UNIVERSITY OF NOTRE DAME
Association for Computing Machinery
Programming Contest

Problem Sets

26 March 1994

There are 7 problems, given in order of increasing difficulty. Please notify a judge immediately if you do not have all seven problems. There are 19 labeled pages in this handout.

If you have a problem of any kind, please notify a judge immediately.

Judges will NOT give out helpful hints on any of the programs. We will clarify any questions that you have about the problem, but no help will be given. Please see the handout entitled "Official Rules" for a further explanation.

For programming purposes, you may assume that all data will be of the format specified in the problem specification. That is, no invalid data will be used to judge your program.

NOTE: *No* output should have a blank final line!

1 Heads / Tails Probability

The probability of n heads in a row tossing a fair coin is 2^{-n}. Calculate the probability for $n = 100, 150, \ldots, 300$.

1.1 Input

None.

1.2 Output

Print a table of n and 2^{-n} in the following for the given values of n, using the following format:

```
2^-100 = x.xxxe-y
2^-150 = x.xxxe-y
2^-200 = x.xxxe-y
2^-250 = x.xxxe-y
2^-300 = x.xxxe-y
```

where each `x` is a decimal digit and each `y` is a decimal integer with no leading zeros or spaces.

2 Wild Thing

Command interfaces to modern operating systems often include a number of *metacharacters* which provide powerful features that allow a user to accomplish a lot of work without resorting to traditional programming. One of the most common metacharacters is a *wildcard* character.

Wildcard characters are used within filenames to permit the user to generically specify a group of files that a command should act upon. A wildcard character itself matches zero or more characters in the same position as the wildcard.

Suppose a user wanted to print all files ending in `.c`, delete all files beginning with `old_`, and copy *all* files (no matther what they are named) to a floppy disk. With wildcard metacharacters, all of this can be done using only three commands instead of using a separate command for each file involved. The metacharacter sequences for these commands would be as follows:

```
*.c
old_*
*
```

Write a program that will accept an input pattern which may contain wildcard characters and decide whether or not each of the filenames that follow it matches the pattern.

2.1 Input

The input file will contain a series of datasets. Each dataset will consist of a pattern line followed by a list of filenames to compare against the pattern. For the purposes of this program, a filename may consist of any group of one to twenty characters, not including asterisks. None of these characters should receive special treatment (periods, for example, should behave as any other characters matched).

The pattern line may include zero or more wildcard characters. A wildcard character will be represented by an asterisk (`*`).

The list of filenames will appear one per line following the pattern line. There will be an unknown number of filenames in the list.

All lines of input will be one to twenty characters in length, including the pattern line. The input file will contain an unknown number of datasets, each separated by a blank line.

2.2 Output

The output for this program will be a short report showing the pattern and the filenames from the subsequent list that match the pattern. Filenames that do not match the pattern should not be printed out. The results from each input dataset should be separated by two blank lines. See the example for the exact format.

2.3 Example

Suppose the input file consisted of the three datasets:

```
C*AT
COMFILE.DAT
COST.DATA
CAT
%X_CAT
COAT
CATCH

*A*
MOUNTAIN.TXT
ALFRED
PROG1A
SECOND.ED
PROG1A.PAS

HELLO
GHOST
NOTHING
TELEPHONE

B**N*
NIBBLE.BIT
BANANA
BNXJ.25
BORN
ABNORMAL.LIS
BRANDISH.SRD
BITNET
```

Your output listing should look like this:

```
MATCHES FOR THE PATTERN: C*AT
COMFILE.DAT
CAT
COAT

MATCHES FOR THE PATTERN: *A*
MOUNTAIN.TXT
ALFRED
PROG1A
PROG1A.PAS
```

```
MATCHES FOR THE PATTERN: B**N*
BANANA
BNXJ.25
BORN
BRANDISH.SRD
BITNET
```

Notice that "`MATCHES FOR PATTERN: HELLO`" was not displayed, because there were no matches in its filename list.

THE FOLLOWING DESCRIPTION REFERS TO PROBLEMS 3, 4, AND 5

Points in Figures

Given a list of figures (rectangles, circles, and triangles) and a list of points in the x-y plane, determine for each point which figures (if any) contain the point.

Input

There will be $n(\leq 10)$ figures descriptions, one per line. The first character will designate the type of figure ("r", "c", "t" for rectangle, circle, or triangle, respectively). This character will be followed by values which describe that figure.

