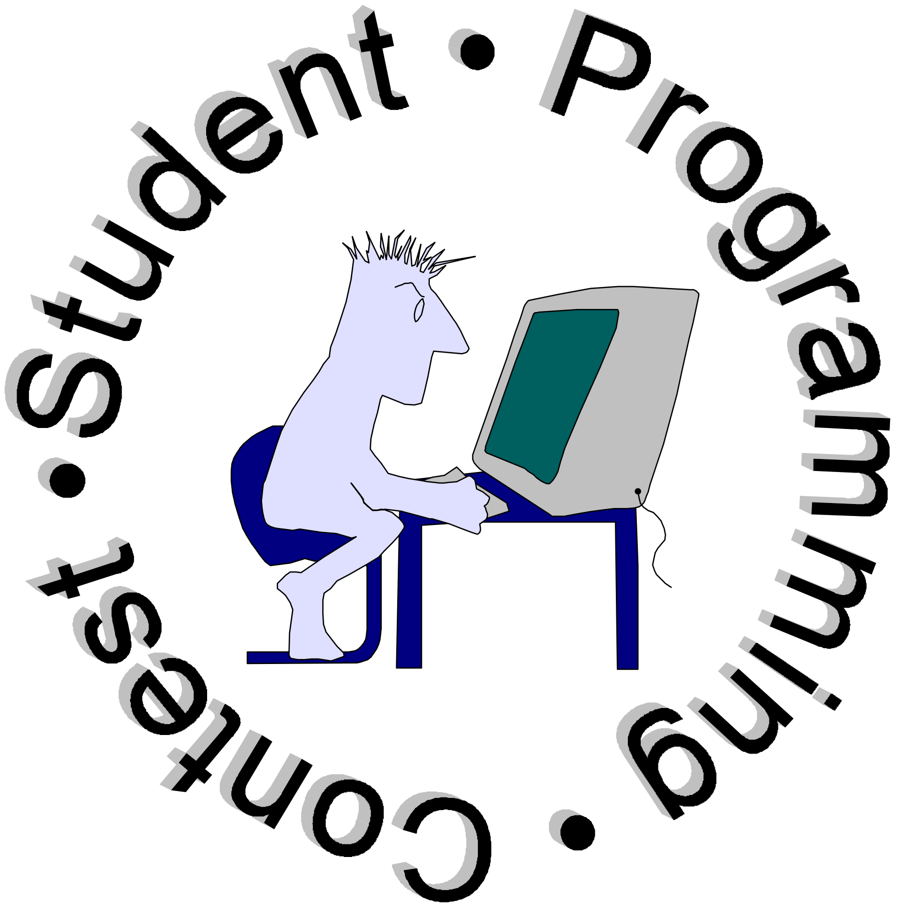
The Fourth Annual

Student Programming Contest

of the

CCSC Southeastern Region



Saturday, November 8, 1997

8:00 A.M. – 12:00 P.M.

Al Gore Rhythm

The Problem

After his brush with the Justice Department over fundraising with Buddhist monks, the Vice President devised a plan to ensure that such activities are carried out in a more discrete manner and are kept less noticeable. Realizing the Democratic National Committee's need for more and more money in the campaign war chest, Mr. Gore devised a "rhythm method" for accepting campaign donations from Buddhist monks and yet avoiding the conception of an independent counsel. Gore's theory is that if the donations are spaced no less than 28 days apart, an independent counsel will not be necessary.

To help Mr. Gore keep track of when it's okay to accept money from Buddhist monks, you must write a program to automatically scan the White House email logs for messages from "`veep@whitehouse.gov`" addressed to "`buddha@whitehouse.gov`" (the code name for the Al Gore Rhythm Method). Each such email is a secret entry in Mr. Gore's Buddhist monk fundraising dairy. Your program must send Mr. Gore ("`veep@whitehouse.gov`") a reply to each such email advising him when the next donation may be accepted. You should assume that the email was sent the day that the donation was accepted. To maintain more secrecy, Mr. Gore refers to a Buddhist monk as a "BM."

Sample Input

Your program must process the White House electronic mail log, stored in the file `whmail.log` as input. The mail log is an ASCII text file which contains a sequence of one or more valid email messages. Each email message consists of a header followed by the body. A header consists of exactly four lines, each of which begins with a unique keyword. The first line is the sender line and begins with the keyword `From` which is followed by a character string representing the email address of the sender. The second line is the recipient line and begins with the keyword `To` which is followed by a character string representing the email address of the recipient of the message. The third line is the date line and begins with the keyword `Date` which is followed by a character string representing the date on which the email message was received. The fourth line is the subject line and begins with the keyword `Subject` which is followed by an arbitrary character string. A single blank space separates the keyword on each header line from the character string that follows. The body of an email message is simply a sequence of one or more lines of text any of which may be blank. The body begins on the fifth line of the email message. There will be normal White House email interspersed with the Al Gore Rhythm email, but your program should ignore all email other than those from "`veep`" to "`buddha`."

