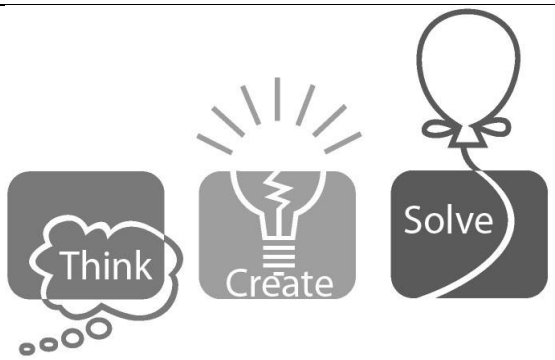
University of Central Florida



2021 Local Programming Contest (Qualifying Round)

Problems			
Problem#	Difficulty Level	Filename	Problem Name
1	Easy	raise	Nice Raises
2	Easy	strchange	Changing Strings
3	Easy	heavy	Heavy Numbers
4	Easy-Medium	cat	Cat's Age
5	Easy-Medium	ranking	Overall Ranking
6	Medium	paren	Don't Complicate it!
7	Medium	coffee	How Much Coffee is Left?
8	Medium	subs	Time to Eat
9	Medium-Hard	tower	Tower Climbing
10	Hard	palprimes	Palindromic Primes

Call your program file: filename.c, filename.cpp, filename.java, or filename.py

For example, if you are solving Time to Eat:

Call your program file: subs.c, subs.cpp, subs.java, or subs.py

UCF Local Contest (Qualifying Round) — September 4, 2021

Nice Raises[1]

filename: `raise`
Difficulty Level: `Easy`
Time Limit: `5 seconds`

Everyone knows that the UCF Programming Team coaches are the best trainers in the world and turn out some of the best problem solvers and programmers. The UCF team members are heavily sought after by different companies. FAAMGInc has arranged a nice raise package to attract the UCF team members. When it comes to giving raises, FAAMGInc gives each team member two options: a fixed raise or double their salary. Each team member can then decide which option they'd like. For example, if the fixed raise offered by FAAMGInc is $5,000, then a team member earning $3,000 would choose the fixed raise option ($5,000) but a team member making $8,000 would pick the double the salary option since that will be a better raise ($8,000 raise instead of $5,000 raise).

The Problem:

Given the fixed raise offered by FAAMGInc and the salary of each team member, determine which option is picked the most.

The Input:

The first input line contains two integers: n ($1 \leq n \leq 30$), indicating the number of team members and r ($1 \leq r \leq 50{,}000$), indicating the fixed raise offered by FAAMGInc. Each of the next n input lines contains an integer (between 1 and 200000, inclusive) indicating the current salary for a team member.

The Output:

Output will be a single integer: 1, 2, or 0. Print 1 if the majority of the team members choose the fixed raise option, 2 if the majority pick the double the salary option, and 0 if the two options are selected by the same number of team members.

If a team member will end up with the same raise choosing either option, that team member doesn't count towards either group. For example, if the fixed raise is $2000 and a team member's current salary is $2000, the fixed raise and double the salary end up with the same raise for this team member so the team member doesn't count towards either group.

[1] The financial statements in this problem are fictional.

Sample Input	Sample Output
6 1000 200 3000 500 2000 800 700	1
4 600 10 9000 20 8000	0
3 400 10000 9000 20	2
6 2500 100 100000 2500 100 2500 200000	0

UCF Local Contest (Qualifying Round) — September 4, 2021

Changing Strings

filename: `strchange`
Difficulty Level: `Easy`
Time Limit: `5 seconds`

We love our UCF and we are going to change everything to UCF!

The Problem:

Given a string, change the string to UCF as follows:

- Characters before the leftmost "U" in the string are changed to hyphen ("−").
- Characters after the rightmost "F" in the string are changed to hyphen.
- Characters between the leftmost U and the rightmost F are changed to "C".

