Computer Science Competition Invitational A 2018 Programming Problem Set

I. General Notes

1. Do the problems in any order you like. They do not have to be done in order from 1 to 12.
2. All problems have a value of 60 points.
3. There is no extraneous input. All input is exactly as specified in the problem. Unless specified by the problem, integer inputs will not have leading zeros. Unless otherwise specified, your program should read to the end of file.
4. Your program should not print extraneous output. Follow the form exactly as given in the problem.
5. A penalty of 5 points will be assessed each time that an incorrect solution is submitted. This penalty will only be assessed if a solution is ultimately judged as correct.

II. Names of Problems

Number	Name
Problem 1	Aaron
Problem 2	Chaoxiang
Problem 3	Déshì
Problem 4	Isabel
Problem 5	Klara
Problem 6	Linus
Problem 7	Logan
Problem 8	Polina
Problem 9	Rocío
Problem 10	Stelios
Problem 11	Vicente
Problem 12	Wally

1. Aaron

Program Name: Aaron.java Input File: None

In computer science class Aaron has learned some basic output techniques and has been asked to produce the following output, which requires no input. Please help him achieve this task by writing a program for it.

```
`````````````````````
____
^^^^^^^^^^^^^^^^^^^
]]]]]]]]]]]]]]]]]]
\\\\\\\\\\\\\\\\\
[[[[[[[[[[[[[[[[
```

**Input:** None

**Output:** The six lines of output as shown above, the first consisting of twenty-one tilde characters (`), the next line twenty underscore symbols, then a row of caret symbols, followed by some right square brackets, backslashes, and finally a row of left square brackets. All of these characters can be found on the keyboard.
```

2. Chaoxiang

(pronounced "shauw sheng")

Program Name: Chaoxiang.java **Input File: chaoxiang.dat**

In science class, Chaoxiang is learning about Kelvin as it relates to Fahrenheit and has been taught two ways to convert from one to the other. He has decided to use his computer science skills and write a program, but still needs some help from you.

Write a program to input values from a file representing temperatures in Fahrenheit and express the equivalent value in the Kelvin temperature measuring system. This temperature scale was designed by Lord Kelvin (William Thomson, 1824-1907). Kelvin was a British inventor and scientist (he was born in Belfast, Northern Ireland in 1824).

The following chart represents three common temperatures in the three systems we use, with zero Kelvin representing the theoretical temperature called absolute zero, supposedly at which all molecular movement stops.

	Kelvin	Celsius	Fahrenheit
Water boils	373.16K	100°C	212°F
Water freezes	273.16K	0°C	32°F
Absolute zero	0K	-273.16°C	-459.68°F

(http://www.enchantedlearning.com/chemistry/glossary/Kelvin.shtml ... For reference AFTER the contest)

There are two generally recognized formulas to do this, shown below. Both will produce the same output, and will produce the correct values for this program.

$K = (y\ °F - 32) \times 5/9 + 273.16$
$K = (y\ °F - 32) \div 1.8 + 273.16$

Input: A data file containing several Fahrenheit values, each on one line.

Output: The equivalent Kelvin temperature measure, rounded and expressed to two places of precision.

Sample input:
```
212
32
-459.68
```

Sample output:
```
373.16
273.16
0.00
```

3. Déshì

Program Name: Deshi.java Input File: deshi.dat

Deshi has just learned about strings in CS class, and wants to test her string processing skills. She creates some random strings using all lowercase letters and makes up a rule that determines whether or not the string fits a particular criteria.

She decides that a string is "ACCEPTABLE" if it has a good balance and sequence of vowels and consonants, and not acceptable if it does not.

The criteria she decides on is as follows:

- A string is "ACCEPTABLE" if there are never more than 7 consonants together in sequence, and never more than 4 vowels together, otherwise it is "NOT ACCEPTABLE".

For example, the string "eairphnanf" contains 10 characters and is “ACCEPTABLE” since the longest vowel sequence, "eai", is only 3 characters long and the longest consonant sequence, "rphn", is only 4 characters long.

The string "dekfqkexcxeoiiecooqmjvkqujitie" is “NOT ACCEPTABLE” since there is a vowel sequence of length 5, "eoiie", exceeding her limit of 4.

The string "hqloiuuycblqjsxo" is “NOT ACCEPTABLE” since there is a consonant sequence of length 8, "ycblqjsx", longer than 7, the consonant sequence length limit.