- For a rectangle, there will be four real values designating the x-y coordinates of the upper left and lower right corners.
- For a circle, there will be three real values, designating the x-y coordinates of the center and the radius.
- For a triangle, there will be six real values designating the x-y coordinates of the vertices.

The end of the list will be signalled by a line containing an asterisk in column one. The remaining lines will contain the x-y coordinates, one per line, of the points to be tested. The end of this list will be indicated by a point with coordinates `9999.9 9999.9`; these values should not be included in the output. Points will not coincide with a figure border.

Output

For each point to be tested, write a message of the form:

```
Point i is contained in figure j
```

for each figure that contains that point. If the point is not contained in any figure, write a message of the form:

```
Point i is not contained in any figure
```

Points and figures should be numbered in the order in which they appear in the input.

Example

Here is a sample input set:

```
c 8.5 5.1 8.5
r 8.5 17.0 25.5 -8.5
c 10.4 -3.4 3.4
t 10.2 3.4 34.0 13.6 28.9 5.1
*
0.0 0.0
6.8 5.1
11.9 11.9
17.0 10.2
17.0 25.5
15.3 4.25
9999.9 9999.9
```

See figure 1 for a diagram of these figures and data points.

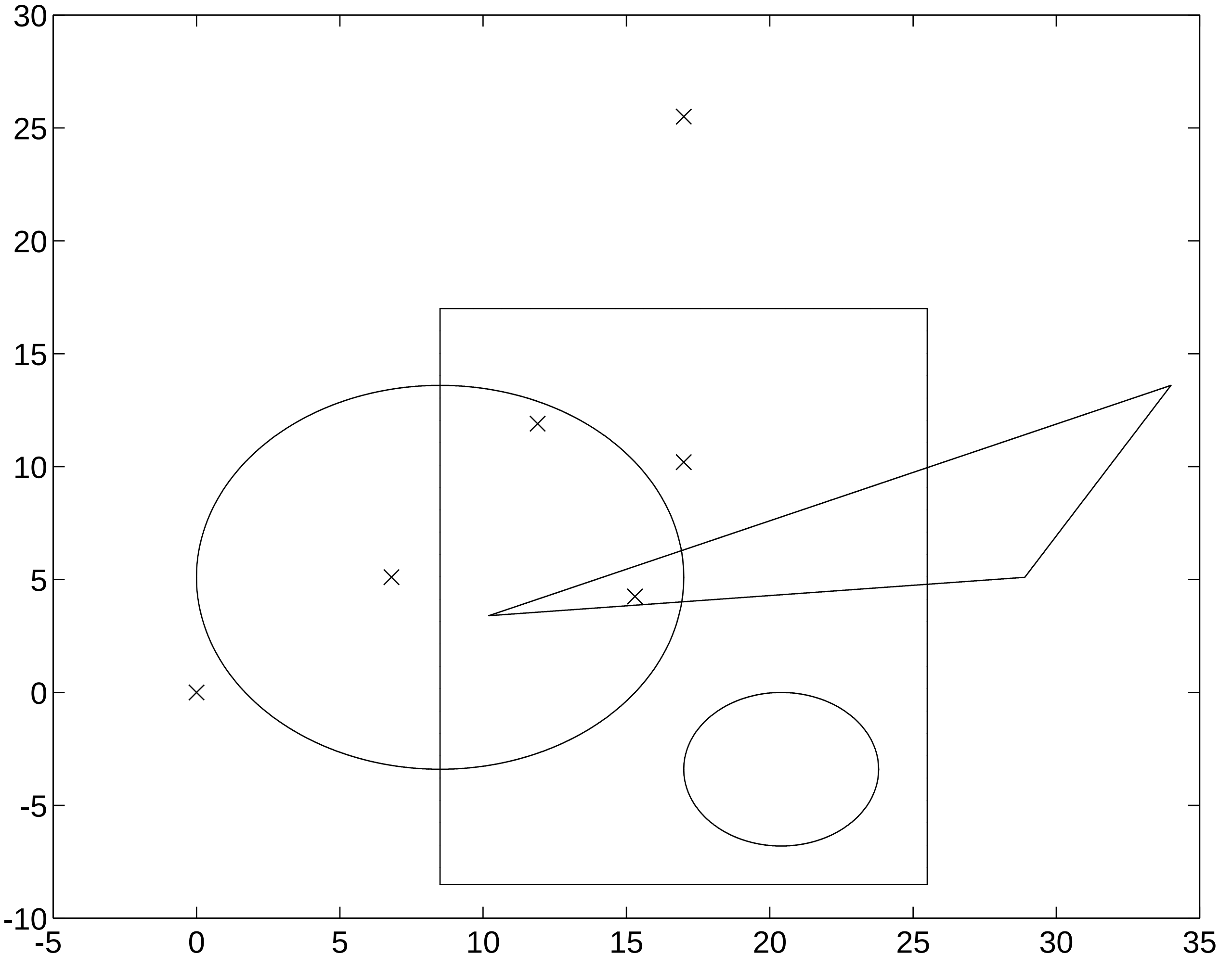

Figure 1: Diagram of input points

This input produces the following output:

```
Point 1 is not contained in any figure
Point 2 is contained in figure 1
Point 3 is contained in figure 1
Point 3 is contained in figure 2
Point 4 is contained in figure 2
Point 5 is not contained in any figure
Point 6 is contained in figure 1
Point 6 is contained in figure 2
Point 6 is contained in figure 4
```

3 Points in Figures: Rectangles

For this problem, implment the Points in Figures scenario with only rectangles. That is, rectangles will be the only type of figure in the input file. Here is a sample input file:

```
r 8.5 17.0 25.5 -8.5
r 0.0 0.0 5.5 10.3
r 2.5 2.5 12.5 12.5
*
2.0 2.0
4.7 5.3
6.9 11.2
20.0 20.0
17.6 3.2
-5.2 -7.8
9999.9 9999.9
```

The above input file would produce the following output:

```
Point 1 is contained in figure 2
Point 2 is contained in figure 2
Point 2 is contained in figure 3
Point 3 is contained in figure 3
