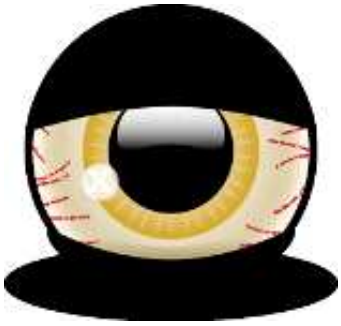


Challenge24
The Legend

Electronic Contest

10th Jubilee BME International 24-hour Programming Contest!

www.challenge24thelegend.com

Electronic Contest

Welcome to the Electronic Contest round of the 10th BME International 24-hour Programming Contest!

Rules

The Electronic Contest contains 7 problems. You have all the time in the world to solve them, but we take submissions from 10:00 to 15:00 CET. The inputs of the problems can be found in a zip file that you have probably already downloaded from the website. Each problem will have exactly 10 test cases.

You can use any platform or programming language to solve the problems. We are interested only in the output files, you don't need to upload the source code of the programs that solved them. Once you are done, you can upload your output files via the submission site:
http://x.challenge24thelegend.com/ch24ECServer. Your solutions will be evaluated on-line.

There are two major problem types:

- Problems that have an exact solution will be evaluated almost immediately, and a final score will be given for uploaded outputs. We will refer to these problems as "non-scaled problems". In the Electronic Contest, A, B, D, E and F are non-scaled problems.
- Problems that do not have a known "best" solution will be evaluated periodically and the score is scaled according to best ever uploaded output. In this case you will compete directly against the other teams. We will assign a numeric value to each uploaded solution and in every couple of minutes calculate the actual scores from them. We will refer to these problems as "scaled problems". In the Electronic Contest, C and G are scaled problems.

Note that points are awarded per output file and not per problem. If your solution only works for some of the input files, you will still be awarded points for the correct output files. A single output file however is either correct or wrong - partially correct output files are not worth any points. Achievable maximal points can differ per test case, these will be indicated at the problem descriptions.

Additional information for non-scaled problems:

Be quick about uploading the output files, because the scores awarded for every output file decrease with time. Uploading it just before the end of the contest is worth **70%** of the maximum points achievable for the test case. During the contest its value decreases linearly with time. However you should also be careful with uploading solutions. Uploading an incorrect solution is worth **-10** points. This penalty is additive, if you upload more incorrect solutions, you will receive it multiple times.

Please note that there is no point in uploading another solution for an already solved testcase because you cannot achieve more points with it. Therefore the system will not register additional uploads for solved testcases.

Additional information for scaled problems:

In this case there will be no penalty for uploading a solution later so you are able to achieve maximal points by submitting in the very last minute, if you beat the other teams' solutions.

When you upload a solution for a scaled problem, your summed points will temporarily decrease, until the next scaling occurs, when you will receive your points. Also be aware that your points might decrease in time when another team submits a better solution than yours.

Please be aware that your last submission will count which is not necessarily your best.

Tracing

For certain problems we provide you a way to get more detailed feedback (a trace or log of the evaluation) for your solution. Unless otherwise stated in the problem description, you will receive a **-5** point penalty for each trace you ask. Tracing applies to all solution files if multiple solutions are uploaded in one zip. The tracing penalty is additive as well, so be careful about your uploads when you ask for detailed feedback.

Upload frequency:

In some cases we define a minimal time gap between two submissions. This means that you are not able to upload two solutions for the same problem within this minimal time gap. This is applied for different test cases of the same problem as well.

Good luck and see you in the finals!

About the Submission site

The location of the submission site is:

http://x.challenge24thelegend.com/ch24ECServer

You will be able to log in to the submission site with your registered email address and password. After login you will see your login email and team name in the top right corner. You can submit solutions by logging in with any of your team members' account. You can also use multiple accounts to submit the solutions. The site has five pages:

Announcements

This page will show the announcements made during the contest. These can be general information, or something connected to a problem (e.g. if an ambiguity is found in the description, we will correct it by posting an Announcement).