The string "utuqleznljaqihtbqiuaoeezdldbkwdalceseecyd" is NOT ACCEPTABLE for both situations. There is a consonant sequence of length 8, "zdldbkwd", and a vowel sequence of length 6, "iuaoee", both exceeding the established limits.

Input: A data file of several strings, each containing a random sequence of only lowercase letters from the English alphabet, 'a' through 'z'.

Output: The word ACCEPTABLE or the phrase NOT ACCEPTABLE based on the rules stated above, followed by a single space, and then the string being tested.

Assumptions:

- The data file will contain no more than 50 strings.
- Each line will contain only one string.
- The length of each string will be no more than 50 and no fewer than 10 characters.

Sample Input:

```
eairphnanf
dekfqkexcxeoiiecooqmjvkqujitie
hqloiuuycblqjsxo
utuqleznljaqihtbqiuaoeezdldbkwdalceseecyd
```

Sample Output:

```
ACCEPTABLE eairphnanf
NOT ACCEPTABLE dekfqkexcxeoiiecooqmjvkqujitie
NOT ACCEPTABLE hqloiuuycblqjsxo
NOT ACCEPTABLE utuqleznljaqihtbqiuaoeezdldbkwdalceseecyd
```

4. Isabel

Program Name: Isabel.java Input File: isabel.dat

Isabel is curious about her classmates, specifically on which day of the week they were born. She needs your help in devising an algorithm to do this, given only three values for each person: the year number, month number and day number of their birthday. She was born on September 9, 2003, which she knows was a Thursday. Two other friends, Rocio and Wally, were born on Christmas Day, 2004, a Saturday, and Halloween, 2003, a Friday, respectively. Klara was born on Friday, November 1, 2002.

She wants to output a list, sorted by day of the week, and then by first name, showing a report of all her classmates' birthdays. For two or more classmates born on the same day of the week, she needs them listed alphabetically by first name. In the sample output below, Wally and Klara are both born on Friday, and so Klara is listed first, then Wally.

Input: A data file consisting of several lines, each containing four items: a first name, followed by three integers representing birth year, month, and day, with single space separation.

Output: Each name, birth day of the week, and birthday in the format shown, listed first by day of the week in order from Sunday to Saturday, and then by first name for any names born on the same day of the week. The output format is as shown, with the name left aligned starting in column 1, the colon following the name located in column 12, and single space separation for the rest of the line.

Sample input:
```
Isabel 2003 9 18
Rocio 2004 12 25
Wally 2003 10 31
Klara 2002 11 1
```

Sample output:
```
Isabel     : Thursday, September 18, 2003
Klara      : Friday, November 1, 2002
Wally      : Friday, October 31, 2003
Rocio      : Saturday, December 25, 2004
```

5. Klara

Program Name: Klara.java   Input File: klara.dat

Klara is learning how to manipulate character grids and decides to do some experimenting with her newly learned techniques. She creates some random square character patterns, anywhere from 4 to 20 as the side dimension of the square, and uses two characters, the pound sign ('#') and the dot ('.') to fill the square. She then finds the very middle of the square, discovers what that character is, and then proceeds to change every instance of that character in the grid to a dash ('-'). However, to make it a bit more of a challenge, she decides that this can only happen if it is "accessible" from the middle, going up, left, down, or right, but not diagonally. If the other character is blocking the way, acting as a wall, she can't go that way and has to find a way around somehow, if it is even possible. She discovers that sometimes she can't change all the characters because they are unreachable.

Let's look at two examples, shown in the boxes below. The size of the first grid is 4, which means the middle character is located in row 2, column 2, which contains a pound sign. All the remaining pounds signs are reachable, except for the one on the top row, which is blocked from being reached by dots, as shown in the second box.

The third box is an eleven by eleven grid, and the middle character of the box is a dot, which means she wants to change all of the dots to dashes. However, she can only reach a few of them as you can see in the second box since the pound signs are blocking the way and there is no way to reach the other dots.