Point 4 is not contained in any figure
Point 5 is contained in figure 1
Point 6 is not contained in any figure
```

4 Points in Figures: Rectangles and Circles

Implement the Points in Figures scenario with rectangles and circles. That is, rectangles and circles will be the only types of figures in the input file. Here is a sample input file:

```
r 8.5 17.0 25.5 -8.5
c 20.2 7.3 5.8
r 0.0 0.0 5.5 10.3
c -5.0 -5.0 3.7
r 2.5 2.5 12.5 12.5
c 5.0 15.0 7.2
*
2.0 2.0
4.7 5.3
6.9 11.2
20.0 20.0
17.6 3.2
-5.2 -7.8
9999.9 9999.9
```

The above input file would produce the following output:

```
Point 1 is contained in figure 3
Point 2 is contained in figure 3
Point 2 is contained in figure 5
Point 3 is contained in figure 5
Point 3 is contained in figure 6
Point 4 is not contained in any figure
Point 5 is contained in figure 1
Point 5 is contained in figure 2
Point 6 is contained in figure 4
```

5 Points in Figures: Rectangles, Circles, and Triangles

Implement the entire Points in Figures scenario. That is, rectangles, circles, and triangles can be specified in the input file. Here is a sample input file:

```
r 8.5 17.0 25.5 -8.5
c 20.2 7.3 5.8
t -1.0 -1.0 10.1 2.2 .4 1.4
r 0.0 0.0 5.5 10.3
c -5.0 -5.0 3.7
t 20.3 9.8 10.0 -3.2 17.5 -7.7
r 2.5 2.5 12.5 12.5
c 5.0 15.0 7.2
t -10.0 -10.0 10.0 25.0 30.0 -10.0
*
2.0 2.0
4.7 5.3
6.9 11.2
20.0 20.0
17.6 3.2
-5.2 -7.8
9999.9 9999.9
```

The above input file would produce the following output:

```
Point 1 is contained in figure 4
Point 1 is contained in figure 9
Point 2 is contained in figure 4
Point 2 is contained in figure 7
Point 2 is contained in figure 9
Point 3 is contained in figure 7
Point 3 is contained in figure 8
Point 3 is contained in figure 9
Point 4 is not contained in any figure
Point 5 is contained in figure 1
Point 5 is contained in figure 2
Point 5 is contained in figure 6
Point 5 is contained in figure 9
Point 6 is contained in figure 5
Point 6 is contained in figure 9
```

Note that the sample input file for each of the three implementations used the points and figures from the one before it. That is, the test points are the same in all the files, the rectangles are the same in all the files, and the circles are the same in the second and third files. Figure 2 is a diagram of the final input file.

