ACM International Collegiate
Programming Contest 2000/2001

Sponsored by IBM

Problem Set

Arab and North African Regional Contest

Supported by Médi telecom, RAM, BP and EMO

Al Akhawayn University in Ifrane, Morocco

November 19th , 2000

This problem set should contain eight problems and ten numbered pages

Problem A: **Convert**

Source file: **convert.c/cpp/java**
Input File: **convert.in**
Output File: **convert.out**

The importance of postfix and prefix notation in parsing arithmetic expressions is that these notations are completely free of parentheses. Consequently, an expression in postfix (or prefix) form is crucial because having a unique form for an expression greatly simplifies its evaluation. But we humans prefer to read and evaluate an infix expression especially when it is parenthesized.
We would like to try the conversion from the postfix format to the parenthesized infix format for expression written in some functional language. The language will consist of the unary function INV and the binary functions ADD and MUL.

Input

The input shall consist of several cases. Each case will be represented on a separate line. The number of cases will be given in the first line of the input.

Output

For each input case, the converted expression should be on a separate line, and should have a space after each comma.

Sample Input

```
2
2 –3 Add Inv  6 10 Mul Mul
100 1000 MUL 10 ADD INV
```

Sample Output

```
Mul ( Inv (Add (2, -3)), Mul (6, 10))
Inv ( Add (10, Mul (100, 1000)))
```

Problem B:
Text Justification/Wrapping

Source File: **text.c / cpp / java**
Input file: **text.in**
Output filr: **text.out**

This problem is about right justification of the text. By that we mean that the printed form of the text is such that the right margin is aligned for all lines in the output. This task is normally achieved by first attempting to split words across lines, and then by leaving a certain amount of spaces between words.

Assumptions
For this problem, assume that words are not to be split between lines and that each line is to be both left and right justified, except for the last line of text. Any extra blank characters which are required in the justification of the text are to be distributed as uniformly as possible between the words of a line. Furthermore, we assume that there are no paragraphs and that pagination and indentation are not required. Finally, each word in the input text is separated from every other word by a blank, and each punctuation symbol is followed by a blank. For the output, assume 50 characters per line.

Sample Input

Sultan Qaboos university is a young university compared to universities found in the ArabianGulf region. However, it has an excellent reputation in the region. The university enjoys solid infrastructure, stable academic platforms, and state of the art facilities. The university includes six colleges, four centers, three libraries, and a hospital. A new college of information technology seems to be on the horizon, and may be announced any time.

Sample Output

The output must assume 50 characters per line:

Sultan Qaboos university is a young university compared to universities found in the ArabianGulf region. However, it has an excellent reputation in the region. The university enjoys solid infrastructure, stable academic platforms, and state of the art facilities. The university includes six colleges, four centers, three libraries, and a hospital. A new college of information technology seems to be on the horizon, and may be announced any time.

Problem C: **Logic World**

Source file: logicworld.c / cpp / java
Input file: logicworld.in
Output file: logicworld.out

"Logic World" is a company that produces systems with extensive use of bitwise operators. Before hiring engineers, the company tests their ability to write programs that manipulate bit-strings to evaluate both their programming skills and their mastery of the bitwise operators: OR, AND, NOR, NAND and XNOR. The test gives three bit-strings of length 40 represented in hexadecimal, and five operations to be performed on these bit-strings. Given the strings S1, S2 and S3, the operations to be performed will be: S1 AND S2, S2 OR S3, S1 AND (SE OR S3), S1 NAND S2 NOR S3, (S1 NOR S2) XNOR (S2 NOR S3)

Challenge yourself by taking this test and see whether you would be hired by "Logic World"

Input

The input data consists of a set of cases terminated by a case of only X's. Each case is made of 3 bit-strings separated by a comma and a space.

Output

For each case, results of the five operations should be given as shown in the sample output. Leave a blank line after each case.

Sample Input

```
FFFFFF4444, 222F0FFF11, AABBCCDDEE
XXXXXXXXXX, XXXXXXXXXX, XXXXXXXXXX
```

Sample Output

```
Case 1:
Operation 1: 222B0CDD00
Operation 2: FFFFFFFF55
Operation 3: AABFCF4444
Operation 4: 0004030000
Operation 5: AABFCFFF55
```

Problem D: Maze

Source file: maze.c /cpp/java
Input file: maze.in
Output file: maze.out

As a child, did you ever dream of playing in a maze? If you had thought about it, you may have come up with the idea of marking your paths as you went along. If you were trapped, you could then go back to the last crossing and take another path. By doing this, your actually applying what is called ***backtracking.*** This problem is about trying to get out of a maze.
Given a maze and a starting point within, you are to determine whether there is a way out. There is only **one** exit from the maze. You may move horizontally or vertically (but not diagonally) in any direction in which there is an open path, but you may not move in a direction that is blocked. If you move into a position where you are blocked on three sides, you must go back the way you came (backtrack) and try another path.

