# Questions

The Philadelphia Classic

Fourth Annual High School Programming Contest

the

dining philosophers

University of Pennsylvania

March 3, 2001

# Contents

# Part I
# Questions

## 1 365-365

When the number 64,253 is multiplied by 365 the product is 23,452,345. Notice that the first four digits are the same as the last four digits (2345 and 2345).

Write a program that will find all integers that can be multiplied by 365 to produce an eight-digit number, such that the first four digits are the same as the last four digits.

### Input:

This program accepts no input.

### Output:

This program should output a string formatted like the following, showing the number of matches.

```
1234 matching numbers found
```

Note: 1234 is *not the answer.*

(Hints: Use mod, and think about the range of numbers you need to test.)

## 2 Pig Latin

Write a program to convert English text to Pig Latin.

A word is converted to Pig Latin in the following way:

1. If the word starts with a vowel, just place the letters "hay" at the end of the word.
2. If the word starts with a consonant, move the first letter of the word to the end, and follow it by placing the letters "ay".

### Input:

Input will be provided on the command line (i.e. each word of the text will be its own argument)

Words only contain upper and lower case letters (`a-z` and `A-Z`).

### Output:

Output the Pig-Latin-ized text.

### Examples:

**Input:** `this is a test`

**Output:** `histay ishay ahay esttay`

**Input:** `one million dollars`

**Output:** `onehay illionmay ollarsday`

## 3 Anagrams

Write a program to compare two strings and determine if they are anagrams of each other.

Two strings are anagrams of one another if they contain exactly the same characters in (perhaps) different order. For example, the following two strings are anagrams of each other: "Mildred Smedley" and "slid remedy meld".

Ignore case and punctuation. `"M"`="`m`" and `"Eh?!?!?"`=`"Eh"`. Only numbers and letters are relevant.

### Input:

Input will be passed on the command line in the first two arguments.

Strings need to be placed inside quotes when passed to the program (and therefore cannot contain quotes.)

### Output:

The program should display the two strings it recieved and then print either:

```
The two strings are anagrams of each other.
```

or

```
The two strings are NOT anagrams of each other.
```

### Examples:

**Input:** `"She sells seas shells on the sea shore."`
`"Shells... SEAS... ShORE. sHE sells ONE th? Sea?!?!!"`

**Output:**

```
She sells seas shells on the sea shore.
Shells... SEAS... ShORE. sHE sells ONE th? Sea?!?!!
The two strings are anagrams of each other.
```

**Input:** `"toledo ohio" "ohio river"`

**Output:**

```
toledo ohio
ohio river
The two strings are NOT anagrams of each other.
```

## 4 Longest Substring Sequence

Find the length of the longest sequence of a repeated character in the provided input.

Ignore case and all non alphabetical characters. You may assume that the string has no spaces.

### Input:

The program accepts one argument, which is a sequence of characters (`a-z`), containing no spaces. Case should be ignored.

### Output:

Your program should output the string it recieved as input, as well as the length of the longest sequence and the character repeated in it. If there is a tie for longest sequence, then output any one correct solution.

See the examples below for formatting.

### Examples:

**Input:** `aaaabbbc`

**Output:**

```
String: aaaabbbc
Longest sequence is 4 of A.
```

**Input:** `aabbccaaabbc`

**Output:**

```
String: aabbccaaabbc
Longest sequence is 3 of A.
```

## 5 Nested Squares

Write a program to generate a set of nested squares in the following manner:

Draw a square with sides of the given length drawn with A's. Within this square, another square will be drawn with B's. Within the B square, another square will be drawn with C's, etc. The procedure will stop once the original square is entirely filled.

Do not allow squares to grow beyond the A-Z range. Do not accept the negative lengths. The solution should handle these cases by printing an error.

### Input:

Your program should accept one integer describing the length of a side of the outer square.

### Output:

Your program should output the square, as shown in the examples below.

### Examples:

**Input: 5**

**Output:**

```
AAAAA
ABBBA
ABCBA
ABBBA
AAAAA
```

**Input: 8**

**Output:**

```
AAAAAAAA
ABBBBBBA
ABCCCCBA
ABCDDCBA
ABCDDCBA
ABCCCCBA
ABBBBBBA
AAAAAAAA
```

## 6 Pascal's Triangle

Write a program that generates N (N $\leq$ 13) rows of Pascal's Triangle. The following is a description of Pascal's Triangle:

The top of Pascal's Triangle, row 0, contains the number 1. The following row, row 1, contains two 1's. Each element in a row is generated by summing the two elements in the row above it which are directly to the right and the left of the element you are working on. All elements outside the triangle are zeros, and are not printed. Please look at the example output below to see what the output looks like.

Your program will take as the input the number of rows, and should print an equilateral Pascal's Triangle. *You may not hardcode the triangle. You must generate it!*

Hint: The rows can be generated by computing rCe (r CHOOSE e), where r is the row number and e is the element number of the row (Row element numbers start with 0)

### Input:

The input to your program will be an integer between 1 and 13 inclusive, that represents the number of rows to generate.

