that deposits and debits are made, the bank will give the benefit of the doubt to its customers so long as the ending balance for the day is not negative.

You are to write a program that reads the starting balances for up to 20 accounts followed by up to 200 transactions. Your program is supposed to determine which, if any, accounts end in negative balances (<0) at the end of the day. Any such account must be reported for further investigation. (The low number of accounts and transactions is because this is just a test of your program.)

Input

Input to your program will start with a single line containing the number of accounts at the bank (1<=A<=20). The following A lines each contain exactly 2 integers. The first integer, representing the account number, starts in column 1 and is exactly 6 digits in length with leading zeroes as necessary. The second integer is the starting balance of the account (in cents). The balance and all transactions are stored and processed as integers (in cents) because mathematical operations on floating point types are substantially more expensive than mathematical operations on the integer type. Starting balances may be positive or negative. Positive balances do not have a sign, but negative starting balances have a leading '-' sign.

Following the list of accounts and their starting balances are up to 200 transactions. Transaction lines start with a single character action field in column 1 ('C' for credit and 'D' for debit) followed by a blank in column 2 followed by the six-digit account number (with leading zeroes) followed by a blank in column 9 followed by the transaction amount (in cents) Your program should continue reading transactions until it reaches the end-of-file. You can be assured that there are no errors in the input file nor any extraneous spaces or blank lines.

Output

After reading all transactions, your program should print a list of accounts that end the day with negative balances. Each entry in the list should be the account number in columns 1-6 (with leading zeroes as necessary) followed by a single blank in column 7 followed by the balance of the account starting in column 8. Balances are shown in dollar format with parentheses around them to indicate a negative balance. For example, a balance of –42748 cents would be shown as ($427.48) because it is negative and has a decimal point because the balance is expressed in dollars. Your program should print both the parentheses and the dollar sign. The accounts should be ordered in the same order that they appeared in the input file relative to one another.

Example: Input File
```
12
322525 -803
353839 23923
484595 409439
002305 82385
103592 -9239
268358 190909
352920 58285
235235 2948234
000102 28523
253093 -123
111111 100000
325001 32952
C 268358 10293
D 322525 35252
D 353839 45823
D 111111 23253
D 353839 23455
D 268358 5000
D 111111 253998
D 322525 5000
D 268358 19382
D 322525 19203
D 352920 85492
C 268358 1092
D 268358 28345
C 322525 92385
C 111111 23902
D 353839 32523
C 353839 123952
D 111111 23528
C 103592 29395
D 353839 90934
D 322525 23852
C 353839 2535
C 268358 17398
C 322525 2359
D 353839 83499
```
Output to screen
```
353839 ($1258.24)
352920 ($272.07)
253093 ($1.23)
111111 ($1768.77)
```

Problem 5

Bouncing Marbles

9 Points

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

In this problem, you are to write a program that will determine the path of a marble as it bounces through a maze. The maze consists of a 10 by 10 set of cells and marbles only travel in exactly 1 of 4 directions: toward the top, toward the bottom, toward the left, or toward the right. Cells in the maze can contain the structures in the following table.

Cell Structures		**Where the marble goes if it approaches from the…**			
		Left	**Right**	**Bottom**	**Top**
.	Empty cell. The marble travels through this cell in the same direction.	Right	Left	Top	Bottom
/	Angle #1. The marble bounces at 90 degrees.	Top	Bottom	Right	Left
\	Angle #2. The marble bounces at 90 degrees.	Bottom	Top	Left	Right
O	Blocked cell. The marble bounces back in the direction from which it came.	Left	Right	Bottom	Top

Table 3 : Description of Cell Structures in the Maze

Your program will be given a 10 by 10 maze. Your program is to determine if the maze has the potential to trap a marble (imperfect maze) or if there is **no** position/direction combination on the maze that can keep the marble from bouncing off of the maze (perfect maze). For example, you can see from the maze below that a marble that starts at row 0 column 2 (0,2) and is traveling toward the bottom of the maze will:

- Bounce toward the right at (1,2),
- Bounce toward the bottom at (1,7),
- Bounce toward the left at (4,7),
- Bounce toward the bottom at (4,0), and
- Exit the maze from position (9,0).

```
          Columns
        0123456789
      0 .O....OOO.
      1 ..\....\O.
      2 ...O./....
      3 ..//......
 Rows 4 /....../.O
      5 .........O
      6 ./...O...O
      7 ..O.../..O
      8 ......./..
      9 ..OOOOOO/.
```

Figure 3: Sample Imperfect Maze

Likewise, a marble that enters from (8,2) traveling toward the right will:

- Bounce toward the top at (8,7),
- Bounce toward the right at (4,7),
- Bounce toward the left at (4,9),
- Bounce toward the bottom at (4,7),
- Bounce toward the left at (8,7), and
- Exit the maze from position (8,0).

Finally, a marble that enters the maze from (2,6) traveling toward the bottom will:

- Bounce toward the left at (7,6),
- Bounce toward the right at (7,2),
- Bounce toward the top at (7,6),
- Bounce toward the bottom at (0,6),
- Bounce toward the left at (7,6), and
- Continue bouncing forever (an imperfect maze).

A perfect maze does not have the potential to strand a marble bouncing forever on it. Therefore, the maze in Figure 3 is an imperfect maze. It is left to you to verify that the maze in Figure 4 is a perfect maze because there is no way to strand a marble on it (i.e. all marbles will exit).

```
  0123456789
0 ......\...
1 ..........
2 .../...O..
3 ......\...
4 ..O.......
5 ....O.....
6 ../.../...
7 ........O.
8 ..../.....
9 ..\..O....
```

Figure 4: Sample Perfect Maze

Input

Input to your program will consist of a series of mazes. Each maze consists of exactly 10 lines each with 10 columns. Each line is made up of only those characters in **Table 3**. Your program will read in the mazes one at a time and determine if it is a perfect maze (marbles cannot be trapped) or an imperfect maze (marbles can be trapped). You can be assured that all mazes in the input file are correct and complete. There will be exactly one blank line after each maze (and no other blank lines anywhere in the input file). There will be no extraneous input or embedded or trailing spaces. Your program should read to the end of file.

Output

For each maze from the input file, your program should print the message "Perfect Maze" if the maze cannot trap a marble. If the maze can trap a marble, your program should print the message "Imperfect Maze". Messages should begin in column 1. There should be no extraneous output including embedded or trailing blanks and blank lines.

Example: Input File

```
.O....OOO.
..\....\O.
...O./....
..//......
/....../.O
.........O
./...O...O
..O.../..O
......./..
..OOOOOO/.

......\...
..........
.../...O..
......\...
..O.......
....O.....
../.../...
........O.
..../.....
..\..O....
```

```
..........
..........
..........
.../.\....
..........
...\./....
..........
..........
..........
..........
```

Output to screen

```
Imperfect Maze
Perfect Maze
Imperfect Maze
```

Problem 6 | **5 Points**

Consonant Blends

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

It is common that computer programs count the number of words, number of letters, etc. in the English language. In this problem, you are to write a program which counts the most common two-letter consonant blends in the English language. A two-letter consonant blend is any combination of two consonants that appear contiguous in an English word. Some notes about consonants blends:

· consonants that are separated by anything (spaces, punctuation, other letters) are not consonant blends
· ordering matters when distinguishing between consonant blends (“st” is not the same thing as “ts”)
· case does not matter in distinguishing consonant blends (“st” is the same as “St” and “ST” and “sT”)
· a consonant is all letters in the English alphabet except for “AEIOU” (in this problem, “y” is a consonant)

Your program will read text from an input file and determine the 5 most frequently occurring consonant blends from the text. In case of a tie, the consonant blends should be printed in alphabetical order.

Input

Input to your program consists of free-form text such that any printable ASCII character can appear. Your program should read the entire input file finding and counting the consonant blends in it.

