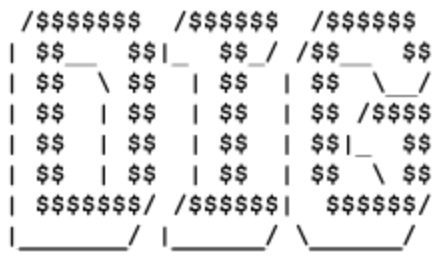
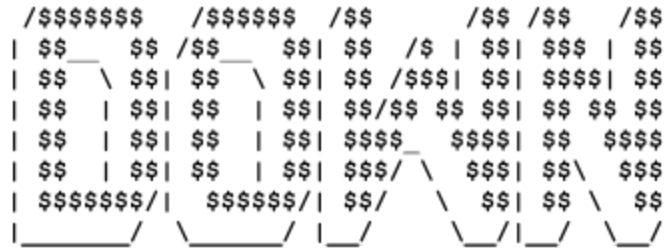
Seven Lakes High School Kickoff Classic





DO NOT OPEN THE CONTEST PACKET
UNTIL TOLD TO DO SO

1. Aww Man!

File Name: AwwMan.java **Input:** N/A

Problem Description:
“Welcome to my mine, we are mining diamonds,” Steve said, seconds before walking into a creeper.

Output Description:
You will print out an ASCII version of the Minecraft Creeper Face.

Sample Output:

```
@@@@@@@@@@@@@           @@@@@@@@@@@@@
@@@@@@@@@@@@@           @@@@@@@@@@@@@
@@@@@@@@@@@@@           @@@@@@@@@@@@@
@@@@@@@@@@@@@           @@@@@@@@@@@@@
@@@@@@@@@@@@@           @@@@@@@@@@@@@
            @@@@@@@@@@@@@
            @@@@@@@@@@@@@
            @@@@@@@@@@@@@
     @@@@@@@@@@@@@@@@@@@@@@@@@@@
     @@@@@@@@@@@@@@@@@@@@@@@@@@@
     @@@@@@@@@@@@@@@@@@@@@@@@@@@
     @@@@@@@@@@@@@@@@@@@@@@@@@@@
     @@@@@@@@           @@@@@@@@
     @@@@@@@@           @@@@@@@@
     @@@@@@@@           @@@@@@@@
```

2. Cooking Food

File Name: Cooking.java **Input:** cooking.in

Problem Description:
Steve is out for a day of mining and exploring with his friends, but he forgot his furnace and foods at home. Instead, all he has are bugged furnaces from the beta – which cook food according to some weird rules. He's not sure how it works, but it seems like the number of seconds required to cook an item is the sum of the ASCII values of all characters (including spaces). Strange.

Input Description:
Each food item is on a separate line with no empty lines before or in between valid inputs.

Output Description:
The time only in minutes and seconds (no hours) required to "cook" each food item. Each time will be on a separate line and in the form of "xmin, ysec" where x and y corresponds to the number of minutes and seconds respectively.

Sample Input:
```
Raw Beef
Raw Porkchop
Raw Chicken
Potatoes
Mushroom Stew
```

Sample Output:
```
11min, 40sec
19min, 28sec
17min, 3sec
14min, 7sec
21min, 49sec
```

3. Nearest Island

File Name: NearestIsland.java **Input:** nearestisland.in

Problem Description:
On his way towards mining redstone, Steve got stuck in the ocean with nothing but his boat and a map. Luckily, he sees several islands nearby on said map, and he wants to travel to the nearest one and set up shelter to escape all the Phantoms coming his way.

Given a file with at least 2 points, represented by coordinate pairs, return the coordinate pair for the point nearest the first given point. All points will be unique and have a unique distance to the first point. All points will be in the first quadrant of the Cartesian plane and may be doubles.

Input Description:
Coordinate pairs, one per line, at least two will be provided. The first line will be the starting point.

Output Description:
The coordinate pair of corresponding to the point, which is not the same as the first point, nearest the first point. Formatting should be “[x, y]”, where x and y are the x and y coordinates. x and y should be rounded to the nearest tenth.

Sample Input:
```
0 0
19.2 2
12 18.8734
34 0.000000012
10 3
```

Sample Output:
```
[10.0, 3.0]
```

4. Fighting Zombies

File Name: Zombies.java **Input:** zombies.in

Problem Description:
Steve loves digging for diamonds, but the cave is always full of zombies. Luckily, he has a voodoo map that lets him mess with zombies, but it's kind of glitchy at times – randomly adding letters and messing with the capitalization.