Team status

You can see your team's status here, with all your submissions and the points received for them. If you click on the points, you will be able to download the file you have submitted. Also this is the page where you can download the detailed logs (traces) of those submissions that you have asked trace for.

Submit solution

This is where you can post your solution files. You can either upload a single output file or a .zip file that may contain multiple outputs. The naming of the output files must strictly match the following format: X99.out - where X is the problem's character code followed by a number (1 or 2 digits) identifying the test case.

You may ask for a trace by checking the checkbox here. You can download the detailed log later. Asking for trace is applied to **all** solutions submitted in the same zip file.

Standings

Here you can see the current standings of the contest. This will not be available in the last hour.

Problem information

Here you can see a summary of the various parameters of each problem

- Problem name and code
- is it a "scaled problem"
- can you ask for a trace
- minimum time between two uploads for this problem

Also you can see the maximum points that you can get for each testcase if you upload a correct solution at that time.

Contact

You can subscribe to the public mailing list of the contest by sending an email to `ec-subscribe@lists.challenge24thelegend.com`. The address of the list is `ec@lists.challenge24thelegend.com`.

During the contest we will be available on IRC on the `irc.challenge24thelegend.com` server (using the default port, `6667`). For general discussion about the contest use the `#challenge24` channel, for problem specific discussion use `#a`, `#b`, `#c`, `#d`, `#e`, `#f` and `#g` channels.

A. Hotel

You are the owner of a hotel and have N rooms which have to be numbered. Unfortunately many of your customers are very superstitious. They believe that bad luck will come if their room number contains the number 13 in any way. If a number contains a 1 and anywhere after it a 3 it will bring bad luck.

A few examples:

- 13: unlucky
- 1253: unlucky
- 99147344: unlucky
- 31: good
- 232: good
- 879342316511: good

You have to buy digits and number the rooms. As you do not want to waste any money, you want to order the exact amount needed of each digit. Note that only unlucky numbers can be skipped, otherwise rooms should be numbered in order starting from 1.

Input

Each line of the input contains a single integer. Each integer is a different test case and gives N, the number of rooms in the hotel. The last line contains a 0 to mark the end of the test cases.

Output

For each given N the output should contain 10 lines describing the number of different digits needed to number N rooms.

```
<the number of 0s needed>
<the number of 1s needed>
<the number of 2s needed>
<the number of 3s needed>
<the number of 4s needed>
<the number of 5s needed>
<the number of 6s needed>
<the number of 7s needed>
<the number of 8s needed>
<the number of 9s needed>
```

Example input

```
20
1000
500000
0
```

Example output

```
2
12
4
1
2
2
2
2
2
2
229
312
309
270
302
300
300
300
300
300
252265
285734
354041
285734
354041
335723
262139
262078
253237
252601
```

B. LAN party

You might have heard about the conferences politicians hold on global warming. These are the events where our leaders care about nature the most - as indicated by the fervor in their speeches.

Surprisingly, it turns out that many of these speeches are so boring that not even the other politicians can listen to all of them. Some of our brightest leaders relieve the tedium by having a LAN party.

Each politician has their own desktop PC. Since the use of Wi-Fi is prohibited due to security reasons, the computers are connected with crossover cables. Each cable connects exactly two computers, but each PC can be connected to many others. Currently, they have a highly redundant system with lots of connections, but the maintainer plans to remove most of the cables.

The politicians are only allowed to keep one less cable than the total number of PCs. Thus, they can still have a connected network - however, only functional computers can transmit data between neighboring ones. Unfortunately, politicians always turn their PCs off when they're not playing. Politicians only play games within their groups - but they can only play together if there is a path of cables between any two player's computer through other computers in the group (since computers outside the playing group might be switched off).

Given the set of cables and the groups of politicians, your task is to determine which cables the politicians should keep, so that the whole network is connected, and each group (as a subnetwork) is connected in itself (so the group can play even if all outside computers are down).