Sample contents of `whmail.log` could appear as:

```
From bill@whitehouse.gov
To buffy@airhead.net
Date Saturday, October 4, 1997
Subject Get together
Hey, honeypie. Ole Big Daddy sure is missin you. I'll be a
lookin for you to come around again this weekend.
Love,
Bill

From veep@whitehouse.gov
To buddha@whitehouse.gov
Date Monday, October 6, 1997
Subject BM
Dear Buddha, I just had a BM. Please advise.

From reno@justice.gov
To bill@whitehouse.gov
Date Wednesday, October 8, 1997
Subject Roby Ridge

Mr. President:
   The situation with the lady in Roby is quickly deteriorating.
I advise we use an Apache loaded with napalm to flush the crazy
woman out. If it kills her, it serves her right for that Vaseline
trick. Dead or alive, at least it will be over. If I don’t hear
from you within the next hour, I’ll send for the chopper.

Janet
```

Sample Output

The output of your program must be directed to the screen and must consist of a reply email for each Al Gore Rhythm email found in the log. Each reply must specify the date on which the next donation may be accepted. Your output must be formatted *exactly* as that below, which is the output corresponding to the sample input above.

```
From buddha@whitehouse.gov
To veep@whitehouse.gov
Date Saturday, November 8, 1997
Subject Re: BM

Thank you for advising me of your BM. You may not have
another BM until Monday, November 3, 1997.
```

P R O B L E M   T W O

Remorseful Sam

The Problem

Sam Morose served for many years as communications officer aboard the U.S.S. Dahdit, a U.S. Coast Guard frigate deployed in the South Pacific. Sam never quite got over the 1995 decision to abandon Morse code as the primary ship-to-shore communication scheme, and was forced to retire soon after because of the undue mental anguish that it was causing. After leaving the Coast Guard, Sam landed a job at the local post office, and became a model U.S. Postal worker…

That said, it isn't surprising that Sam is now holding President Clinton hostage in a McDonald's just outside the beltway. The FBI and Secret Service have been trying to bargain with Sam for the release of the president, but they find it very difficult since Sam refuses to communicate in anything other than Morse code. Janet Reno has just called you to write a program that will interpret Sam's demands in Morse code as English.

Morse code represents characters of an alphabet as sequences of dits (short key closures) and dahs (longer key closures). If we let a period (.) represent a dit and a dash (-) represent a dah, then the Morse code version of the English alphabet is:

A	`.-`	G	`--.`	M	`--`	S	`...`	Y	`-.--`
B	`-...`	H	`....`	N	`-.`	T	`-`	Z	`--..`
C	`-.-.`	I	`..`	O	`---`	U	`..-`		
D	`-..`	J	`.---`	P	`.--.`	V	`...-`		
E	`.`	K	`-.-`	Q	`--.-`	W	`.--`		
F	`..-.`	L	`.-..`	R	`.-.`	X	`-..-`		

Sample Input

Your program must takes its input from the ASCII text file `morse.in`. The file contains periods and dashes representing a message composed only of the English alphabet as specified above. One blank space is used to separate letters and three blanks are used to separate words. Sample contents of the file could appear as follows:

```
.... . .-.. .-.. ---   -- -.--   -. .- -- .   .. ...   ... .- --
```

Sample Output

Your program must direct its output to the screen and must be the English interpretation of the Morse code found in the input file. There must be no blanks between letters in the output and only one blank between words. You must use all capital letters in the output. The output corresponding to the input file above is:

```
HELLO MY NAME IS SAM
```

Wood Chuckin

The Problem

How much wood could a woodchuck chuck if a woodchuck could chuck wood? The answer to this age-old question had persistently eluded scientists until earlier this year. Two Hungarian zoologists finally arrived at an equation that has been empirically verified as relating the amount of wood that a woodchuck can chuck to the length of the woodchuck's right front tooth. It seems that a woodchuck can chuck 5 kilograms of wood for each millimeter of right front tooth. So, a woodchuck with a 15 millimeter right front tooth could chuck 75 kilograms of wood.

Write a program that calculates how much wood a woodchuck could chuck given the length of its right front tooth.

Sample Input

Your program must take its input from the ASCII text file `chuck.in`. The file consists of one or more lines with two entries. The first entry on a line is the name of the woodchuck and the next entry is the length of that woodchuck's right front tooth as measured in millimeters and specified as a positive integer. Sample contents of the file could appear as:

```
Harry 25
Fuzzy 34
Bushy 19
```

Sample Output

Your program must direct its output to the screen and list the amount of wood able to be chucked for each woodchuck listed in the input file, **in ascending order of chucking amount**. The output for the sample input above would be:

```
Bushy the woodchuck can chuck 95 kilograms of wood.
Harry the woodchuck can chuck 125 kilograms of wood.
Fuzzy the woodchuck can chuck 170 kilograms of wood.
```

Bull's Eye

The Problem

You have been tasked to develop a scoring program for an electronic version of darts. The user is presented with a target composed of ten concentric circles, with the inner-most circle carrying the most points if hit. The points associated with a circle decreases as its radius increases. No points are given for a hit outside the target, that is outside the outer-most circle.