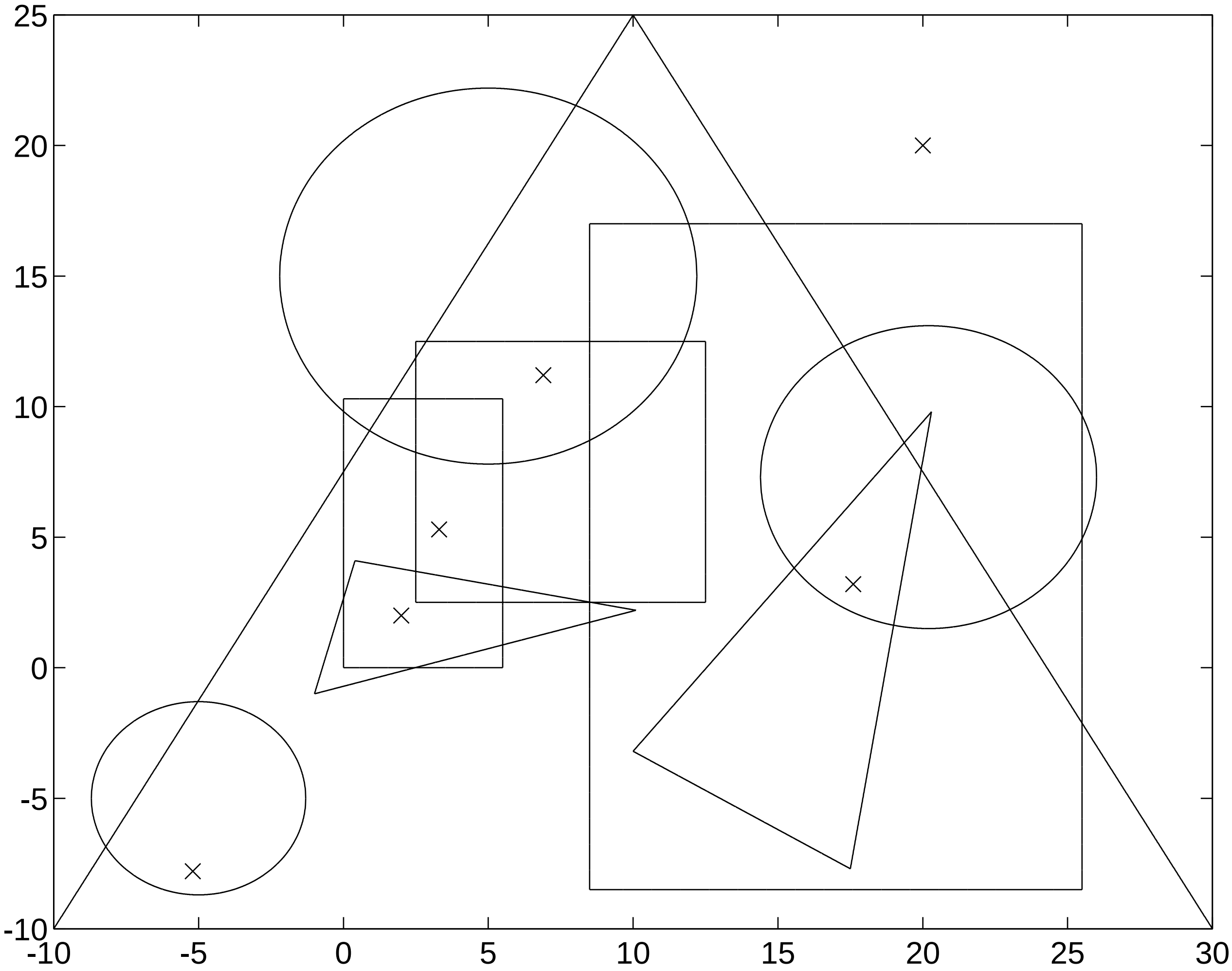

Figure 2: Diagram from the final input file

6 Irrigation Flow Rates

The Mishawaka Irrigation company builds custom designed pipeline irrigation systems for farmers and ranchers. One such irrigation system is shown below (it matches the first dataset in the example input):