Note: you can request trace for this problem for -5 points. The trace output would contain a short hint about how the solution failed.

Input

The first line contains three integers: N, M and G. N is the number of politicians (the names of the politicians are integers 1..N), M is the number of cables (numbered 1..M), and G is the number of groups.

The following M lines contain the names of the two politicians whose computers are connected by the corresponding cable.

Each of the remaining G lines represents a group. The first integer of the line is S, the size of the group, and the next S integers are the names of the participants. Note that a politician can participate in more than one group.

Output

The output should be one line containing N-1 integers: the indices of the cables the politicians should keep. You may assume that a solution exists.

Example input

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

Example output

```
1 2
```

Explanation: Politicians 2 and 3 are in a group, so they must be connected. The choice of the other cable is arbitrary.

C. Triangles

In this task, you have to approximate an image using a given number of transparent triangles.

Your input will be a JPG image and the number of triangles you can use.

Your output should consist of N lines (where N equals the number of triangles). The structure of lines is the following:

```
X1 Y1 X2 Y2 X3 Y3 R G B
```

`X1 Y1, X2 Y2, X3 Y3` are floating point numbers specifying the three vertices of the triangle. The X coordinate is mapped as follows: the left edge of the leftmost pixel is 0; the right edge of the rightmost pixel is 1. Similarly, the top edge of the topmost pixel has the Y coordinate 0, while the bottom edge to the bottommost pixel is 1. It is allowed to have vertices outside the image (in this case the triangle will be clipped by the canvas boundaries).

`R G B` are three integers, from 0 to 255, specifying the color of the triangle.

Starting with a gray (**#808080**) canvas, each triangle will be drawn using 50% alpha transparency, with sub-pixel accuracy. Your score will be determined by how close this ends up to the original image, compared to submissions from other teams.

Difference between two images is defined as the total sum of differences between pixel colors. For each corresponding pixel, the differences of the `R G B` channels are summed. The smaller the total sum, the better.

This problem is a scaled problem; your submissions are compared against each other.

Triangle counts for the problems

1. 20
2. 20
3. 1000
4. 60
5. 100
6. 50
7. 100
8. 250
9. 400
10. 1000

Example

Input image (`example.jpg`), triangle count 10, and its approximation:

```
0.871795 0.4125 0.564103 0.975 -0.269231 0.4 255 255 0
1.14103 0.325 1.30769 1.4375 0.24359 1.15 232 0 0
1.12821 0.6125 -0.192308 0.6 0.564103 0.1 255 255 0
1.03846 0.575 0.320513 0.8875 0.269231 -0.175 255 255 0
0.666667 0.5 0.692308 0.7875 0.346154 0.3625 0 0 0
-0.269231 -0.375 -0.217949 0.6875 0.730769 -0.15 237 0 0
0.0384615 0.4125 0.769231 -0.0125 0.487179 1.2125 255 255 0
0.602564 1.0375 -0.435897 0.0375 -0.025641 1.3 229 0 0
1.39744 0.8875 1.23077 0.175 -0.0641026 -0.425 231 0 0
0.923077 1.325 1.05128 1.05 0.0897436 -0.0125 141 0 0
```

For this approximation, our calculated difference is `2708674`.

Scoring

The exact formula that determines your final score is:

finalscore = $100^{(5\text{-value/min})/4}$

Where `min` is the score received by the best submission so far, and `value` is the score received by your submission.

D. Sort IT out

Dilbert complained that there was a huge mess on the company storage server, so management came up with a brilliant idea to make some order in the chaos: the lines of each text file on the server should be sorted.

Knowing that garbled text files aren't very useful (and management never withdraws a bad decision) the software engineer team worked out the following solution: for each sorted text file keep a list of the lines which should be swapped to get back the original content. This list itself should be sorted as well of course.

Dilbert asks for your help to find the minimum amount of line swaps for a given document. He notes that some documents contain repeated lines, which can make things a bit harder.

Note: you can request trace for this problem for -5 points. The trace output would contain a short hint about how the solution failed.