Given an input file, output each line with every occurence of any of the characters in "ZOMBIE" (case-insensitive, not including the quotes) removed.

Input Description:
Lines of text, none of which are empty

Output Description:
Each line of text (on separate lines) without any of the characters found in "ZOMBIE" (case-insensitive, not including the quotes). Note that the output may be an empty string.

Sample Input:
```
nZombie zombiEn
Hello World!
zomBies
Really long example input
You're actually reading this?
```

Sample Output:
```
n n
Hll Wrld!
s
Rally lng xapl nput
Yu'r actually radng ths?
```

5. Creeper

File Name: Creeper.java **Input:** creeper.in

Problem Description:
One day when he was digging Redstone, Steve met a creeper and died, proclaiming “Aw Man”. He’s been deathly afraid of them ever since. When he was given some hints about roaming creepers, Steve dedicated his time to killing them - earning the title Steve the Great, Killer of Creepers, Destroyer of the Boom - until he died again.

When picking up his stuff, Steve finds some letters scorched into the ground. The prophecy of Notch and Herobrine, the letters CREEPER foretell an apocalypse of future terror.

Given “HCYrHeYeHpYeHrY”, the H’s and Y’s can be removed to form “creeper” (case-insensitive), so “Aw Man” is outputted. Given “0c0r0e0e0p0”, no combination of letters can be removed to form “creeper”, so “I’m Still Ruling the World” is outputted.

Input Description:
Text containing only alphanumeric characters (no spaces), all on separate lines

Output Description:
If letters can be removed from a word so that the word “creeper” can be formed, output “Aw Man”. Otherwise, output “I’m Still Ruling the World”.

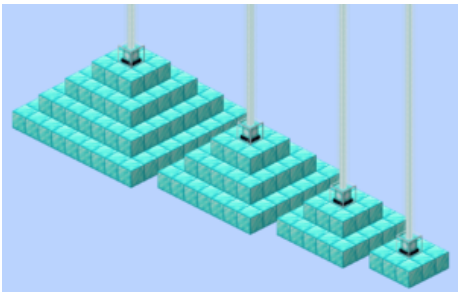
Sample Input:
```
HCYrHeYeHpYeHrY
0c0r0e0e0p0
ItsACreepeeeeeeer
IEat10CreepsADayER
IEat10CreepsADayR
```

Sample Output:
```
Aw Man
I Still Rule The World
Aw Man
Aw Man
I Still Rule The World
```

6. Beacon Crafting

File Name: Beacon.java **Input:** beacon.in

Problem Description:
Building a beacon is hard work, but Steve is willing to do this in order to make his redstone computer run faster. He's not too sure how many levels are needed for Haste, so help him calculate how many ingots he needs to craft each level of beacons as shown below.



Your job is to figure out how many ingots (one block takes 9 ingots) of materials it will take, and how many faces the pyramid will have (the pyramid is filled). Note that a level 1 beacon only has the beacon block. The picture's beacon levels are respectively 5, 4, 3, and 2 from left to right.

Input Description:
Input will consist of many lines, each with one integer ($0 \leq n \leq 2^{32}-1$) on them.

Output Description:
You will output the amount of ingots that it takes to craft the beacon blocks (not the beacon) with the level provided and the amount of faces the pyramid will have.

Sample Input:
```
1
2
100
```

Sample Output:
```
Ingots Needed: 0
Faces: 6

Ingots Needed: 81
Faces: 34

Ingots Needed: 11999691
```

Faces: 119202

7. Logical Redstone

File Name: Logical.java **Input:** logical.in

Problem Description:
Steve is copying his idol Sethbling and building a redstone computer, but he gets all the gates messed up. He is now going to use Sethbling's redstone computer in order to verify his logic circuits before he spends an afternoon placing down redstone. These circuits will be specified in Java precedence rules as Steve is a lazy man.

Input Description:
The input will start off with the number of test cases, and each test case will start with a boolean expression. This expression will have matched parentheses, &&, ||, !, and ^ Boolean operators, along with capital (A-Z) letter variables. There will be a space between tokens (letter or boolean operator). There are no spaces after opening parentheses or before closing parentheses.

After this, each input will be followed by any number of comma separated values, denoting the value of a variable in this format "{Letter}={0|1}". All letters in this list will have been used in the expression.

Output Description:
Output "true" if the specified boolean expression evaluates to true with the specified inputs, and "false" if it evaluates to false. Each output must be on its own line, with one newline in between test cases.