### Output:

The output should be a well balanced equilateral triangle as described above and shown below. It must be nicely spaced out as in the sample output.

### Examples:

**Input:** `6`

**Output:**

```
                 1
               1   1
             1   2   1
           1   3   3   1
         1   4   6   4   1
       1   5  10  10   5   1
```

**Input:** `13`

**Output:**

```
                        1
                      1   1
                    1   2   1
                  1   3   3   1
                1   4   6   4   1
              1   5  10  10   5   1
            1   6  15  20  15   6   1
          1   7  21  35  35  21   7   1
        1   8  28  56  70  56  28   8   1
      1   9  36  84 126 126  84  36   9   1
    1  10  45 120 210 252 210 120  45  10   1
  1  11  55 165 330 462 462 330 165  55  11   1
1  12  66 220 495 792 924 792 495 220  66  12   1
```

## 7 Vignere Cipher

The Vignere cipher was developed by Blaise de Vignere in the 1500s. It is a generalization of the Caesar cipher and is considered a stream cipher.

In a Vignere cipher, each letter of the alphabet corresponds to a number using the scheme: A=0, B=1, C=2, ... Y=24, Z=25. To encode a message, an encryption key word is "added" to the message. For example, if the message were IDES OF MARCH, and the key DAGGER used, the encryption would be:

Message: IDESOFMARCH Key: DAGGERDAGGE Encryption: LDKYSWPAXIL

Note that spaces are not encrypted and sums greater than 25 wrap back around to the beginning of the alphabet again. For example, L + W yields H. The ASCII code for the letter A is 65.

Create a program that would encrypt and decrypt messages based upon the following scheme: When encrypting, the Vignere method is used. The key is the first word of the message that is at least five letters or longer. For example, in the message "IDES OF MARCH", "MARCH" would be chosen as the key. If the message contains only four letter or shorter words, such as "WILL THIS BE ON THE TEST", then the first five letters of the message ("WILLT") become the key. You may assume that punctuation will not be entered, uppercase letters will be used, and that messages shorter than five letters will not be encoded.

### Input:

The input will be passed on the command line. The first argument will be the command. `e` for Encrypt and `d` for Decrypt. The message and key (if needed) will be in the remainder of the arguments. The message and key will be all upper case characters, consisting only of the characters `A-Z` and spaces.

To encrypt:
```
e message
```

To decrypt:
```
d key message
```

### Output:

The output should be the encrypted or decrypted message.

### Examples:

**Input:** `e THIS IS A TEST MESSAGE`

**Output:** `FLAKIYEFIKLMKWEEYW`

**Input:** `d MESSAGE FLAKIYEFIKLMKWEEYW`

**Output:** `FLAKIYEFIKLMKWEEYW`

**Input:** `e HELLO WORLD`

**Output:** `OIWWCDSCWR`

**Input:** `d HELLO OIWWCDSCWR`

**Output:** `HELLOWORLD`

## 8 Tetris

You are going to play a simulated game resembling tetris against your fellow competitors, where you will output the result of a list of tetris moves.

Your 'tetris' board is 10 squares across (0 to 9 left to right) and has a maximum height of 400 rows.

The pieces in this game are:

```
(S)quare:               (U)pward Tee:
        xx                      x
        Ax                     Axx

(V)ertical Bar:         (L)eft Tee:
        x                       x
        x                      xx
        x                       A
        A
                        (R)ight Tee:
(H)orozontal Bar:              x
        Axxx                   xx
                               A
(D)ownward- tee:
        xxx
         A
```

Each piece has an anchor point which is described above as a capital 'A' (as opposed to the lower case 'x'). (Do not output the A in your answer.)

You will be given a list of pieces and positions, which will arrive at your board in the order they are listed in the input. Each piece will be specified by its one letter name 'R' for right tree, 'H' for horizontal bar, etc. (see the sample input file below), and the position will be a number from 0 to 9.

Simulate a tetris-like "drop" of the piece at the given position. The piece will come to rest either at the bottom of the board, or one row above the first piece that it hits on the way down. This is exactly like tetris, and an example is provided at the end of the question.

Just as in tetris, if any row completely fills with blocks, that row is removed as soon as it is filled, and all rows above it are shifted downward.

### Input:

Your input will be on the command line, with the first argument being the piece name (a single capital letter, representing a piece), the second argument being that piece's position, and third being the next piece's representative letter... etc. Thus your input may be: `V 5 L 8 R 4` for a vertical bar in position 5, a Left in position 8, etc...

## Output:

Your output shoud be an ASCII representation of the board in its final state, only including only the rows of the board which have at least one part of a piece on them (we do not want a 400 character long board printed only if the bottom 5 rows are filled with pieces).

## Example:

A brief example of dropping and printing.

