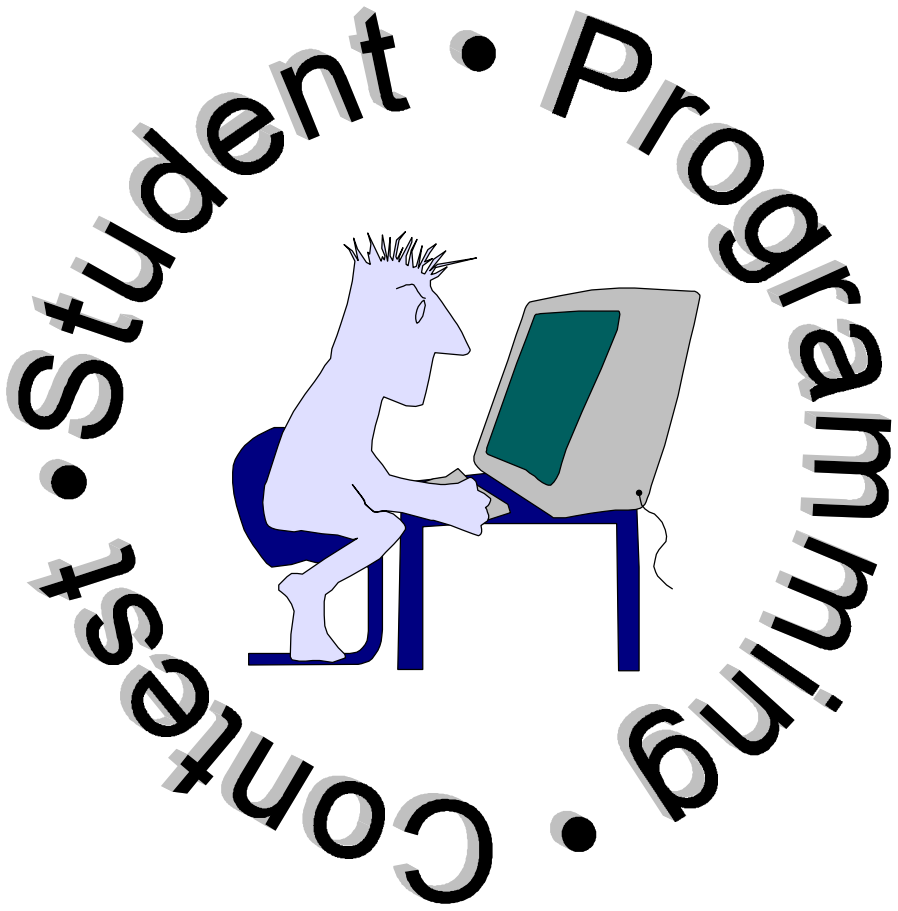
The Third Annual

Student Programming Contest

of the

CCSC Southeastern Region



Saturday, November 9, 1996

8:00 A.M. – 12:00 P.M.

The Trouble With Tribbles

The Problem

Kirk: "Get us out of here, Mr. Scott!"
Scotty: "She canna take much more of this, Cap'n!"
Spock: "Interesting."
Kirk: "Report, Bones?"
McCoy: "*#%@ it, Jim, I'm just a doctor!"
Scotty: "She's gonna blow, Cap'n!"
Spock: "Fascinating."
Kirk: *(Grasping the helm with his trademark wild-eyed look)* "Then let it blow. At least we'll take those tribbles with us! At least we'll die free: Ah, freedom, that noble state in which all men yearn to live; that blessed…"
McCoy: *(Grabbing Kirk's arm with his trademark ornery old man look)* "Shut up, Jim!"
(Turning to Spock) "Why don't you do something, Mr. Smarty-Pants-Pointy-Eared-Halfbreed??"
Spock: "It would be illogical to do nothing. In fact, doctor, while you and the others have been shamelessly displaying your human emotions, I have been calculating the amount of time we have remaining until the warp core collapses upon itself. My calculations show that we can stop this catastrophe from occurring if we direct a high-power quantum phase hole in time neutrino subspace blast toward the warp engines in when the tribble count reaches exactly …

Suddenly Spock's eyes roll back in his head and his body goes rigid.

McCoy: "*#%@ it, Spock! Didn't I tell you to keep a little something on your tummy? Low blood sugar isn't something to screw around with! Now look what's happened!"