Assume the string will contain at least one "U", at least one "F" after that U, and at least one character between the U and F.

The Input:

There is only one input line; it contains a string of uppercase letters. The string will have at least 3 and at most 50 characters.

The Output:

Print the input string converted to UCF.

Sample Input	Sample Output
ABUDEGHFXYZ	--UCCCCF---
CCUZF	--UCF
ABFABCUABABFABUABFABUAB	------UCCCCCCCCCCF-----
UABCFABCDE	UCCCF-----

UCF Local Contest (Qualifying Round) — September 4, 2021

Heavy Numbers

filename: `heavy`
Difficulty Level: `Easy`
Time Limit: `5 seconds`

Consider a positive integer *a*. We define *weight* of *a* as:

(number of digits in *a*) * (sum of the digits in *a*)

For example, if *a* = 5767, then weight of *a* is:

(4) * (5 + 7 + 6 + 7) = 100

The Problem:

Given two positive integers, determine which one weighs more, i.e., it is heavier.

The Input:

There is only one input line; it contains two integers separated by exactly one space (blank). Assume each integer is between 1 and 1,000,000 (inclusive).

The Output:

Print 1 (one) if the first number is heavier, 2 (two) if the second number is heavier, and 0 (zero) if the two numbers weigh the same.

Sample Input	Sample Output
59 1001	1
8 567	2
123 90	0

UCF Local Contest (Qualifying Round) — September 4, 2021

Cat's Age

filename: `cat`
Difficulty Level: `Easy-Medium`
Time Limit: `5 seconds`

There are several proposals for converting a cat's age to human age. One such approach is as follows:

a. The first year of cat's age is equivalent to 15 years of human age.

b. The second year of cat's age is equivalent to 9 years of human age.

c. Every year after the second year of cat's age is equivalent to 4 years of human age.

For example, if a cat is 13-year old, the cat would be {(15) + (9) + [(13 – 2) * 4)]} = 68-year old in human age. Note that the above approach implies:

a. If a cat is less than one-year old, each month of cat's age will be equivalent to 15 months of human age. For example, if a cat is 5-month old, the cat would be (5 * 15) = 75-month old in human age, which is 6-year-and-3-month old.

b. If a cat is older than one year but less than two-year old, each month of cat's age after the first year will be equivalent to 9 months of human age. For example, if a cat is 1-year-and-7-month old, the cat would be {(15 years) + [(7 * 9) months]} = {(15 years) + [63 months]} = 20-year-and-3-month old in human age.

c. If a cat is beyond 2-year old, each month of cat's age after the second year will be equivalent to 4 months of human age. For example, if a cat is 5-year-and-7-month old, the cat would be {(15 years) + (9 years) + [(5 – 2) * 4 years] + [(7 * 4) months]} = 38-year-and-4-month old in human age.

The Problem:

Given a cat's age, convert it to human age.

The Input:

There is only one input line; it contains two integers: y ($0 \le y \le 20$), indicating the year part of the cat's age and m ($0 \le m \le 11$), indicating the month part of the cat's age.

The Output:

Print two integers indicating the cat's age in human age (years and months). Note that the output value for months should not exceed 11.

Sample Input	Sample Output
13 0	68 0
0 5	6 3
1 7	20 3
5 7	38 4
20 11	99 8

UCF Local Contest (Qualifying Round) — September 4, 2021

Overall Ranking

filename: `ranking`
Difficulty Level: `Easy-Medium`
Time Limit: `5 seconds`

The Regional Competitions of ICPC (International Collegiate Programming Contest) allow universities to enter more than one team in the contest. The scoresheet for the contest will list the ranking for each team. For example, if UCF has three teams, GT two teams, and Auburn four teams, the final team ranking may look like:

```
1. Auburn
2. GT
3. UCF
4. UCF
5. UCF
6. Auburn
7. Auburn
8. GT
9. Auburn
```

The regional contest does not show the "overall performance" of each university which is the average of the team rankings for the university. For the above scoresheet, Auburn has the overall performance of "(1 + 6 + 7 + 9) / 4 = 5.75"; UCF has the overall performance of "(3 + 4 + 5) / 3 = 4.0"; and GT has the overall performance of "(2 + 8) / 2 = 5.0". So, the overall ranking of the universities will be:

```
1. UCF
2. GT
3. Auburn
```

Note that lower average indicates better performance, hence higher university ranking.