Input: The should come from a text file. The file contains the original maze, represented as a square matrix of symbols, with one row of the matrix per line. Each symbol is a character that indicates whether the corresponding maze is an open path ('O'), a trap ('+'), or the exit ('E'). The size of the maze is 10X10 positions. At the end of the maze in the input file, there a series of starting position coordinates. Each line contains a pair of values, representing the row and the column of the starting position in the maze. Your task is to process each of these positions in your quest for the exit from the maze.

The example of the above input file is as follows:

O	O	+	E	+	O	O	+	+	+
O	+	+	O	+	O	+	O	O	O
O	O	O	O	O	O	+	O	+	O
+	+	+	+	+	O	+	+	O	O
O	O	O	+	O	O	O	+	O	+
O	+	O	+	O	+	+	+	O	+
O	+	O	+	O	O	O	+	O	O
+	+	O	+	+	+	O	+	+	O
O	+	O	O	O	O	O	+	+	O
O	+	O	+	+	O	+	O	O	O

1	2
10	1
10	8
7	6
1	7
8	7
7	9
9	3
7	1
2	8

Processing: For each starting position, start your move into the maze from the given starting position (in terms of a row and a column numbers) until you either get to 'E' (exit) or get trapped (cannot move). You can move only where 'O' is found.

Output: For each of the ten starting positions, print out the result of your move in the maze, by displaying either Exit or Trapped in front of the starting position.

Your program should produce the following output:

Starting Position	Result
1,2	Exit
10,1	Trapped
10,8	Trapped
7,6	Exit
1,7	Exit
8,7	Exit
7,9	Trapped
9,3	Exit
7,1	Exit
2,8	Trapped

Problem E:
Merkle-Hellman Knapsack Public Key Cryptosystem

Source file: **MHK.c / cpp / java**
Input file: **MHK.in**
Output file: **MHK.out**

In this problem you will implement the Merkle-Hellman Knapsack cryptosystem. The encryption and decryption algorithms are based on solving a knapsack problem. The knapsack problem consists of finding a way to select some of the items to be packed such that their sum (the amount of space they take up) exactly equals the knapsack capacity (the target). Formally the knapsack problems is as follows: given a set S= $\{a_1, a_2, \ldots a_n\}$ and a target sum T, where each $a_i \geq 0$, is there a selection vector V= $\{v_1, v_2, \ldots v_n\}$ each of whose elements is 0 or 1, such that $\Sigma_i \ (a_i * v_i) = T$. The idea behind Merkle-Hellman scheme is to encode a binary message as a solution to a knapsack problem reducing the ciphertext to the target sum obtained by adding terms corresponding to 1s in the plaintext. That is, blocks of plain text are converted to knapsack sums by adding into the sum the terms that match with 1bits in the plaintext as shown below:

Plaintext:		1	0	1	0	0	1
Knapsack:		1	2	5	9	20	43
Ciphertext:	49						

Plaintext:		0	1	1	0	1	0
Knapsack:		1	2	5	9	20	43
Ciphertext:	27						

The receiver will have to use the ciphertext (the target sum) to recover the plaintext.

A knapsack problem is said to be "easy" if $\forall k \ a_k > \Sigma_j^{(k-1)} \ a_j$. The solution for the easy knapsack problem is straightforward, and can be achieved in O(n). Furthermore, there exists a way for transforming an easy knapsack set S into a non-easy knapsack H simply by multiplying the elements of S by
w mod n where w and n are well chosen integers. Formally the terms of H are:
$h_i = (w * s_i) \bmod n$

For example if S = {1,2,4,9}, w=15, and n=17 then H = {15,13,9,6}.

Encryption algorithm (executed by the sender)
H is called the public key and is made public by the receiver

1. Choose w and n such that n > max (S) and n is prime, and w < n. Construct H from S, w, and n.
2. Sender uses H to encipher blocks of of m bits $P_0=[p_1,p_2 \ldots p_m]$, $P_1=[p_{m+1}, p_{m+2} \ldots p_{2m}]$ and so forth (m is the number of terms in H). as follows: $T_i = P_i * H = \Sigma_j \ (p_j * h_j)$. Thus $T_0 = P_0 * H$, $T_1 = P_1 * H$ and so on. T_i are then transmitted to the receiver via a reliable channel.

Decryption algorithm (executed by the receiver)
The tuple (S,n,w) is called the private key and is kept secret by the receiver

1. Use w and n to compute w^{-1} where $w^{-1} * w = 1 \bmod n$. if n is prime then $w^{-1} = w^{(n-2)} \bmod n$
2. Compute $A_i = w^{-1} * T_i \bmod n$
3. Find Pi by solving $A_i = P_i * S$

Input file

The input consists of *N* test cases. The number of them (*N*) is given on the first line of the input file. Each test case begins with a line containing a plaintext to be encrypted. The second line contains the number of elements (m) in the knapsack S that show in the third line. The knapsack elements are positive integers separated by space. The fourth line of each text case contains n and w in this order separated by space.