With Spock unable to speak and move, the crew looks to you to finish Spock's calculations and save the ship. Here's the situation: A nasty colony of tribbles has set up a breeding ground in the warp core of the Enterprise. The trouble with tribbles is their unbelievably fast rate of reproduction. If the tribbles are not stopped soon, the sheer number of them in the engine will cause the ship to explode. Currently there are 2187 tribbles in the engine. Each tribble autonomously reproduces at the rate of 3 per hour; that is, each single tribble splits into three new tribbles once each hour. Now, for reasons not understood (even by Spock), tribbles are vulnerable to high-power quantum phase hole in time neutrino subspace blasts during their 15^{th} hour of reproduction. The crew doesn't know when the tribbles began reproducing, but they do know that it all started with only one tribble. Write a program that tells Kirk when to give the engine a high-power quantum phase hole in time neutrino subspace blast by giving the count of tribbles that would be present after 15 hours of reproduction and the number of hours remaining until the blast must be given.

Sample Input

There is no external input for this problem.

Sample Output

Your program should direct its output to the screen and should look like: (except for where lines break and the spacing around the numbers X and Y)

```
Anonymous Crew Member: "Captain, I have it! Direct the blast at the engine when the
tribble count reaches exactly X. We have approximately Y hours before you must do
this."
```

Idiot Savant or Just an Idiot?

The Problem

An idiot savant is a wonderfully gifted person who can perform highly specialized tasks which are often quite difficult to the rest of us, while not being able to perform routine tasks which the rest of us take for granted. For example, one such person can tell you exactly the day of the week a given date will fall on (for ANY date you give him!) but he finds it just beyond his ability to tie his shoes.

If you think about it for a moment, the software we write shares some commonality with idiot savants. Researchers in artificial intelligence have succeeded in writing software which is capable of autonomously driving a vehicle at highway speeds along the interstate, but it is quite incapable of understanding the stories which my 7 year old daughter reads to me. Of course, software is probably more of an idiot than a savant, but you get the idea.

Write a program that imitates the capability of the idiot savant I mentioned above. Your program should read in a date in MM/DD/YYYY format and respond with what day of the week that date falls on, if the date is valid. Here are some helpful hints:

- Thirty days hath September, April, June, and November. All the rest have thirty-one, except February which has 29 days in a leap year and 28 days in a non-leap year.
- A year is a leap year if either (a) it is evenly divisible by 4 and not by 100, or (b) it is evenly divisible by 400.
- You can calculate the day of the week on which January in a given year starts by the following magic formula:

  Let y = year – 1. Then Jan.1 falls on day X where
  day = (36 + y + (y div 4) - (y div 100) + (y div 400)) mod 7
  where Sunday = 0, Monday = 1, etc., Saturday = 6.

Sample Input

Your program should take its input from the text file `dates.dat`. This file consists of an undetermined number of lines, each of which contains a single date in MM/DD/YYYY format (month/day/year). Each line is exactly 10 characters long, is correctly delimited, and is left-justified. Here are a few sample lines from the file. The first line is for August 17, 1989.

```
08/17/1989
06/14/1995
43/12/1996
```

Sample Output

Your program should direct its output to the screen. Appropriate output for the sample inputs given above is:

```
8/17/1989 falls on a Thursday
6/14/1995 falls on a Wednesday
Invalid date in the input.
```

Word Counter

The Problem

Ever had an English composition professor who actually counted the words in your term papers to make sure that you did not exceed the limit? After you get an "F" on a 5001 word paper because it was supposed to be a 5000 word paper, a few things come to your mind: What sort of pathetic life does this person lead that causes them to COUNT 5001 WORDS?? How can I keep this from happening again?

Write a program that processes a text file and calculates the number of words it contains. A word is simply a sequence of non-blank, non-punctuation characters.

Sample Input

Your program should take its input from the text file `words.dat`. This file contains arbitrary text delimited by the punctuation marks:

. , ; : ! ?

In addition to punctuation marks, text is also delimited by blank spaces.

Sample Output

Your program should direct its output to the screen and appear as follows:

```
This file contains X words.
```

Where X is an integer representing the number of words in the file.

Buffy the Cipher Slayer

The Problem

A distinguished professor at a major research university in the northeast had spent a career perfecting a novel approach to sending secret messages. Convinced that his method was secure and that it would land him an endowed chair at a university in a better climate, he boldly published his work in a scholarly journal. Sadly, the professor's life work was crushed by freshman Phi Mu pledge Buffy McBuff, who happened to be scanning the journal after a long day at Macy's. When interviewed about breaking the cipher, Buffy described the professor's method as "like, a reverse diagonal thing, like" and stated that it is easily solved.