Input

The input is plain text (ASCII encoded and apart from the line endings only characters with byte code >=32 and <=126 can appear in the file)

Output

The output should be a sorted list of line number pairs (each line of the output should contain two decimal numbers separated by space and the lines should be sorted).

After sorting the lines of the given input text, it should be possible to restore the original content by swapping all the line pairs listed in the output in the order they appear.

Lines are numbered from 1. Sorting means lexicographical ascending order based on character codes. For example 'BOB' is less than 'alice' and '23 5' is less than '4 1'. The output should adhere to the input format.

The number of lines in the output should be minimal.

Example input

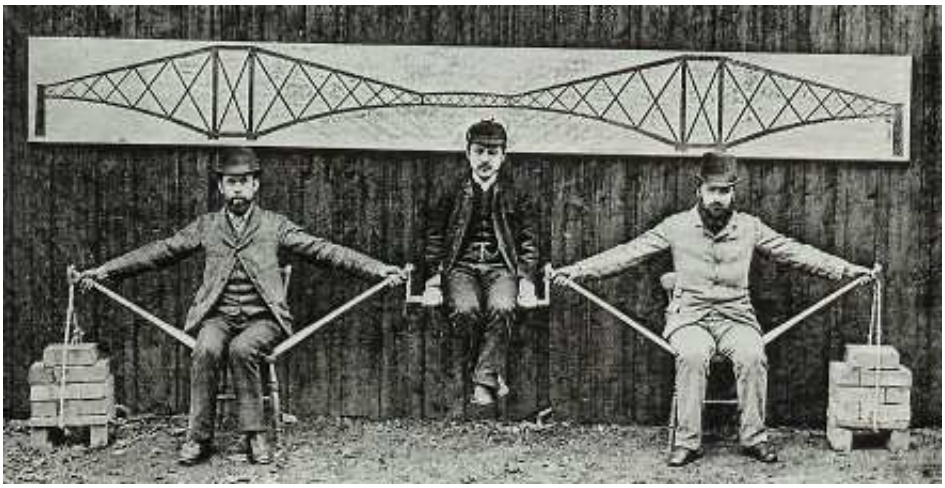
```
foo
bar
baz
egg
spam
quux
```

Example output

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

E. Bridge building

Bridge builders need your help for static load testing!

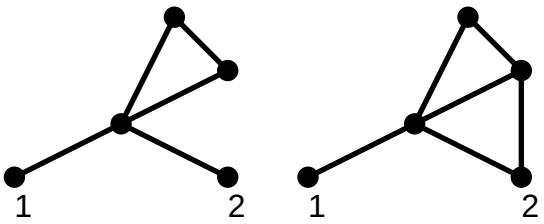


(The most famous early cantilever bridge is the Forth Rail Bridge, built in 1890. This bridge held the record for longest span in the world for seventeen years. The structural principles of the suspended span cantilever are illustrated in the photo.)

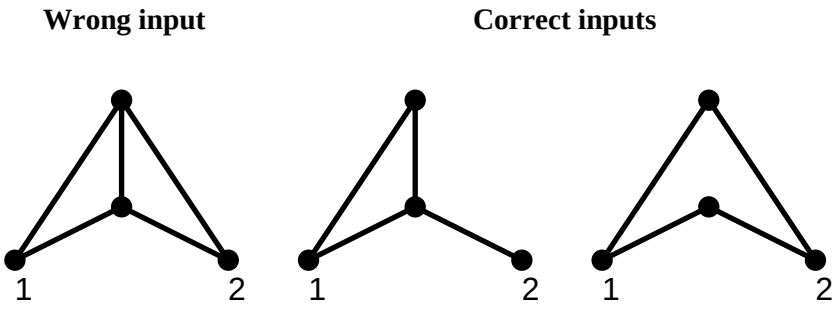
Given a grid structure built of joints and rods of different strength, you need to determine how to distribute a predefined amount of load among the joints in a way that the structure does not break. To simplify the problem, we assume the following restrictions:

- Joints and rods are weightless.
- Joints pass force, but do not pass torque from one rod to another. (A rod can only pull or push a joint in the direction of its axis)
- Rods do not deform.
- The grid structure is in the 2d plane (joints are given by x, y coordinates).
- A unit load placed in a joint results a unit force in the -y direction.
- The grid structure has exactly two fixed joints which can take arbitrary amount of force, the rest of the joints are held by the rods. (This is illustrated in the images below, where the fixed joints are marked with 1 and 2)
- The given grid structure is statically stable, no parts can be moved.