Assume that the universities will not have the same overall performance, i.e., no need for tiebreaker.

The Problem:

Given the scoresheet for a contest with each university having one or more teams, find the university ranking based on the overall performances.

The Input:

The first input line contains an integer, n ($2 \le n \le 100$), indicating the number of teams in the regional contest. The teams are listed on the following n input lines, one per line, in the order of

team ranking at the regional. Each team name will be 1 to 20 letters (uppercase and/or lowercase). Assume there will be at least two universities in the input.

The Output:

Print the universities, one per line, in the order of overall performance.

Sample Input	**Sample Output**
9 Auburn GT UCF UCF UCF Auburn Auburn GT Auburn	UCF GT Auburn
7 UofX UofA UofC UofB UofA UofB UofY	UofX UofC UofA UofB UofY
8 UA UB UA UA UA ua Ua uA	UB UA ua Ua uA

UCF Local Contest (Qualifying Round) — September 4, 2021

Don't Complicate It!

filename: `paren`
Difficulty Level: `Medium`
Time Limit: `3 seconds`

The number of parentheses and the level of nesting are usually a good indication of how complicated an expression is.

The Problem:

Given a string containing only "(", ")" and blanks (spaces), compute its complexity. The complexity is defined as follows:

- Each innermost set of parentheses adds 1 to the total complexity.
- Each set of parentheses containing only innermost set of parentheses adds 2 to the total complexity, i.e., one level out will add 2 to the complexity.
- Each set of parentheses at the next level out adds 3 to the complexity, and so on.

The Input:

There is only one input line; it contains a valid expression. The expression (string) will be 2-60 characters, each character being either "(", ")" or blank (space). Note that the blanks can be anywhere but the string will not exceed 60 characters (including the blanks). Assume that there will be at least one set of parentheses in the input, i.e., the input will not be all blanks.

Note that a valid expression satisfies the following:

1. All the parentheses are matched, i.e., every opening parenthesis has a corresponding closing parenthesis.
2. The matched parentheses are in the correct order, i.e., an opening parenthesis comes before its corresponding closing parenthesis.

Note again that the input is a valid expression, i.e., you do not need to check for errors.

The Output:

Print the complexity of the expression.

Sample Input	Sample Output
()()()()()	5
(() ( ) () ( ))	6
((()) () (()))	10
((())) () ((()) () (()))	17
((((((((((((((((((((())))))))))))))))))))	210

Note: The last Sample Input has parentheses nested 20 levels.

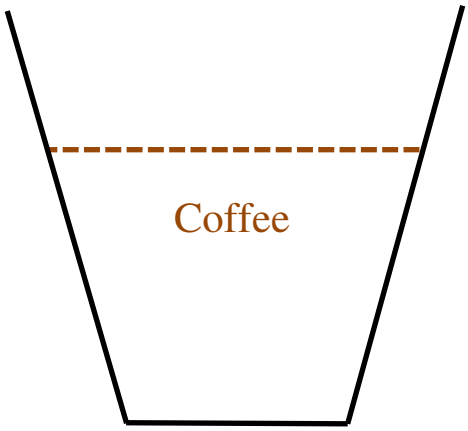
UCF Local Contest (Qualifying Round) — September 4, 2021

How Much Coffee is Left?

filename: `coffee`
Difficulty Level: `Medium`
Time Limit: `1 second`

As a budding computer scientist, you've realized that it's very important to have your morning coffee. Furthermore, it's good to know when the coffee will run out so you can properly plan when to refill.