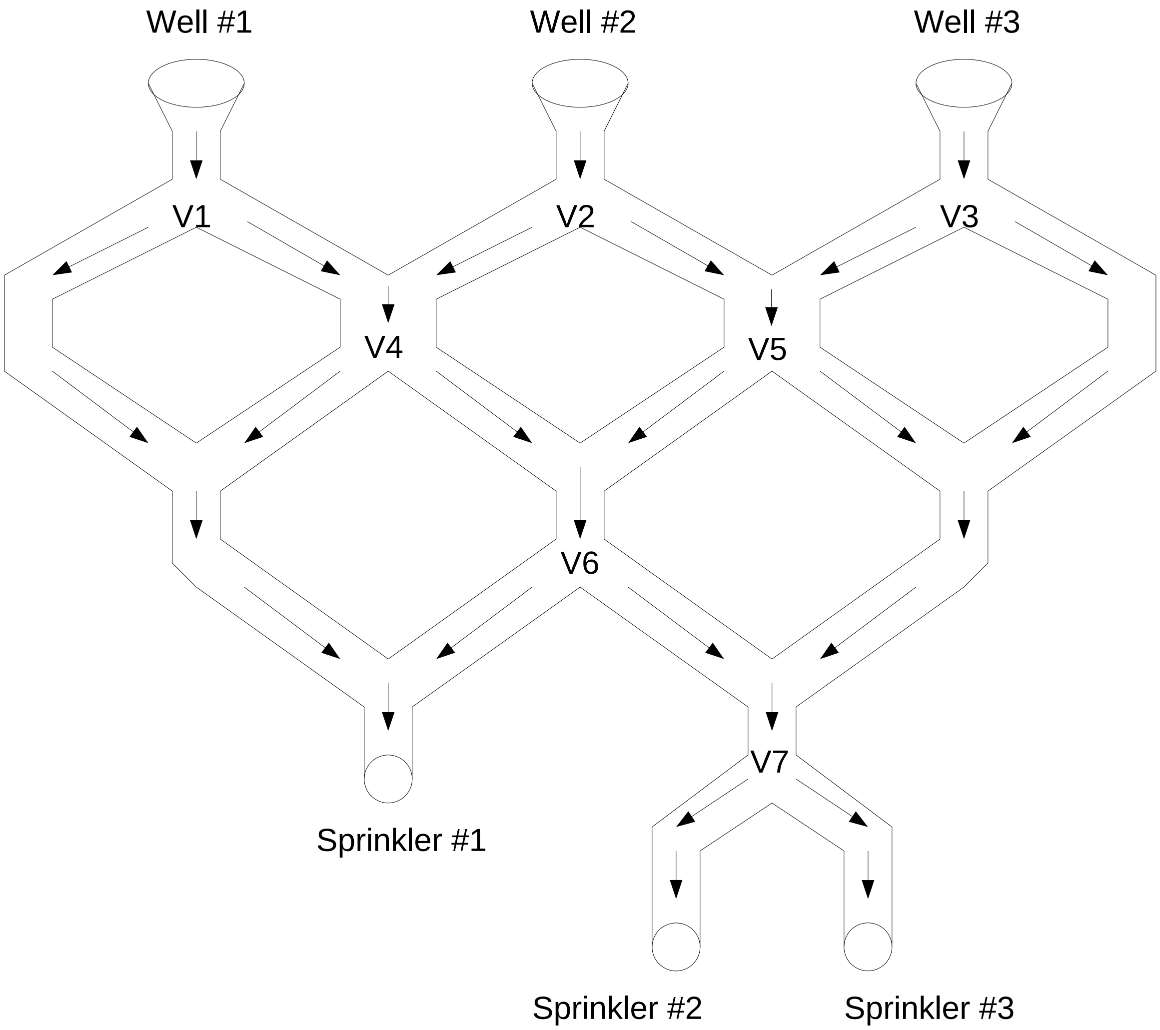

Figure 3: Irrigation network

For the above example, three wells provide water for the irrigation system and a variety of valve settings ($V_1, V_2, \ldots, V_7$) are used to control the amount of water that is delivered to each of the three sprinkler systems. Furthermore, the flow of water occurs only in the direction indicated by the arrows. Each valve can be set to the right (in which case all the water is diverted to the right), or to the left (in which case all the water is diverted to the left).