Here's how the cipher works: To send a secret message an NxM ($0 < N, M < 8$) grid of randomly chosen words is constructed. The words of the message replace words in certain locations in the grid. The locations of the meaningful words are chosen in a way that the recipient of the message can extract them. Each word in the grid is exactly ten characters long (possibly padded to the right with blanks). The first meaningful word is word M on row 1; the second meaningful word is word M-1 on row 2; the third meaningful word is word M-2 on row 3; and so on. If N > M, the cipher "wraps"; that is, the M+1 meaningful word is word M on row M+1. When the last row is reached, the message is complete.

Sample Input

Your program should take its input from the text file `buffy.dat`. This file consists of an undetermined number of reverse diagonal cipher grids, each preceded by a single line with two integers separated by a blank space. The first integer represents the number of rows in the next grid and the second integer represents the number of columns. Sample input would be:

```
3 5
flower    baby      hulahoop  nasty     attack
tractor   piano     fetch     at        suede
Elvis     bolt      dawn      marsupial ball
4 2
blue      how
now       axe
plow      brown
cow       party
```

Sample Output

Your program should direct its output to the screen. The correct output for the sample input given above is:

```
Message 1 => attack at dawn
Message 2 => how now brown cow
```

Parenthetical Expressions

The Problem

Three standard ways of representing arithmetic expressions are in prefix, infix, and postfix notation. In prefix, a binary operator immediately precedes its two operands; in infix a binary operator is placed between its two operands; in postfix a binary operator immediately follows its two operands. For example, here are three equivalent forms of the same arithmetic expression:

Prefix: + 3 4
Infix: 3 + 4
Postfix: 3 4 +

Although we are accustomed to using infix notation, it isn't as compact as the other two because of the need for parentheses. For example, to add 5 to 4 then multiply the result by 8 we would have to use parentheses in infix notation to force the addition to be done before the multiplication, while prefix and postfix would not need them:

Prefix: * + 5 4 8
Infix: (5 + 4) * 8
Postfix: 5 4 + 8 *

Write a program which accepts prefix arithmetic expressions involving single-digit operands and the operators + (addition), - (subtraction), * (multiplication), and / (division) and outputs an equivalent infix expression, fully parenthesized.

Sample Input

Your program should take its input from the text file `prefix.dat`. This file consists of an undetermined number of prefix expressions, one per line. Sample contents of the file would appear as:

```
+ 3 4
* + 5 4 8
```

Sample Output

Your program should direct its output to the screen. The correct output for the sample input given about is:

```
(3 + 4)
((5 + 4) * 8)
```

igPay atinLay anslatorTray

The Problem

Ifyay ouyay ancay eadray isthay, enthay ouyay owknay atwhay ouyay ustmay oday. If you can't read the previous sentence, then let me explain the problem a bit more. Pig Latin is an elegant but archaic language that was in common usage in the days of Charlemane and King John. It is a little known fact that the Magna Carta was first written in Pig Latin and only later translated into lesser tongues.

There is a simple correspondence between Pig Latin and English. First assume that an English word is of the form XY where X is the beginning letter or group of letters and Y is the remainder of the word. Each corresponding word in Pig Latin is of the form YXay, with only a few exceptions. For example, the translation of the Pig Latin "ownay" is "now" in English. In general, you can translate the Pig Latin form "YXay" into the English form "XY". So, the first sentence on this page is translated in English as "If you can read this, then you know what you must do." Punctuation is unaffected.

Write a program that reads in Pig Latin and translates it into English. I'll give you the rules for translating English into Pig Latin, so all you will have to do is "invert" them to write your program. The rules for translating English into Pig Latin are based on how the English word begins:

X	Example
th, kn, wh, tr, sh, fl	"that" → "atthay"
	"know" → "owknay"
a single consonant	"can" → "ancay"
	"dog" → "ogday"
a vowel	"if" → "ifyay"
	"I" → "Iyay"

Notice that in the case of a vowel beginning a word, the vowel is not moved to the end. Rather, an extra "y" is added onto the "ay" in its place. Also, the list of multi-letter beginnings shown on the first row could be expanded significantly. For the purposes of this problem, however, those listed are all you have to deal with.

Sample Input

Your program should take its input from the text file `pig.dat`. This file contains an undetermined number of words in Pig Latin separated from one another by blank spaces, commas, and periods. A sample from this file might look like:

```
enWhay igspay yflay!
```

Sample Output

Your program should direct its output to the screen. The correct output for the input shown above is:

```
When pigs fly!
```