Your coffee mug is the shape of a cross section of a cone. Here is a horizontal view:



You start the morning with a full cup. After some amount of time, you note the depth of the coffee remaining – this is the distance from the bottom of the cup to the surface of the coffee (note that the surface of the coffee is a plane parallel to the bottom of the cup). Given this information, calculate how much more time it will take before the coffee is finished, assuming that you drink coffee at the same constant rate the whole time.

The Problem:

Given the information about the cup and elapsed drinking time, determine how many minutes it will take to finish the coffee.

The Input:

There is only one input line; it contains five space separated integers:

- r ($1 \leq r \leq 100$), the radius of the bottom of the cup in inches,
- s ($r < s \leq 1000$), the radius of the top of the cup in inches,
- h ($2 \leq h \leq 1000$), the height of the cup in inches,
- m ($1 \leq m \leq 1000$), the number of minutes that coffee has been drank,
- d ($1 \leq d < h$), the depth of the remaining coffee in inches.

The Output:

Print a single floating point number on a line by itself: the number of minutes it will take to finish the cup of coffee. Any answer within an absolute or relative tolerance of 10^{-6} will be accepted.

Sample Input	**Sample Output**
3 7 5 15 4	30.284316723
10 50 100 47 50	12.469387755
5 6 12 30 7	35.081603090

UCF Local Contest (Qualifying Round) — September 4, 2021

Time to Eat

filename: `subs`
Difficulty Level: `Medium`
Time Limit: `2 seconds`

The UCF Programming Team has made it to the World Contest Finals (WF), thanks to the great team members and coaches. Fortunately for Dr. Orooji (Team Faculty Advisor), WF is in a city with streets running only horizontally and vertically (this means the chance of Dr. O getting lost is less). The city can be described as a two-dimensional grid with R rows and C columns. The team hotel is at the upper-left corner (cell with indices 1, 1). The contest will be at the bottom-right corner (cell with indices R, C).

The UCF group will start at the hotel and needs to be at the contest site. From a cell, the group can walk into one of the four neighboring cells (north, south, east, west). Note that the cells on the boundary do not have four neighbors. The group would like to make it to the contest with the fewest number of steps – moving from a cell to a neighboring cell is considered taking a step.

One complication with the trip from the hotel to the contest site is that the UCF group gets hungry and needs to eat after they've taken certain number of steps. For example, if they have to eat after taking 10 steps, then their 10^{th} step (or an earlier step) must walk them into a cell with food (sub shop).

The need to eat means the group may not be able to take the shortest path from the hotel to the contest site because the sub shops may not be on that path. Fortunately, there are enough sub shops at different spots (cells) such that the group can eat when needed and make it to the contest site, though they may take a few extra steps than the straight path from the hotel to the contest site.

The Problem:

Given the description of the city, determine the minimum number of steps needed for the UCF group to go from their hotel to the contest site, taking into account that they need to eat after taking certain number of steps (or before taking that many steps if a sub shop is not exactly at that position on the path).

The Input:

The first input line contains four integers: R ($1 \le R \le 50$), indicating the number of rows in the grid, C ($1 \le C \le 50$), indicating the number of columns in the grid, F ($1 \le F \le 100$), indicating the number of steps the group can take and then need to eat, and S ($1 \le S \le 100$), indicating the number of sub shops in the grid.

Each of the next S input lines contains two integers ($1 \le r \le R$, $1 \le c \le C$) providing the row and column number for a sub shop. Assume that all sub shops are at different locations and they are not at the hotel or contest site.

The Output:

Print the minimum number of steps needed for the UCF group to go from their hotel to the contest site.

