UNIVERSITY OF NOTRE DAME
Department of Computer Science and Engineering

Problem Sets **September 18, 1994**
Programming Contest

There are 5 problems, given in no particular order of difficulty. Please notify a judge immediately if you do not have all 5 problems. There are 9 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 What Goes Up

Write a program that will select the longest strictly increasing subsequence from a sequence of integers.

1.1 Input

The input file will contain a sequence of integers (positive, negative, and/or zero). Each line of the input file will contain one integer.

1.2 Output

The output for this program will be a line indicating the length of the longest subsequence, a newline, a dash character ('-'), a newline, and then the subsequence itself printed with one integer per line. If the input contains more than one longest subsequence, the output file should print the one that occurs last in the input file.

1.3 Example

Suppose the input file consisted of the following sequence of numbers:

```
-7
10
9
2
3
8
8
6
```

The output would be

```
3
-
2
3
8
```

Notice that the second 8 was not included — the subsequence must be *strictly* increasing.

2 Permutation Arrays

In many computer problems, it is necessary to permute data arrays. That is, the data in an array must be re-arranged in some specified order. One way to permute arbitrary data arrays is to specify the permutations with an index array. Let x be an array that is to be permuted and let x' be the permuted array. Then, we have the relationship between x and x' that $x'_i = x_{p_i}$.

2.1 Input

The input file will contain two lines of numbers. The first line will be an index array containing the integers $1 \ldots n$, where n is the number of integers in the list. The numbers in the first line will have been permuted in some fashion. The second line will contain a list numbers in floating point format.

2.2 Output

The output for this program will be the list of floating point numbers from the input file, ordered according to the permutation array from the input file. The output numbers must be printed one per line in the same format in which they each appeared in the input file.

2.3 Example

Suppose the input file consisted of the following sequence of numbers:

```
3 1 2
32.0 54.7 -2
```

The output would be

```
-2
32.0
54.7
```

3 Word Scramble

Write a program that will reverse the letters in each of a sequence of words while preserving the order of the words themselves.

3.1 Input

The input file will consist of several lines of several words. Words are contiguous stretches of printable characters delimited by white space.

3.2 Output

The output will consist of the same lines and words as the input file. However, the letters within each word must be reversed.

3.3 Example

Suppose the input file consisted of the following sequence of words:

```
I love you.
You love me.
We're a happy family.
```

The output would be

```
I evol .uoy
uoY evol .em
er'eW a yppah .ylimaf
```

4 The Department of Redundancy Department

Write a program that will remove all duplicates from a sequence of integers and print the list of unique integers occuring in the input sequence, along with the number of occurences of each.

4.1 Input

The input file will contain a sequence of integers (positive, negative, and/or zero). The input file may be arbitrarily long.

4.2 Output

The output for this program will be a sequence of ordered pairs, separated by newlines. The first element of the pair must be an integer from the input file. The second element must be the number of times that that particular integer appeared in the input file. The elements in each pair are to be separated by tab characters. The integers are to appear in the order in which they were contained in the input file.

4.3 Example

Suppose the input file consisted of the following sequence of numbers:

```
3 1 2 2 1 3 5 3 3 2
```

The output would be

```
3       3
1       2
2       3
5       1
```

5 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.

5.1 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.

5.2 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.

5.3 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.