Suppose you are given the input sequence: `V 6 H 5`

After processing `V 6` the board looks like:

```
4      x
3      x
2      x
1      x
 ----------
 0123456789
```

and after processing `H 5` the board will look like:

```
5     xxxx
4      x
3      x
2      x
1      x
 ----------
 0123456789
```

And your output will look like the above board.

**Input:** `V 0 V 1 V 2 V 3 V 4 V 5 V 6 V 7 V 8 V 9 S 4 S 4`

**Output:**

```
  3     xx
  2     xx
  1     xx
  0     xx
    ----------
    0123456789
```

Note the vertical bars filled some rows and were removed.

**Input:** `S 1 V 3 H 5 D 6 U 2 L 4 R 4`

**Output:**

```
10     x
 9     xx
 8     x
 7     x
 6    xx
 5    xx
 4   xxx
 3    x
 2    x xxx
 1  xxx  x
 0  xxx xxxx
   ----------
   0123456789
```

*(Hints: You may want to approach this by using array management, and break up your code into functions that handle dropping pieces, printing,removing filled rows, etc.*

*When printing the final board, make sure you print all the way up to the row that has the upper part of the top most piece, not the row that contains the anchor of the top most piece.*

*Every input piece/position we give you will be valid, you do not need to check for validity.*

*You may want to write the function that removes the filled rows last, but be sure to apply it after each piece is dropped, and remember that more than one row can be removed every turn.)*

## Part II
# Addendum

## 1 Rules

- The Philadelphia Classic is a four hour, eight question competition.
- Each team consists of up to four participants and a coach/chaperone.
- Teams may bring any written materials, such as: books, old homework assignments, notes, etc., that they feel will assist them during the competition.
- No PDA's, notebooks, cell phones, pagers, fancy calculators, or other electronic devices will be allowed. Simple four-function calculators will be allowed at the judges discretion.
- Communication with *any source other than your student team member, i.e. coaches, other persons not on the team, web pages, newsgroups, etc. is strictly forbidden.*
- *The questions will be specified in detail, often with a specific output format and where necessary specific input format.*
- *Each problem will be graded on a 10-7-5 scale. This means if a team is successful on the first submission, they will receive 10 points, 7 points for the second submission, and 5 for any subsequent submissions. A team may submit as many times as is necessary to correctly solve the problem.*
- *No partial credit will be given for submissions. In the event of an incorrect submission, teams will be given information on the nature of the error.*
- *In the event of a tie, the following rules shall determine rankings, in order of importance: Least number of submissions for all problems. Oldest correct submission (i.e. who submitted first). Best score on individual questions in reverse order. Least number of submissions on individual questions in reverse order. Oldest correct submission on individual questions in reverse order.*
- *All submissions will be reviewed individually by a human judge to make sure that differing output formats will be graded correctly, and all teams will be allowed to petition the judge immediately if they feel that an error has occurred in the grading of their submission. Apart from this petition, all decisions by the judge are final.*
- *Furthermore, proctors will be available in the competition area to answer any questions about usage of specific software or for limited clarification of the contest questions.*

## 2 Examples of command line access

Command line arguments are space seperated character strings. In our contest, you enter these strings in a special window in the client.

These arguments are given to your program in specially defined character arrays. In C/C++ the number of arguments is also passed into your program in a specially declared integer. In java, this can be obtained by getting the length of the argument array, as shown below.

In the code skeletons (provided in the client), you can access the command line arguments in the following ways.:

### C/C++

Command line arguments are in the `argv[]` array.

The number of arguments passed into your program (plus one) is found in the integer argc. The number is one higher for reasons you don't need to worry about.

The first argument is a character string in `argv[1]`. The second is `argv[2]`, and so forth.

### Java

The number of arguments passed into your program is found in the property `args.length`.

The first argument is a character string `args[0]`. The second is `args[1]`, and so forth.

Below are some examples of using command line arguments.

```
/*
  Example of taking a variable number of arguments on the command line
  of a C program and printing them back to standard output.  This is
  done by looping over argv until we reach argc, printing arguments as
  we go.
*/
#include <stdio.h>

int main(int argc, char **argv)
{
  int i;

  /* argv[0] is the name of the program, argv[1] contains the first argument */
  for (i = 1; i < argc; i++)
    printf("%s ",argv[i]);
  printf("\n");


  /* Print the first argument as a number
     It will be unhappy if the first argument isn't convertable. */
  printf("The first argument as a number is: %d\n",
 atoi( argv[1] );

  return 0;
}
```

```java
/*
  Example of taking a variable number of arguments on the command line
  of a Java program and printing them back to standard output.  This
  is done by looping over the args String array, printing arguments as
  we go.
*/

public class ExampleVariable {

    public static void main(String[] args) {
        for (int i = 0; i < args.length; i++)
            System.out.print(args[i] + " ");
        System.out.println("");

// Print the first argument as a number
// It will be unhappy if the first argument isn't convertable.
System.out.println( new Integer( args[0] ).intValue() );
    }
}
```