Wrong input **Correct input**



- The given grid structure is such that the forces in the rods can be uniquely determined for a given load distribution.



Note: you can request trace for this problem for -5 points. The trace output would contain a short hint about how the solution failed.

Input

Each line of the input contains three numbers.

The first line contains the number of joints (NJ), the number of rods (NR) and the load (L) which should be distributed among the joints.

The next NJ lines describe the joints. A line contains the x, y coordinates of a joint and the maximum amount of load it can take.

The remaining NR lines describe the rods. A line contains the indices of the two joints connected and the maximum force the rod can take (either push or pull force, the joints are indexed from 1 in the order they appear in the list)

The first two joints are the two fixed joints which can take arbitrary amount of force.

Output

The output should have NJ lines containing the amount of load placed at the joints in the order the joints were given. (Loads are always non-negative.)

The sum of the loads should give L and the absolute value of the forces in each rod should be less than or equal to the given maximum.

Each input can be solved. The loads and coordinates are real numbers. We will check the results with 0.001 tolerance, so a rod will break if according to our calculations the force in it is less than the maximum allowed force + 0.001.

Example input

```
5 6 3
1 0 0
3 0 0.2
0 1 1.5
2 1 1.5
4 1 1.5
1 3 1
1 4 1
2 4 1
2 5 1
3 4 1
4 5 1
```

Example output

```
0
0.2
0.7
1.4
0.7
```

F. Data recovery

Byte magazine sponsored a symposium in November 1975 in Kansas City, Missouri to develop a standard for storage of digital computer data on inexpensive consumer quality cassettes, at a time when floppy disk drives cost over $1000 USD each.

We found an old casette with digital data on it. Unfortunately it seems to be recorded before the standard came out. Help us to recover lost information!

Input format

Inputs are wav files digitized from the cassette. Storage format:

- Stored data is ASCII text and each character is encoded separately (an ASCII character code is between 0 and 127)
- The representation is not binary but ternary: 3 different frequencies are used to represent a ternary digit:
  - 1000 Hz signal means 0
  - 2000 Hz signal means 1
  - 3000 Hz signal means 2

  data rate is 250 baud, so the digits follow each other at 250 Hz
- A character frame is built up from 8 digits: a leading 0, then five data digits in MSB first format, then two closing digits (a 1 and a 2): ’0Ddddd12’ where ’D’ means the most significant data digit.
- Between characters, arbitrary long time can be filled with 3000 Hz frequency signal. So the characters arrive in an asynchronous way.

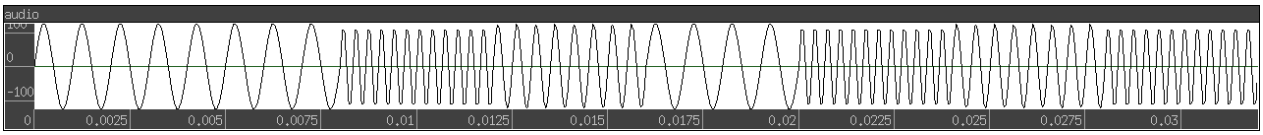
The audio is sampled at 18000 Hz frequency using 8 bit samples and it is given as mono, uncompressed wav. (Wav headers can be easily skipped by ignoring the first 44 bytes and reading the raw data directly; each byte of raw data is an unsigned 8-bit integer.)