Given a sprinkler sytstem configuration and each well's corresponding flow rate (in gallons/minute), your program should determine, for different valve settings, the output flow rates that are being delivered to each sprinkler systems. **NOTE:** You should program for generality (i.e., do not make any assumptions concerning the number of water wells, flow valve settings, or sprinkler heads. Also do not make assumptions about the names of the components).

6.1 Input

The input file will consist of multiple data sets; each data set is terminated by an asterisk (*) on a line by itself. Each data set will consist of an irrigation network description (see table 1) followed by one or more flow valve setting records (see table 2). Each flow valve setting record consists of a sequence of L's and R's, one for each valve described in the irrigation network description. The datasets are separated by lines containing only an asterisk (*) in column one. End of data is marked by a terminal record containing the values of 9999 for #ofWells, #ofSprinklers, and #ofValves.

Record #		Data Items	
1	#ofWells	#ofSprinklers	#ofValves
2	Well$_1$FlowRate	Well$_2$FlowRate	...
3	WellName$_1$	DestinationName$_1$	
...	...	...	
#ofWells+2	WellName$_i$	DestinationName$_i$	
#ofWells+3	SprinklerName$_1$		
...	...		
#ofWells+j+3	SprinklerName$_j$		
#ofWells+j+4	ValveName$_1$	LeftDestinationName	RightDestinationName
...	...	...	...
k	ValveName$_m$	LeftDestinationName	RightDestinationName

Table 1: Input (in table form) for Irrigation Network Description

6.2 Output

For each set of valve settings of each irrigation network, your program should output a message of the following form:

```
Sprinkler #i flow is n gallons/min.
```

Label the output for each irrigation network according to the order in which it was inputted (see the example below). Label the outputs for the valve settings in the order in which they were read in for each irrigation network. See the example for the exact format of the output headings and spacing.

6.3 Example

Assuming flow rates of 200, 40, and 74 gallons/minute, the above irrigation system would have the encoding of the first dataset in this example input:

Record #		Data Items	
k+1	$Valve_1Setting$	$Valve_2Setting$	...
...	...	...	...
n	*		

Table 2: Input (in table form) for Valve Settings

```
3        3        7
200      40       73
W1       V1
W2       V2
W3       V3
S1
S2
S3
V1       S1       V4
V2       V4       V5
V3       V5       V7
V4       S1       V6
V5       V6       V7
V6       S1       V7
V7       S2       S3
R L R L R L R
L R L R L R L
*
2        4        5
100      200
WELL1    VALVE1
WELL2    VALVE2
SPR1
SPR2
SPR3
SPR4
VALVE1   VALVE3   VALVE4
VALVE2   VALVE4   VALVE5
VALVE3   SPR1     SPR2
VALVE4   SPR2     SPR3
VALVE5   SPR3     SPR4
R L R L R
L L L R L
L R L R L
*
9999     9999     9999
```

The output for the above input would be:

```
Irrigation network #1
Valve settings #1
Sprinkler #1 flow is 240 gallons/min
Sprinkler #2 flow is 0 gallons/min
Sprinkler #3 flow is 73 gallons/min
Valve settings #2
Sprinkler #1 flow is 200 gallons/min
Sprinkler #2 flow is 113 gallons/min
Sprinkler #3 flow is 0 gallons/min

Irrigation network #2
Valve settings #1
Sprinkler #1 flow is 0 gallons/min
Sprinkler #2 flow is 300 gallons/min
Sprinkler #3 flow is 0 gallons/min
Sprinkler #4 flow is 0 gallons/min
Valve settings #2
Sprinkler #1 flow is 100 gallons/min
Sprinkler #2 flow is 0 gallons/min
Sprinkler #3 flow is 200 gallons/min
Sprinkler #4 flow is 0 gallons/min
Valve settings #3
Sprinkler #1 flow is 100 gallons/min
Sprinkler #2 flow is 0 gallons/min
Sprinkler #3 flow is 200 gallons/min
Sprinkler #4 flow is 0 gallons/min
```

7 Tempus Fugit

The owner of a successful small business is so busy that she requires all of her appointments to be scheduled two weeks in advance. She has contacted you to write a program that will help her automate this process. Her work week lasts from 9:00am to 5:00pm, Monday through Friday. She leaves for lunch at 12:30pm and returns at 1:30pm each day.