Sample Input:

```
2
M && I && N && E && C && R && A && F && T
M=1,I=1,N=1,E=1,C=1,R=1,A=1,F=1,T=1
M=0,I=0,N=0,E=0,C=0,R=0,A=0,F=0,T=0
M=0,I=0,N=0,E=0,C=0,R=0,A=0,F=0,T=1
A && R || (U && B && A)
A=0,R=1,U=1,B=0
A=1,R=1,U=0,B=0
```

Sample Output:

```
true
false
false

false
true
```

8. Enchantment Table

File Name: Enchantment.java **Input:** enchantment.in

Problem Description:
Steve's new enchantment table helped him enchant his diamond boots with Sharpness VI, his diamond helmet with Bane of Arthropods, and his sword with 32k Efficiency. To avoid these enchanting mishaps, he's going to decrypt secret messages and find the actual enchants.

The enchantment table in Minecraft works by taking two keys ("a" and "b") and outputting a new number corresponding to the alphabet. "a" in this case, has to be coprime with 26 (the number of letters in the alphabet). Assume the corresponding values for each letter are: A = 0, B = 1, ... Z = 25.

To decrypt, we need to find a value "c" in which (a * c) mod 26 = 1. In this case, c = 23 since (17 * 23) % 26 = 1. We can now take a letter, like "E", and plug it into the decryption formula: c * (value of letter - b) mod 26.
Sample Input Example:
a = 17, b = 19, value of letter = 4 (value of E);
23 * (4 - 19) % 26 = 23 * -15 % 26 = 19 -> T

Input Description:
First line has an integer a, followed by an integer b. The following lines are encrypted with the provided keys. Only uppercase letters and spaces are included.

Output Description:
Each decrypted message on separate lines (spaces will remain).

Sample Input:
```
17 19
NITWOGJNN M
AYTPJ TNOJBE ZZ
OWXEJBEZXG MZ
```

Sample Output:
```
SHARPNESS V
FLAME ASPECT II
PROTECTION VI
```

9. Mining Away

File Name: Mining.java **Input:** mining.in

Problem Description:
After mining 32 stacks of redstone, Steve's pickaxe broke and he is now stuck deep in a ravine. Luckily, he has some torches in his inventory that must be placed every 6 blocks. He can get out of the cave if he can place a torch every 6 blocks along his exit path, but he's unsure which way he needs to go. There will only be one exit path.

Input Description:
The input will start with n, the number of cases.
For each case, the next 2 numbers, R and C, denote respectively the rows and columns of the resulting cave.
The next number represents the number of torches in Steve's inventory.
You will receive R lines of length C, denoting the cave. The "S" represents Steve's location, and the "E" represents the cave's exit.

Output Description:
You will print out either "Take Back The Night!" or "Don't Mine At Night!" for each test case. If Steve is able to make it out alive, print "Take Back The Night!". Else, print "Don't Mine At Night!".

Sample Input:

```
1
6 12
2
############
# #        E
# ###### ###
#     ## #S#
#####      #
############
```

Sample Output:

```
Take Back The Night!
```

Solution Explanation:

```
############
# #     7890
# ######6###
#     ##5#1#
#####   432#
############
```

Steve is 10 blocks away (including the starting position + exit) from the exit and has two torches. With his torches, he can travel a maximum of 12 blocks of the cave, so he can make it out.

10. Aruban Civilization

File Name: Civilization.java **Input:** civilization.in

Problem Description:
Mr. McGee, a former Minecrafter, built m villages on Aruba, and some settlers built other villages, totalling n. However, crafty creepers have built decoy villages to trap unsuspecting travellers, but these villages are located far, far way from McGee's villages (but may be close to the settler villages). More formally, a village is an imposter if and only if the shortest path to the closest "McGee" village is longer than *k*.

The letter *n* indicates the number of villages, m indicates the number of villages built by McGee, *k* indicates the minimum distance, and e indicates the number of paths between villages. Each village is represented by an integer in the range 1..*n*.

Input Description:
The first line contains the number of datasets. For each dataset, the first line contains the integers for *n*, *m*, *k*, e. The number *n* will be less than 100. The second line contains *m* integers that represent all of the villages that are considered "Mcgee" villages. There will be at least 1 "Mcgee" village. Then the next e lines each contain three integers, *ai*, *bi*, & *wi* which indicates that there is a pathway from village *ai* to village *bi* with a distance of *wi*.

Output Description:
For each dataset, out "Village i: All hail the holy city" if the village is not a decoy, and "Village i: Torch the village" if the village is a decoy, where i is the index (starting from one) of the village.