Output

Output from your program will consist of exactly 5 lines containing the most frequently occurring consonant blends sorted in descending order of frequency. The first line lists the most commonly occurring consonant blend in columns 1 and 2 followed by a blank in column 3 followed by the number of occurrences of the consonant blend left-justified starting in column 4 with no leading zeroes. The second through fifth lines list the second through fifth most commonly occurring consonant blends in the same format as the first line. All consonants in the output file must be in lower case. You may assume that there are at least five consonant blends in the input file.

Example: Input File

```
LUBBOCK, Texas (Ticker) -- Texas Tech marched into its second straight "Sweet
16" as freshman Plenette Pierson led four players in double figures with 19
points en route to a 76-59 victory over Tulane in the Mideast Region.

Aleah Johnson, Katrisa O'Neal and Tanisha Ellison had 12 points apiece for
third-seeded Texas Tech (27-4), which plays No. 2 Notre Dame on Saturday in
Memphis, Tennessee.

The other "Sweet 16" matchup at Memphis will be top-seeded Tennessee and No.
4 Virginia.

Conference USA Player of the Year Grace Daley scored 24 points for Tulane
(27-5), which failed to set a school record for victories in a season. The
Green Wave have never advanced to the regionals.

The Lady Raiders used a pair of first-half runs to put away the Green Wave.
Ellison scored seven points during an 11-0 burst that gave Texas Tech a 22-16
lead with 7:24 to play before the break. But the key run started 2:48 before
intermission, when Pierson ignited a half-ending, 10-0 run with a jumper that
made it 36-24. Amber Tarr hit a 3-pointer with 24 seconds left and Keitha
Dickerson added one at the buzzer after a steal by Pierson.
```

Output to screen

```
th 19
ch 8
rs 8
nt 7
nd 6
```

Problem 7

Change for a Peso?

6 Points

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

It is common for U.S. citizens travelling abroad to have to spend U.S. dollars in shops, restaurants and with street vendors. However, it is very common that the ratios of the denominations that the U.S. citizen has in dollars will not match the denominations that the person in the foreign country has in his country's monetary units. Your program is supposed to help people from the U.S. who are visiting Mexico. Your program is given the following data:

- the numbers of pennies, nickels, dimes, quarters, $1 bills, and $5 bills possessed by the U.S. citizen,
- the numbers of 1 new peso(NP), 2 NP, 5 NP coins and the numbers of 10, and 20 new pesos bills possessed by the local vendor in Mexico, and
- the value in U.S. cents of 1 new peso.

The U.S. citizen who is visiting Mexico is seeking to purchase the largest number of New Pesos that he can but without any loss to himself or the vendor due to rounding errors. So, the goal of your program is to determine the **maximum** number of U.S. dollars that can be exchanged equitably with the vendor. It may be possible that the U.S. citizen has $11 that he can exchange for 68NP. But, if the vendor happens to have correct NP denominations to make change for $27, the visitor will make the $27 exchange.

Input

Input to your program consists of a series of lines of input that each describe a potential exchange. Each line of input contains the values in Table 4 below. There are no errors or extraneous input in the input file.

Field Name	Value Range	Columns	Description
# pennies	0-5	1	Number of pennies that the U.S. visitor possesses.
# nickels	0-5	3	Number of nickels that the U.S. visitor possesses.
# dimes	0-5	5	Number of dimes that the U.S. visitor possesses.
# quarters	0-5	7	Number of quarters that the U.S. visitor possesses.
# $1 bills	0-5	9	Number of $1 bills that the U.S. visitor possesses.
# $5 bills	0-5	11	Number of $5 bills that the U.S. visitor possesses.
# 1 NP coins	0-5	13	Number of 1 NP coins that the vendor possesses.
# 2 NP coins	0-5	15	Number of 2 NP coins that the vendor possesses.
# 5 NP coins	0-5	17	Number of 5 NP coins that the vendor possesses.
# 10 NP bills	0-5	19	Number of 10 NP bills that the vendor possesses.
# 20 NP bills	0-5	21	Number of 20 NP bills that the vendor possesses.
Value of a NP	1-999	23-25	Value of a New Peso in U.S. cents.