Let's look at one of her squares, shown in the first box on the right. The side length of the grid is 11, which means it has eleven rows and eleven characters in each row. The middle character of the box is a dot, which means she wants to change all of the dots to dashes. However, she can only reach a few of them as you can see in the fourth box since the pound signs are blocking the way and there is no way to reach the other dots.

```
.#..
..##
..#.
..##
```

```
.#..
..--
..-.
..--
```

```
###..#.#.##
###.#..#.#.
#..##.#.#.#
#.####.#.##
##.##.#.#.#
###.#..#...
.#.##..#.##
###..#..#..
#.####.#.#.
#.....###.#
#.#..##..##
```

```
###..#.#.##
###.#..#.#.
#..##.#.#.#
#.####.#.##
##.##-#.#.#
###.#--#...
.#.##--#.##
###..#--#..
#.####-#.#.
#.....###.#
#.#..##..##
```

Input: The data file will contain several sets of data, each set consisting of an integer N, followed by an N x N square grid of characters as described above.

Output: The resulting grid after changing all possible characters, reachable from the center character, to dashes as described above, followed by the string "=====".

(Sample input and output on next page)

(Klara continued)

Sample input:
```
4
.#..
..##
..#.
..##
11
###..#.#.##
###.#..#.#.
#..##.#.#.#
#.####.#.##
##.##.#.#.#
###.#..#...
.#.##..#.##
###..#..#..
#.####.#.#.
#.....###.#
#.#..##..##
7
.#.####
#..#...
#...#.#
.####..
#.#...#
##.####
#...##.
```

Sample output:
```
.#..
..--
..-.
..--
=====
###..#.#.##
###.#..#.#.
#..##.#.#.#
#.####.#.##
##.##-#.#.#
###.#--#...
.#.##--#.##
###..#--#..
#.####-#.#.
#.....###.#
#.#..##..##
=====
.#.####
#..#...
#...-.#
.----..
#.-...#
##.####
#...##.
=====
```

6. Linus

Program Name: Linus.java Input File: linus.dat

Your little brother Linus is in the 4th grade, and his math class is learning all about fractions. Their most recent lesson was about reducing fractions to their lowest terms. Sometimes that is called simplifying the fraction. Linus has promised to do all your chores for a week if you will write a program for him that will reduce the fractions he has been assigned for homework. Of course, you have declined the opportunity because you don't want to set a bad example for Linus by enabling his cheating, but, since you are very handy with Java code let's see what that program might look like anyway.

The worksheet that Linus brought home for homework requires that all the fractions be reduced to their lowest terms and any improper fractions be written as a mixed number. They haven't learned about negative numbers yet and they don't know what to do if a denominator is zero so no need to worry about those situations. They have learned that zero divided by any number is equal to zero and that a number divided by itself is one.

Now, write a program that will reduce fractions to their lowest terms and display them in the proper format. Just don't show it to Linus!

Input: The first line of data will be a number N that represents the number of fractions to be reduced. There will be N more lines each containing two values separated by a space. The first number is the numerator and the second value the denominator. There will be no negative values and none of the denominators will be zero.

Output: For each of the lines of data print a simplified fraction or mixed number equivalent to the one given. Each proper fraction should be printed showing the numerator followed by a forward slash then the denominator with no spaces between any of the three. For example, `3/4`. If the fraction is improper, print the fraction as a mixed number. For example, `3/2` becomes `1 1/2`. Any fraction that reduces to a whole number should be printed as such with no fractional part displayed. For example, `10/5` simplifies as `2`.

Sample input:
```
6
2 4
12 14
6 8
9 27
25 4
7 7
```

Sample output:
```
1/2
6/7
3/4
1/3
6 1/4
1
```

7. Logan

Program Name: Logan.java   Input File: logan.dat

Logan collects valuables of all sorts, including marbles, jewels, rocks, whatever is of interest and value to him. He keeps them in soft leather bags of assorted colors, each color unique, and has marked each bag with the weight, estimated value of the bag, and a general description of the contents. No two bags are identical. Two might weigh the same, but will be of different values. Two might have the same value, but have different weights.

He is about to load up a tote bag with as many of the colored bags as he can carry, to sell at the local flea market, but wants to maximize the value so that he can have the most value to sell. The tote bag has a weight limit he cannot exceed. He needs your help in deciding which of the colored bags he should put into the tote bag to get the most value to sell at the flea market.

For example, let's say that his tote bag has a weight capacity of 5 pounds, and he has 3 colored bags in inventory, one that is blue, another red, and a third green. The blue bag has a value of $5 and weighs 3 pounds, the red bag is worth $3 and weighs 2 pounds, and the green bag is worth $4 and only weighs 1 pound. Altogether, the three bags weigh 6 pounds, too much to carry in the selected tote bag, which means Logan must decide the best two bags to take.