Output format

Your task is to decode the text from the digitized audio signal found on the cassette. The media is rather old and might be a bit distorted, do not expect clean wave forms. If some characters cannot be decoded due to the noise, try to figure them out from the context.

Solutions are compared to reference outputs byte by byte, except that we are tolerant with some specific cases of extra white spaces and we are also liberal on the newline format.

Example



The above figure demonstrates character ’A’ (ASCII 65) with clean sine waves: "0 02102 12". It begins with the start bit (1000 Hz), then contains the character data (1000, 3000, 2000, 1000, 3000), closing with the two stop bits (2000 and 3000 Hz).

G. Compiler

Your task is to write a compiler for a simple programming language. The result should be assembly code that we can compile for our old and forgotten 16-bit minicomputer.

The programs of the teams compete against each other (this is a scaled problem). Submission scoring is based on the number of cycles your program takes to execute. Faster programs score better.

To ensure validity of submissions, programs have a return value, and operate on an I/O memory (the contents of which are known only to your program at runtime). Programs that produce invalid output will not be accepted.

To make your task slightly easier, we allow you to run programs in debugging mode. You can enable this by making your submission to problem "X" instead of problem "G", and enabling the "detailed log" checkbox. In this case, you will receive a trace of execution. Note however, that you will not get a score for these submissions. Also, the contents of the I/O memory may be different from the real submissions. **Please note** that by submitting to problem "X" instead of problem "G", you ensure that you will get **no** trace penalty (however, you can ask for a such trace only once every 5 minutes).

If your submitted program runs much longer than it reasonably should, we might kill it before it HALTs. These runs will be considered a failure.

Input programs

- Input programs are composed of functions, numbered from 1.
- Function 1 is the main function, the entry point, and it has no arguments.
- Function 1 never returns (it will always call the halt instruction).
- Functions are composed of a series of expressions. The only data type is the word (16-bit signed integer).
- The return value of a function is the value of its last expression.
- All functions have a given, constant number of arguments.
- Function arguments should be evaluated from left to right, before the called function is entered.
- Function arguments are passed by value. Function arguments can be changed from inside the functions, but this has no outside effects.

Problem format

Line 1 contains two integers:

number of functions number of general purpose registers

From line 2, a line for each function, with two integers:

number of arguments length of definition in words

Then a line for each function with the function definition.

Function definitions are given in prefix notation.

Example input

```
2 3
0 12
1 5
halt - + call 2 3 call 2 4 call 2 5
* get 1 get 1
```

Meaning:

- Two functions, three usable general purpose registers
- Function 1 has no arguments, it is defined in 12 words
- Function 2 has 1 argument, it is defined in 5 words
- f1() := halt(f2(3) + f2(4) - f2(5));
- f2(x) := x*x;

List of possible expressions in the input

Expression	Value / Meaning	Relevant machine instruction
`+` *expr1 expr2*	*expr1* + *expr2*	ADD
`-` *expr1 expr2*	*expr1* - *expr2*	SUB
`*` *expr1 expr2*	*expr1* * *expr2* (lower 16 bits)	MULT
`/` *expr1 expr2*	*expr1* / *expr2*	DIV
`%` *expr1 expr2*	*expr1* % *expr2*	DIV
`halt` *expr*	Call HALT with *expr*	HALT
`get` *argument_number*	Value of argument, numbered from 1	BPGET, others
`set` *argument_number expr*	Set value of argument numbered from 1 to *expr*, returned value is *expr*	BPSET, others
`call` *function_number arg1 ...*	Call function, numbered from 1, with given arguments	CALL, others
`in` *expr1*	Read memory at address *expr1*+32000	LOADAT
`out` *expr1 expr2*	Write memory at address *expr1*+32000, using value *expr2*; returned value is *expr2*	STOREAT
`>` *expr expr_t expr_f*	Conditional. 1. Evaluate *expr* 2. If result is > 0, evaluate and return *expr_t* (true branch) 3. If result is <= 0, evaluate and return *expr_f* (false branch) Example: `> + 8 -5 * 2 2 * 3 3` is 4 (if 8-5 > 0 then 2*2 else 3*3)	SGT

Note that all expressions have a return value, and thus any expression can stand as an argument to another expression. (`halt` is an exception, because its return value will never be used.)