During the week, she collects appointment requests for two weeks in advance in the order that they were made. Your program will process the requests in the same order, effectively giving priority to those requests that were made the earliest. After running your program, she will confirm the scheduled appointments and reschedule those appointements that the program could not arrange for her.

She asks that your program schedule her appointments in ten-minute blocks that begin on the hour and at ten, twenty, thirty, forty, and fifty minutes past the hour. Any requests that do not begin and/or end at these ten-minute points in time must be scheduled between the ten-minute points which encompass them. For example, an appointment request for 10:15am to 10:30am would be scheduled from 10:10am to 10:30am.

She also wants your program to insert a ten minute break after each scheduled appointment so that no two appointments occur in immediate succession. The ten-minute break will occupy the next ten-minute block in her schedule. Ten-minute breaks are not to be added for appointments which can be scheduled to end at 12:30pm or 5:00pm. No appointment is permitted to extend into the lunch period, or to be scheduled to begin during the lunch period. She would like no more than four hours of appointments scheduled on any one day. When computing the four-hour limit, use the adjusted length of the appointment (do not use the original request and do not include the ten-minute breaks which follow the adjusted appointment times). Thus, the appointment request for 10:15am to 10:30am contributes 20 minutes to the four-hour time limit for that day.

Finally, the owner tells you that if an appointment cannot be scheduled as requested, your program should attempt to reschedule the appointment at the same time on each successive day of the week, scheduling it on the first day that it fits: if an appointment cannot be scheduled at 10:00am on Wednesday, your program should try to schedule it on Thursday at 10:00am, then at Friday at 10:00am if it cannot be scheduled on Thursday (do not attempt to schedule the appointment earlier in the week or at different times of the day). If your program cannot schedule the appointment in this manner, the name of the person requesting the appointment should be added to the list of appointments that could not be scheduled.

7.1 Input

The input file will consist of at most 25 appointment requests on a separate line in the following format:

```
name day start_time duration
```

where the name field occupies the first ten character positions of the request, the day field occupies the next three character positions, and the remaining two fields each consist of a pair of integers. The name field will contain ten alphanumeric characters and the day field will contain one of the three character sequences: MON, TUE, WED, THU, or FRI. Following the day will be two times: the starting time of an appointment request, and its duration. Both times consist of two integers, hours followed by minutes. For example, the appoint request:

```
Johnstone TUE 09 15 1 30
```

means that Johnstone requests an appointment for the Tuesday of that week starting at 9:15am for one hour and thirty minutes. Since the schedule is in ten-minute blocks, your program would attempt to schedule the appointment for Tuesday from 9:10am to 10:50am (adding 1 hour and 40 minutes to the total appointment time on Tuesday). Your program may assume that all fields in an appointment request will contain legitimate data and that times will be expressed in terms of the maximal number of hours (9:15am will always be 9 15 rather than 8 75 and ninety minutes will always be 1 30 rather than 0 90).

7.2 Output

Your output will consist of a schedule for the week followed by a summary listing the names of those people whose appointments could not be scheduled during this time. Days with no scheduled appointments should be noted.

7.3 Example

Supposed that the input file consisted of the appointment requests:

Johnstone	TUE	09	15	1	30
Peterson	MON	09	00	0	30
McKeever	FRI	09	30	1	00
Garzarelli	THU	10	45	0	20
Tucker	MON	10	00	2	30
Davis	MON	02	30	1	00
Corrigan	MON	02	00	0	15
Trump	WED	01	00	3	00
Logan	THU	09	45	1	05
Schulman	THU	11	10	0	30

Your output would look like this:

```
APPOINTMENT SCHEDULE FOR THE WEEK

MONDAY
Peterson 9:00 to 9:30
Tucker 10:00 to 12:30
Davis 2:30 to 3:30

TUESDAY
Johnson 9:10 to 10:50
Corrigan 2:00 to 2:20

WEDNESDAY
No Appointments Scheduled

THURSDAY
Garzarelli 10:40 to 11:10

FRIDAY
McKeever 9:30 to 10:30
Schulman 11:10 to 11:40

APPOINTMENTS COULD NOT BE SCHEDULED FOR:
Trump
Logan
```