He considers each combination. The blue and red bags will fit, with a total weight of 5 pounds, and a total value of $8. The red and green bags are not as valuable a combination, with 3 pounds of weight, but only a $7 value. The best value combination is blue and green, with a combined weight of only 4 pounds, but a value of $9, the best of all.

He does have tote bags of varying weight capacities, some that can handle up to 1000 pounds, and up to one hundred different colored bags that contain various precious items. Depending on the tote bag he chooses, and the number colored bags he has in his current inventory, write a program that helps Logan make the best decision about which colored bags to place in the selected tote bag in order to maximize the value of his flea market inventory.

Assumptions:
Tote bag weight range, 1 to 1000 pounds
Capacity of any colored bag, 1 to 100 pounds
Number of different colored bags, 1 to 100
Value of any particular bag, $1 to $100
Every bag has a unique single word color description, weight, and value combination, no bag the same as another.

Input: The data file will contain an initial integer N, indicating N data sets to follow. Each data set consists of an integer T representing the total weight capacity of the tote bag, an integer I indicating how many colored bags he has in inventory, followed by I sets of data, each on one line consisting of three items, an integer V representing the total value of the colored bag in dollars, an integer W indicating the weight of the colored bag in pounds, and the color of the bag. The description of the contents of the bag will not be considered in this process.

Output: For each data set, list the original weight capacity of the bag, how much of the weight capacity is used, the total value of the contents, and the colors of the bags included, listed in original order as listed in the data file. Each element of the output is shown on its own line.

(Sample input and output on next page)

(Logan, cont)

Sample input:
```
3
5
3
5 3 BLUE
3 2 RED
4 1 GREEN
6
5
1 5 BLUE
6 4 RED
4 3 GREEN
7 2 YELLOW
3 1 ORANGE
10
6
9 2 VIOLET
4 5 INDIGO
7 3 RED
6 7 PINK
2 1 BLACK
5 4 BROWN
```

Sample output:
```
5
4
$9
BLUE
GREEN
6
6
$14
GREEN
YELLOW
ORANGE
10
10
$23
VIOLET
RED
BLACK
BROWN
```

8. Polina

Program Name: Polina.java Input File: polina.dat

In a recent CS lesson on encryption, the students were given an assignment to work with a partner to create their own encryption algorithm for encoding and decoding messages. After much thought, trial and error, Polina and her partner decided on the following encoding process.

1. For each code number following the message, convert it to an equivalent three-digit hexadecimal value. (They were careful to choose code numbers that would always convert to three digits in hex.)
2. If the first digit of the code number was even, for step 3 they would start counting from the left of the current message, otherwise they would count from the right side.
3. Switch the message characters in the places given by the 2nd and 3rd digits of the code number, counting from the side determined by step 2. (Polina was also careful to make sure these two digits for each code number were different, and also designated positions within the length of the message.)
4. Add an "A" to the left side of the message and "AA"" to the right side.
5. Add the 1st character of the hex value to the left side of the message and the 3rd character to the right side.
6. Repeat steps 1, 2, 3 and 5 for each additional code number.

For example, if the message was "INVITATIONAL" and the code number is 914 (392 in hex), the encoding result would be "3AINVITATILNAOAA2". Since the code number first digit is odd, they counted from the right 1 and 4 places, and switching the characters "L" and "O", which resulted in "INVITATILNAO". They then added an "A" to the front and "AA" to the back, and finally put the first hex digit (3) at the front and the last hex digit (2) at the back, resulting in the final encoded message, "3AINVITATILNAOAA2".

For the message "DISTRICT" followed by two code numbers, 425 (1A9) and 562 (232), the first encoded message was "1ADRSTIICTAA9", and the final one "21ADRSTIACTAI92".

Input: Several messages, each on one line, followed by one or more code numbers (no more than ten) to be used to encode the message. All messages are single words containing all uppercase letters and no symbols or spaces.

Output: The encoded message according to the rules stipulated above, with all letters in uppercase.