Target platform

- 16-bit signed words (all registers and memory cells)
- Memory size: 32768 words
- 2..14 general purpose registers (number may vary in each problem)
- Each instruction runs in a specified number of cycles

Special registers:

- `SP`: stack pointer
- `BP`: base pointer

General purpose registers:

- `R0` .. `R13`

Memory layout

- Assembled program is written from word 0; entry point is 0
- Stack is downwards from word 31999
- Reserved I/O memory mapped from word 32000 to 32767
- Initial value of `SP` and `BP`: 31999

Instruction set

Instruction	Action	Used cycles	Size (words)	Memory & General			
`LOAD reg const`	Load into register from constant address.	2	2				
`LOADAT reg1 reg2`	Load into reg1 from address in reg2.	3	1				
`STORE reg const`	Store value in reg1 at constant address.	2	2				
`STOREAT reg1 reg2`	Store value in reg1 at address in reg2.	3	1				
`DATA reg const`	Load constant into register.	1	2				
`MOV reg1 reg2`	Copy value in reg1 into reg2.	1	1				
`BPGET reg const`	Load into register from address `BP+const`.	3	2				
`BPSET reg const`	Store value in register at address `BP+const`.	3	2	**Arithmetic**			
`NEG reg`	Negation: `reg := (0 - reg)`	1	1				
`ADD reg1 reg2`	Addition: `reg1 := reg1 + reg2`	1	1				
`SUB reg1 reg2`	Subtraction: `reg1 := reg1 - reg2`	1	1				

`MULT reg1 reg2`	Multiplication: `reg1 := (reg1 * reg2) & 0xffff, reg2 := (reg1 * reg2) >> 16` The case when reg1 is the same as reg2 is undefined.	1	1				
`DIV reg1 reg2`	Division: `reg1 := reg1 / reg2, reg2 := reg1 % reg2` The case when reg1 is the same as reg2 is undefined.	1	1	**Flow control**			
`JMP const`	Jump to constant address.	1	2				
`JMPI reg`	Jump to address in register.	2	1				
`SGT reg`	If value in register > 0, skip following instruction.	1	1				
`HALT reg`	Halt execution, final result is value in register.	0	1	**Stack**			
`PUSH reg`	Store register at address in `SP`, then decrease `SP` by 1.	3	1				
`POP reg`	Increase `SP` by 1, then load value at new address in `SP` into the register.	3	1				
`CALL const`	Push offset of instruction after `CALL`, then jump to constant address.	3	2				
`CALLI reg`	Push offset of instruction after `CALLI`, then jump to address in register.	3	1				
`RET`	Pop, jump to resulting address.	3	1				

Assembler

- The assembler expects 1 instruction per line.
- Empty lines are ignored.
- Beginning from a ; character, everything is ignored until the end of line (so you can emit comments).
- The assembler is not case sensitive.

Jump labels may be defined like this (on their own line):

```
labelname:
```

Jump labels must be unique. The names of jump labels can be used anywhere as the value of a constant; the address of the instruction directly following the definition of the label will be substituted.

Example output, a solution to the example input

```
function1:
    data r0 3      ; 3*3
    call function2
    push r0

    data r0 4      ; 4*4
    call function2

    pop r1         ; result of 3*3
    add r0 r1      ; add to result of 4*4, save
    push r0

    data r0 5      ; 5*5
    call function2

    pop r1         ; (3*3+4*4) - 5*5
    sub r1 r0

    halt r1        ; halt(result)

function2:
    mov r0 r1      ; mult r0 r0 is undefined
    mult r0 r1
    ret
```

Scoring

The exact formula that determines your final score is:

finalscore = $20 + 80^{(5\text{-value/min})/4}$

Where `min` is the score received by the best submission so far, and `value` is the score received by your submission.