Sample Input:

```
2
4 1 2 3
1
1 2 1
1 3 1
3 4 1
5 2 5 6
1 5
1 2 3
1 3 2
2 4 1
2 5 50
3 4 2
4 5 50
```

Sample Output:

```
Village 1: All hail the holy city
Village 2: All hail the holy city
Village 3: All hail the holy city
Village 4: All hail the holy city

Village 1: All hail the holy city
Village 2: All hail the holy city
Village 3: All hail the holy city
Village 4: All hail the holy city
Village 5: Torch the village
```

11. Bridge Crossing

File Name: Crossing.java **Input:** crossing.in

Problem Description:
Steve cheated in a command block and is trying to abscond with as many items as possible given the weight limit. Steve has a selection of cheated weapons (including 32ks and notch apples), and he wants to leave with as much valuable stuff as possible. He wants to get the maximum expense of items given his weak carrying capacity. Luckily, his command block lets him duplicate items, so he can abscond with even more merchandise.

Input Description:
First integer t denotes the number of test cases. For each test case the first line contains integers, n and c which indicates the number of items to take, and the weight capacity of Steve. The next n lines contains a one word-long name and two consecutive integers which are the weight of the item and the expense of the item.

Output Description:
Output the maximum expense that Steve can abscond with.

Sample Input:
```
1
5 24
PickAxe 13 34
Diamond_PickAxe 12 78
Bow 3 100
Golden_Apple 24 500
Diamonds 1 101
```

Sample Output:
```
2424
```

12. Mindmess Interpreter (1 of 3)

File Name: Mindmess.java **Input:** mindmess.in

Problem Description:
Mindmess – the patented Turing-complete redstone programming language (according to MumboJumbo). It's the simplest programming language ever to run a redstone computer – with just 8 operations and a simple array-based stack. Steve's computer is slightly broken, so he wants to verify these programs on Mumbo's computer before porting it for his own.

The data "pointer" is an index into an infinitely large array char array. The array must be resizable as the program executes, but there are no memory concerns.

Note: Java does not have unsigned bytes, but your code must act as if the data array uses unsigned bytes (-1 wraps around to 255 and 256 wraps to 0).

Character	Operation
>	Increment the index (to point to the next array slot)
<	Decrement the index (to point to the next array slot)
+	Increment the value at the index
-	Decrement the value at the index
.	Print the value (as a char - 65 turns to 'A') at the index
,	Read one byte and store it at the pointer
[	If the value at the index is 0, jump to the next ']'
]	If the value at the index is not 0, jump to the last '['

12. Mindmess Interpreter (2 of 3)

A sample program is given below. The array values are shown before instructions execute

```
Array: [0]
++[-]
^
Increment the value in the current index (0) by 1

Array: [1]
++[-]
 ^
Increment the value in the current index (0) by 1

Array: [2]
++[-]
  ^
If the value in the current index (0) is 0, move to the next “]”

Array: [2]
++[-]
   ^
Decrement the value in index 0 by 1

Array: [1]
++[-]
    ^
If the value in the current index (0) is not 0, move to the last “[”

Array: [1]
++[-]
  ^
If the value in the current index (0) is 0, move to the next “]”

Array: [1]
++[-]
   ^
Decrement the value in the current index (0) by 1

Array: [0]
++[-]
    ^
If the value in the current index (0) is not 0, move to the last “[”

Array: [0]
++[-]
     ^
Execution complete
```

12. Mindmess Interpreter (3 of 3)

Input Description:
Lines of Mindmess code all on one line. There are no extraneous characters. There is one empty line between each test case. If input is requested from the program, all input will be found on the next line.

Output Description:
The output of running the Mindmess code, with each output on its own line. (Note the Mindmess program may print newlines, but there should be one extra newline after every execution).

Sample Input:

```
++++++++[>++++[>++>+++>+++>+<<<<-]>+>+>->>+[<]<-]>>.>---.+++++++..+++.>
>.<-.<.+++.------.--------.>>+.>++. //All one line

>++++++++++[-<+++++++++++++++>]>++++++++++<<[->+>-[>+>>]>[+[-<+>]>+>>]
<<<<<<]>>[-]>>>++++++++++<[->-[>+>>]>[+[-<+>]>+>>]<<<<<]>[-]>>[>++++++[
-<++++++++>]<.<<+>+>[-]]<[<[->-<]++++++[->++++++++<]>.[-]]<<++++++[-<++
++++++>]<.[-]<<[-<+>] // All one line
```

Sample Output:

```
Hello World!
165
```