Sample input:
```
INVITATIONAL 914
DISTRICT 425 562
UILCONTEST 691 472 456
```

Sample output:
```
INVITATIONAL 3AINVITATILNAOAA2
DISTRICT 21ADRSTIACTAI92
UILCONTEST 112OSLICANTEUTAA388
```

9. Rocío

Program Name: Rocio.java   Input File: rocio.dat

Output star patterns have always fascinated Rocio, and recently she came up with this design. She picks two integer values A and B, such that A represents a square the size of the entire pattern, and B will be the spine length of each corner, with the middle of the pattern being a square of stars. It's a bit difficult to express this, so look at the samples below. For the two values 7 and 2, the output shows a pattern contained within a 7 by 7 area, with the center of the pattern being a square of stars, and spines of length 2 extending diagonally from each corner. She picks two number such that B is always less than half the value of A, to make the pattern possible.

Write a program that creates the pattern she devised, and end each pattern with a line of equal signs to underline the pattern exactly.

Input: Several pairs of non-negative integers A and B, each pair on one line separated by a single space, A > B, B < A/2.

Output: The pattern formed by these two integers, as described above and shown below. Each pattern is underlined with a row of equal signs of length A.

Sample input:
```
7 2
10 3
9 1
```
Sample output:
```
*     *
 *   *
  ***
  ***
  ***
 *   *
*     *
=======
*        *
 *      *
  *    *
   ****
   ****
   ****
   ****
  *    *
 *      *
*        *
==========
*       *
 *******
 *******
 *******
 *******
 *******
 *******
 *******
*       *
=========
```

10. Stelios

Program Name: Stelios.java **Input File: stelios.dat**

Stelios is researching efficient ways to make connections within graphs and needs to determine the fewest number of steps it takes to get from one node to any other node. He knows about the direct connections within a network system, but needs to know about the indirect connections as well, ones that take two or more steps to reach.

For example, in a simple graph as shown below, with nodes A, B and C, and direct connections between A and B, B and C, and A and C, it only takes one step to get from one node to any other node. The direct connections would be designated using alphabet pairs, like AB, BC and CA. The order of the letters in each pair does not matter. The link from A to B is the same as the link from B to A. The data could also be written BA, CB and AC and mean the same.

```
A  -  B
 \   /
   C
```

```
  A B C
A 0 1 1
B 1 0 1
C 1 1 0
```

This graph can be represented by a connection matrix such as the one shown above. The value 1 means there is a direct connection from one node to the next. The first 1 in the top row means that you can get from node A to node B in one step. The next 1 indicates the direct connection from node A to C. The first 1 in the second row indicates the connection from B to A, and so on.

In a four-way graph like the one shown below, indicated by these data pairs - **`AB BC CD BD`** - it may take more than one step to get from node to another, as you can see in the connection matrix. To get from A to either C or D takes two steps, which is indicated by all the 2 values in the grid: A to C, A to D, C to A, and D to A.

```
A - B  -  C
     \   /
       D
```

```
  A B C D
A 0 1 2 2
B 1 0 1 1
C 2 1 0 1
D 2 1 1 0
```

In this 6-node graph, the one-step connections are designated as AB, AC, CF, BF, BD, BE and ED, resulting in a graph and connection matrix as shown below.

```
A - B - D
|   | \ |
C - F   E
```

```
  A B C D E F
A 0 1 1 2 2 2
B 1 0 2 1 1 1
C 1 2 0 3 3 1
D 2 1 3 0 1 2
E 2 1 3 1 0 2
F 2 1 1 2 2 0
```

The longest connections in this example are between nodes C and D, or C and E, going either direction, each taking three steps.

Stelios needs your help to write a program to calculate the connection matrix given the number of nodes in the graph, and a number of alphabet pairs representing the one-step connections within the graph.

Input: Several sets of data, each set consisting of an integer N, followed by several uppercase alphabet pairs on the same line, as described and shown above, with single space separation.

(Stelios, cont)

Output: The N x N connection matrix as described above and shown in the sample outputs below, each value of the matrix representing the fewest steps it takes to get between nodes. Single space separation is required between all values on each line. Each grid is followed by a dashed line exactly the length taken by the grid, as shown in the sample output below.

Assumptions:

- 3 <= N <= 26
- The alphabet pairs will be derived from a set of sequential uppercase letters labeling the nodes of the graph, always starting with A and ending with the corresponding letter for the value of N, i.e. N = 5, last node letter is E, N = 6, last letter is F, etc. For example, a three-node graph will always use the letters A, B, and C, a four-node graph the letters A through D, a ten node graph the letters A through J, and so on.
- The most number of possible steps between any two nodes will be 9.