Output file

Print exactly 3 lines for each test case. The first line should contain the encrypted values of the plaintext of the set separated by space. The second line should contain the plaintext obtained by applying the decryption algorithm to the plaintext, preceded by *Recovered plain text:*. The third line should contain the value of w^{-1}

Sample Input

```
2
Salaam!
4
1 2 4 9
17 15
hello there?
4
1 2 4 9
19 7
```

Sample Output

```
29 25 22 16 22 28 22 16 22 16 22 44 9 16 0 24
Recovered plain text: Salaam!
8
23 7 23 20 23 21 23 21 23 36 9 0 29 14 23 7 23 20 29 9 23 20 15 36 0 16
Recovered plain text: hello there?
11
```

Problem F: Arithmetic Progressions

Source File : progression.c
Input File : progression.in
Output File: progression.out

An arithmetic progression is of the form a, a+b, a+2b....., a+nb where n=0,1,2,3...Assume a and b are non-negative integers (0,1,2,3,....).

Write a program that finds all arithmetic progressions of length n in the set S of bisquares. The set of bisquares is defined as the set of all integers of the form p^2 + q^2 (where p and q are non-negative integers).

Input
As input, your program should accept the length of progressions N to search for and an upper bound M to limit the search to the bisquares in the range from 0 to M. Each line of the input file contains N M. Assume M <=10,000.

Output

For each pair N, M in the input file, you should print line showing the pair N, M followed by all the progressions found. If no progressions are found, you program should write No progressions found.

Sample Input

```
8 200
13 100
```

Sample Output

```
8 200

Difference of 12:
1  13 25 37 49 61 73  85
13 25 37 49 61 73 85  97
25 37 49 61 73 85 97  109
37 49 61 73 85 97 109 121

Difference of 24:
1  25 49 73 97  121 145 169
2  26 50 74 98  122 146 170
25 49 73 97 121 145 169 193
26 50 74 98 122 146 170 194

13 100

No Progressions found
```

Problem G: Pseudo Randoms

Source File: random.cpp / c / java
Input File: random.in
Output File: random.out

Generating random numbers is ususally done through a function of the form:

$$seed(x+1) = [\ seed(x) + STEP\ ]\ \%\ MOD$$

The numbers generated will be between 0 and MOD-1. However this function will generate the same pattern of numbers in a cyclic manner. A carful choice of the STEP and MOD will minimize this problem by generating a Uniform distribution of all the values between 0 and MOD-1 inclusive.

For example, if STEP=3 and MOD=5, the function will generate the series of pseudo-random numbers 0, 3, 1, 4, 2 in a repeating cycle. So all the numbers between 0 and MOD-1 are generated including 0 and MOD-1 every MOD iterations of the function. Note that by the nature of the function to generate the same seed(x+1) every time seed(x) occurs means that if a function will generate all the numbers between 0 and MOD-1, it will generate pseudo-random numbers uniformly with every MOD iterations.

If STEP = 15 and MOD = 20, the function generates the series 0, 15, 10, 5 (or any other repeating series if the initial seed is other than 0). This is a poor selection of STEP and MOD because no initial seed will generate all of the numbers from 0 and MOD-1.

Your task is to determine if the choices of STEP and MOD will generate a uniform distribution of pseudo-random numbers.

Input
Each line of input will contain a pair of integers for STEP and MOD in that order (1<=STEP,MOD<=100000).

Output
For each line of input, your program should print the STEP value right-justified in columns 1 through 10, the MOD value right-justified in columns 11 through 20 and either "Good Choice" or "Bad Choice" left-justified starting in column 25. The "Good Choice" message should be printed when the selection of STEP and MOD will generate all the numbers between and including 0 and MOD-1 when MOD numbers are generated. Otherwise, your program should print the message "Bad Choice". After each output test set, your program should print exactly one blank line.

Sample Input

```
3 5
15 20
63923 99999
```

Sample Output

```
         3         5    Good Choice

        15        20    Bad Choice

     63923     99999    Good Choice
```

Problem H: Sigma

Source File: **Sigma.c / cpp / java**
Input File: **Sigma.in**
Output File: **Sigma.out**

Let's consider a set, that will call Sigma, which consists of an integer sequence with the following properties:

$$\begin{cases} a_0 = 1 \\ a_m = n \\ a_0 < a_1 < a_2 < \ldots < a_{m-1} < a_m \\ \forall\ k \in [1,m],\ \exists\ i\ \&\ j \in [0,k-1] \text{ such that } a_k = a_i + a_j \end{cases}$$

Given an integer n as input. Your program should construct and outputs a sequence of integers, with minimal length, satisfying the properties above for n.
Note that there can be more than one sequence satisfying the properties above. Your program should output the one with the maximal sum.

Input

The input consists of 100 test cases. Test cases are stored in an input file named Sigma.in. Each test case in the file is stored in a separate line and the number –1 marks the end of the input file.

Output

The output file should be named Sigma.out and should follow this organization:
Each test case output consist of a sequence of integers separated by one blank.

Sample Input

```
3
4
9
77
80
87
99
-1
```

Sample Output

```
1 2 3
1 2 4
1 2 4 8 9
1 2 4 6 9 17 34 68 77
1 2 4 8 16 32 64 80
1 2 4 8 16 24 28 29 58 87
1 2 4 8 16 32 33 66 99
```