Table 4 : Format of the Input File

Output

For each input line, your program should print the maximum amount of U.S. dollars and cents that can be equitably exchanged between the visitor and vendor on a line by itself. The dollar sign (required) should appear in column 1 immediately followed by the dollars and cents (d.cc format). See the sample output for an illustration of the required output format. If there is no combination of the U.S. and Mexican monetary denominations that can be exchanged, your program should print the message "No monetary exchange" starting in column 1. Your program should not contain any extraneous output including embedded blanks or blank lines. Note that your program will have only 2 minutes to solve all of the input file. (The exhaustive solution written for the judges easily meets this requirement on a 200 MHz Pentium.)

Example: Input File

```
4 5 5 5 5 0 3 3 3 0 0 016
5 5 5 5 5 5 5 5 5 5 5 045
1 1 1 1 1 1 1 1 1 1 1 019
4 3 5 0 3 2 5 3 5 2 3 123
```

Output to screen

```
$3.84
$31.95
No monetary exchange
$13.53
```

Problem 8

Are we there yet?

3 Points

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

Every parent has heard the question, "Are we there yet?" countless thousands of times. If you are Homer Simpson, you simply answer "No!" countless thousands of times. On behalf of all other parents, you are going to write a program that will estimate an answer to the follow-up question, "How much longer?"

The families are traveling along U.S. interstates which have mile markers alongside the road. The values on the mile markers represent how far (in miles) the marker is from the origin or entry of the interstate highway into the state you are in. Your program will use collected data to estimate "how much longer" (in hours, minutes, and seconds) to the destination. Your program is given two data points each with a time (expressed in military time) and a mile-marker that was passed at that time. Your program is then given a mile marker that is the destination. Your program will then calculate "how much longer" it will take to reach the destination mile marker. You may safely make the following assumptions:

1. The family only travels along interstate highways.
2. All points referenced by your program will be within a single state.
3. The family only travels during a single day and therefore all times can be represented as hours, minutes, and seconds. (Your program will not have to consider progress samples taken on different days.)
4. The family can travel either direction on an interstate (so that mile markers may count up or down as the family progresses).
5. The family will never travel at such high speed that they can pass two mile markers at the same second.
6. All times used by your program have been adjusted appropriately for changing time zones etc.

For example, if your program was given:
1. the family passed mile marker 417 at 10:45:17
2. the family passed mile marker 278 at 13:12:27
3. the family's destination was mile marker 116

Your program should determine that it will take 2 hours, 51 minutes and 31 seconds (expressed 02:51:31) to reach mile marker 116 from mile marker 278.

Input

Input to your program consists of a series of "Are we there yet?" tests. Each input line describes exactly one test. A test consists of two points (mile marker and time the mile marker was passed) and a destination mile marker. Test point #1 starts with an integer for the mile marker at point 1 (in column 1) followed by a single blank followed by the military time the point was passed in the format hh:mm:ss (including the colons). The input line then contains a blank followed by the details for point #2 (which are in the same format as point #1). The input line then contains a blank followed by the mile marker of the destination. You should assume that the family passes point #1 and then passes point #2. You should also assume that the input file is correct and does not include any extraneous input including embedded or trailing blanks nor embedded or trailing empty lines.

Output

For each input test, your program should print the amount of time that it will take for the family to drive from point #2 to the destination mile maker. The output should be in the format "hh:mm:ss" with leading zeroes as necessary to maintain the 2-digit format of each time element.

Your program should print only the amount of time to go and nothing else. See the sample output below.

Example: Input File
```
110 12:00:00 310 14:00:00 610
417 10:45:17 278 13:12:27 116
27 01:25:44 96 02:21:59 816
```
Output to screen
```
03:00:00
02:51:31
09:46:57
```