The center of the target lies at the center of a window on the screen. The radius of the inner-most circle is 50 pixels, and the radius of each of the remaining circles is 50 pixels greater than the circle just within it. Thus, the outer-most circle has a radius of 500 pixels. The player of the darts game shoots a Nintendo-like gaming gun at the screen and the game's software records the (x,y) coordinate of the "hit." The game window is viewed by the software as a Cartesian plane with the origin (0,0) at the center of the window. The game player gets five shots and then the software tallies the score. Points are distributed as follows: circle 1 (inner-most circle) = 500 points; circle 2 = 300 points; circle 3 = 250 points; circle 4 = 200 points; circle 5 = 150 points; circle 6 = 100 points; circle 7 = 75 points; circle 8 = 50 points; circle 9 = 25 points; circle 10 (outer-most circle) = 10 points; outside all circles = 0 points;

Write a program that performs the scoring function of the electronic darts game.

Sample Input

Your program must take its input from the ASCII text file `darts.in`. The file consists of one or more game summaries. Each game summary records the game player's name and the (x,y) coordinates of each of their five shots. Thus, each game summary consists of six lines: the first contains the player's name and the remaining five lines record the shot coordinates, one per line, with the x (horizontal) coordinate appearing first and the y (vertical) coordinate appearing second. Exactly one blank space separates the x and y coordinates. Sample contents of the file could appear as:

```
Frank
75 200
0 1
650 780
300 490
15 344
Sally
0 0
95 63
100 250
378 -97
1 1
```

Sample Output

Your program must direct its output to the screen and must report a score summary for each player recorded in the input file. The score summary should indicate the player's name, the points associated with each hit, and the total score for the game. Your program's output must for formatted exactly as shown below, which is the output corresponding to the sample input given above.

```
Score Summary for Frank
-----------------------
   Hit 1 =          150
   Hit 2 =          500
   Hit 3 =            0
   Hit 4 =            0
   Hit 5 =           75
              ---------
   Score =          725


Score Summary for Sally
-----------------------
   Hit 1 =          500
   Hit 2 =          250
   Hit 3 =          100
   Hit 4 =           50
   Hit 5 =          500
              ---------
   Score =         1400
```

Monotonic Words

The Problem

I'm sure you're familiar with monotonous words from your CS lectures, but you may not have heard of monotonic words. The concept is similar to that of a monotonic function. A function f is *increasing* on some interval of numbers I if for every x, y in I, x < y implies that f(x) < f(y). Likewise, a function is *decreasing* on some interval of numbers I if for every x, y in I, x < y implies that f(x) > f(y). A function is *monotonic* if it is either increasing on I or decreasing on I. A word, then could be said to be increasing if for every pair of characters x, y in the word such that x appears before y in the word, x appears before y in the alphabet. A word is decreasing if for every pair of characters x, y in the word such that x appears before y in the word, x appears after y in the alphabet. A monotonic word is one that is either increasing or decreasing.

Write a program that identifies monotonic words.

Sample Input

Your program must take its input from the ASCII text file `mono.in`. The file contains a sequence of one or more words, one per line. Sample contents of the file could appear as:

```
the
ardvark
billow
begin
```

Sample Output

Your program must direct its output to the screen and identify all monotonic words in the input file. The output which corresponds to the sample input above is:

```
the
begin
```

Hailstones

The Problem

In the January 1984 issue of *Scientific American*, an interesting sequence of numbers known as a hailstone series was described. The series is formed by generating the next number based on the number just prior to it. If the previous number was even, the next number in the series is half of it. If the previous number was odd, the next number is three times it plus one. Although the series goes up and down, it eventually settles into a steady state of 4, 2, 1, 4, 2, 1, … For example, starting at 21, the hailstone series is: 21, 64, 32, 16, 8, 4, 2, 1, 4, 2, 1, … For 21, it required five steps before the steady state was reached.

Write a program that computes how many steps will be necessary before the steady state is reached in the hailstone series beginning at a given number.

Sample Input

Your program must take its input from the ASCII text file `hail.in`. The file contains a sequence of one or more integer values, one per line. Sample contents of the file could appear as:

```
21
10
```

Sample Output

Your program must direct its output to the screen and must tell the number of steps required to reach the hailstone steady state for each integer recorded in the input file. Your program must format its output exactly as that shown below which is the output corresponding to the sample input above.

```
5 steps were necessary for 21.
4 steps were necessary for 10.
```