Sample Input **Sample Output**

10 6 5 7 1 6 5 4 8 1 9 1 8 3 10 1 2 5	18
5 10 5 3 1 5 2 7 5 6	13

Explanation of the first Sample Input/Output:

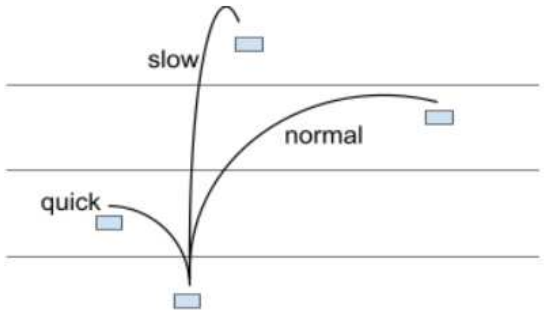
The group must take the path that goes through the sub shops at locations (2,5), (5,4) and (8,3), since they need to eat every 5^{th} step (or before 5^{th} step).

UCF Local Contest (Qualifying Round) — September 4, 2021

Tower Climbing

filename: `tower`
Difficulty Level: `Medium-Hard`
Time Limit: `1 second`

You are speed running a game where you need to jump on various platforms and reach the top of the tower. The tower has multiple uniformly spaced layers. Each layer contains a single platform. The platform is so narrow that it can be thought of as a point. Depending on upgrades, your character can jump at most k layers up (in a single jump). The time it takes to jump from one point (platform) to another point is defined by the following equation:



$$Time((x_1, y_1), (x_2, y_2)) = |x_1 - x_2| + (y_1 - y_2)^2$$

where x_1 and x_2 represent (respectively) the x location of your starting and ending points (platforms) in a jump, and y_1 and y_2 represent (respectively) the layer for the starting and ending platforms for the jump.

Note that the layers are numbered sequentially starting at 1, i.e., the y coordinate of platforms going up are 1, 2, 3, etc. Note also that, as indicated by the *Time* formula and shown in the picture, the time it takes to jump to a nearby layer is less than the time it takes to jump to a layer farther away.

You start at the first layer and finish at the last layer. You'd like to confirm that you have the optimal strategy, time wise.

The Problem:

Given the x location of the platforms and the maximum height you can jump, determine the least amount of time to reach the top of the tower.

The Input:

The first input line contains two integers: n ($2 \le n \le 20{,}000$), indicating the number of layers, and k ($1 \le k \le 8$), indicating the maximum height you can jump. Each of the next n input lines contains an integer (between 1 and 1,000,000,000, inclusive), indicating the x coordinate of a platform. The platforms are provided in order from starting point to ending point (going up the tower), i.e., the y coordinate for platforms will be 1, 2, 3, etc. (so the y coordinates are provided in the input implicitly rather than explicitly).

The Output:

Print the total amount of time it takes to reach the top (last) platform from the bottom (first) platform.

Sample Input	Sample Output
4 3 5 4 10 6	8
2 1 99 100	2

UCF Local Contest (Qualifying Round) — September 4, 2021

Palindromic Primes

filename: `palprimes`
Difficulty Level: `Hard`
Time Limit: `12 seconds`

A prime number is an integer greater than 1 which is only divisible by the integers 1 and itself, and no other positive integers. A palindromic number is one that is the same when read forwards or backwards, when written with no leading zeros. A palindromic prime is a prime number that is also a palindrome. The first few palindromic primes are 2, 3, 5, 7, 11 and 101.

The Problem:

Given two integers, ***L*** and ***H***, determine the number of palindromic primes that are between L and H, inclusive.

The Input:

The first and only input line will contain two space separated integers: ***L*** ($2 \leq \boldsymbol{L} \leq 10^{12}$) and ***H*** ($\boldsymbol{L} \leq \boldsymbol{H} \leq 10^{12}$), the lower and upper bounds (inclusive) for the search.

The Output:

Print a single integer on a line by itself: the number of palindromic primes between ***L*** and ***H***, inclusive.

Sample Input **Sample Output**

2 101	6
238 382	3
93139 97879	15