Sample Input:
```
3 AB BC CA
4 AB BC CD BD
6 AB AC CF BF BD BE ED
```

Sample Output:
```
0 1 1
1 0 1
1 1 0
-----
0 1 2 2
1 0 1 1
2 1 0 1
2 1 1 0
-------
0 1 1 2 2 2
1 0 2 1 1 1
1 2 0 3 3 1
2 1 3 0 1 2
2 1 3 1 0 2
2 1 1 2 2 0
-----------
```

11. Vicente

Program Name: Vicente.java Input File: vicente.dat

Bit parity is a new topic for Vicente and his fellow CS classmates, and they have been given a simple program assignment to take a binary string, determine its current parity, and make it even or odd.

Parity is a way to send data in such as a way that the receiver has some idea if the value sent is in its original form, or if there might be an error due to data corruption during transmission of the signal, perhaps due to a packet collision as is known to happen on some network systems.

A simple way to do parity is to sum all the bits of the original message, decide on either an EVEN or ODD parity, let your receiver know what that parity is, and then ensure that the transmitted data fits that parity criteria.

For example, in a bit string like this one, 10110101, with EVEN parity required, the sum of the bits is 5, which means it currently has ODD parity, and needs to be adjusted. The adjustment process is simply to add a bit to the end of the string, either a zero or a 1, making it the desired parity. In this case a 1 is attached to the end to make the total sum 6 of the bits in string, which ensures EVEN parity.

The assignment is to read two items from a data file, the bit string itself and the parity required, and then adjust the bit string accordingly and then expressing both the original string and the parity adjusted string in hexadecimal form.

The original example above, 10110101, converts to B5 in hex, and the parity adjusted string, 101101011, converts to 16B. The same string with ODD parity results in the hex string 16A.

Input: A bit string and a parity designation, either EVEN or ODD, each on one line with a single space of separation.

Assumption: The original bit string will be of length 8, 16 or 32.

Output: Two uppercased hex values representing the original string and the parity-adjusted string, separated by a single space.

Sample input:
```
10110101 EVEN
10110101 ODD
1000000001110001 ODD
```

Sample output:
```
B5 16B
B5 16A
8071 100E2
```

12. Wally

Program Name: Wally.java **Input File: wally.dat**

Wally is taking Computer Science 1 this year, and his class has gotten to the point where they need to learn about identifiers. They have learned that identifiers are the names given to various parts of programs such as variables, constants, methods and classes. Wally has also learned that the identifiers that he chooses for his programs must follow this set of rules.

- An identifier must be a sequence of characters that consist of letters, digits, underscores (_), and the dollar sign ($).
- An identifier must start with a letter, underscore or dollar sign. It cannot start with a digit.
- An identifier cannot be a keyword (reserved word).
- An identifier cannot be `true`, `false`, or `null`.

The CS teacher has provided the class with a list of keywords to learn, which are listed below.

abstract	do	implements	protected	throws
assert	double	import	public	transient
boolean	else	instanceof	return	true
break	enum	int	short	try
byte	extends	interface	static	void
case	false	long	strictfp	volatile
catch	final	native	super	while
char	finally	new	switch	
class	float	null	synchronized	
continue	for	package	this	
default	if	private	throw	

Wally is a bit of an over achiever, and has decided to come up with a list of valid identifiers that he can choose from whenever he needs one. He has created a long list of potential identifiers and stored them into a data file and now wants to write a program that will read that list and print out only the valid identifiers. What would Wally's program look like?

Input: The list of reserved words as shown above, ended by the value 999, followed by a list with an unknown number of potential identifiers each listed on a separate line. Each line of the data file contains either a reserved word, the 999 value indicating the end of the reserved words list, or potential identifiers after the 999 sentinel value.

Output: An alphabetized list of Wally's valid identifiers shown one per line.

Sample input:
```
abstract
assert
boolean
break
byte
case
catch
char
class
continue
default
do
double
else
enum
extends
false
final
finally
float
for
if
implements
import
instanceof
int
interface
long
native
new
null
package
private
protected
public
return
short
static
strictfp
super
switch
synchronized
this
throw
throws
transient
true
try
void
volatile
while
999
x
num
while
final
count
3com
_time
$amount
this
break
7seven
8
Control
a1b2c3
C3PO
```

Sample output:
```
$amount
C3PO
Control
_time
a1b2c3
count
num
x
```