# Practice 1

# MeMo

Write a program that reads 5 integers as input, and outputs their mean (denoted by `me`) and mode (denoted by `mo`). The output will be in `float`, rounded up to three decimal places. You can assume that the mode will be unique.

[Hint : To compute mode make use of 5 extra variables to store the count of respective variables. For example `countA` to store the number of occurrences of variable `a`. To do so compare `a` with each of `b,c,d,e` and update `countA` accordingly]

**Input**:
```
a b c d e
```

**Output**:
```
mean = me
mode = mo
```

**Sample Input 1**:
```
0 4 3 4 4
```

**Sample Output 1**:
```
mean = 3.000
mode = 4.000
```

**Sample Input 2**:
```
-25 -2 -10 -25 -1
```

**Sample Output 2**:
```
mean = -12.600
mode = -25.000
```

```
#include <stdio.h>

int main() {

 int num1 = 0, num2 = 0, num3 = 0, num4 = 0, num5 = 0;

 scanf("%d %d %d %d %d", &num1, &num2, &num3, &num4, &num5);

 // Note the division with a float

 float me = (num1 + num2 + num3 + num4 + num5) / 5.0;

 printf("mean = %.3f\n", me);

 int countA = 1;

 int countB = 1;

 int countC = 1;

 int countD = 1;

 int countE = 1;
```

```
if (num2 == num1) {
countA += 1;
}
if (num3 == num1) {
countA += 1;
}
if (num4 == num1) {
countA += 1;
}
if (num5 == num1) {
countA += 1;
}
if (num3 == num2) {
countB += 1;
}
if (num4 == num2) {
countB += 1;
}
if (num5 == num2) {
countB += 1;
}
if (num4 == num3) {
countC += 1;
}
if (num5 == num3) {
countC += 1;
}
if (num4 == num5) {
countD += 1;
}

float mo = 0.0;
```

```
if (countA >= countB && countA >= countC && countA >= countD && countA >= countE) {
mo = num1;

} else if (countB >= countC && countB >= countD && countB >= countE) {

mo = num2;

} else if (countC >= countD && countC >= countE) {

mo = num3;

} else if (countD >= countE) {

mo = num4;

}

printf("mode = %.3f", mo);

return 0;
```

# }Friendship

There are four friends Abhay, Aditya, Akhilesh, and Atin standing at different points on a 2-D plane. Abhay is standing at `(xA, yA)`, Aditya at `(xB, yB)`, Akhilesh at `(xC, yC)`, and Atin at `(xD, yD)`. The strength of the friendship between any two friends is inversely proportional to the distance between the points where they are standing. Find the two friends whose friendship is the strongest. It is guaranteed that the pair will be unique.

The distance between two points `(x1, y1)` and `(x2, y2)` is calculated as

`|x2-x1|+|y2-y1|`. **Input**: Eight integers in the order `xA, yA, xB, yB, xC, yC, xD, and yD`.

**Output**: Print in **alphabetical** order the names of the two friends whose friendship is the strongest. Print the two names in a single line, separated by a single space.

**Sample Input 1**:
```
1 1 2 2 4 4 8 8
```

**Sample Output 1**:
```
Abhay Aditya
```

**Sample Input 2**:
```
-8 2 -2 8 -1 9 -4 6
```

**Sample Output 2**:
```
Aditya Akhilesh
```

```
#include <stdio.h>

#include <stdlib.h>

int main() {

 int x1, y1, x2, y2, x3, y3, x4, y4;

 scanf("%d %d %d %d %d %d %d %d", &x1, &y1, &x2, &y2, &x3, &y3, &x4, &y4);

 int d12 = abs(x1 - x2) + abs(y1 - y2);
```

```
int d13 = abs(x1 - x3) + abs(y1 - y3);

int d14 = abs(x1 - x4) + abs(y1 - y4);
int d23 = abs(x2 - x3) + abs(y2 - y3);

int d24 = abs(x2 - x4) + abs(y2 - y4);

int d34 = abs(x3 - x4) + abs(y3 - y4);

if (d12 < d13 && d12 < d14 && d12 < d23 && d12 < d24 && d12 < d34) {

printf("Abhay Aditya");

} else if (d13 < d12 && d13 < d14 && d13 < d23 && d13 < d24 && d13 < d34) {

printf("Abhay Akhilesh");

} else if (d14 < d12 && d14 < d13 && d14 < d23 && d14 < d24 && d14 < d34) {

printf("Abhay Atin");

} else if (d23 < d12 && d23 < d13 && d23 < d14 && d23 < d24 && d23 < d34) {

printf("Aditya Akhilesh");

} else if (d24 < d12 && d24 < d13 && d24 < d14 && d24 < d23 && d24 < d34) {

printf("Aditya Atin");

} else {

printf("Akhilesh Atin");

}

return 0;
```

# }Goal Difference

Football is the most popular sport in the world. You have been asked by the Premier League head to calculate the goal difference for a span of three matches.

In the first line of input you are given an integer (say `X`) representing the team. You are then provided the scores of three matches of that team: `a - b`, `c - d`, and `e - f`, where `a`, `c`, and `e` represent goals scored by the team, and `b`, `d`, and `f` represent goals scored against them. You need to compute the goal difference: total goals scored by `X` minus the total goals scored against `X`, and output it as "Goal Difference for Team X: goal_difference".

- Be careful of extra/missing spaces and extra/missing lines
- Use the visible test cases to guide yourself
- Goal difference can be negative

**Sample Input 1**:

```
2
3 - 1, 4 - 2, 5 - 3
```

**Sample Output 1**:

```
Goal Difference for Team 2: 6
```

**Sample Input 2**:

```
1
1 - 1, 0 - 0, 1 - 1
```

**Sample Output 2**:

```
Goal Difference for Team 1: 0
```

```
#include <stdio.h>

int main() {

 int g1, g2, g3, a1, a2, a3, diff, id;

 // Scan the inputs first

 scanf("%d", &id);

 scanf("%d - %d, %d - %d, %d - %d", &g1, &a1, &g2, &a2, &g3, &a3);

 // Calculate difference = goals scored - goals conceded

 diff = (g1 + g2 + g3) - (a1 + a2 + a3);

 // Printing the final output

 printf("Goal Difference for Team %d: %d", id, diff);

 return 0;

}
```

# Problem on Fractions

You are given four numbers `a, b, c, d` which form two fractions `a/b` and `c/d`. You have to report the integral part of the decimal formed after adding the two fractions and converting the sum into decimal form. If the integral part is smaller than -2, then print "Smaller than -2", else if it is greater than 2, then print "Greater than 2". In all other cases, print the integral part itself.

The integral part of decimal is the largest integer smaller than or equal to the decimal. For example, the integral part of the decimal `-5.87` is `-6`, `-1.00` is `-1`, and `2.37` is `2`.

- Be careful of extra/missing spaces and extra/missing lines
- Use the visible test cases to guide yourself
- Do not use any data structure other than `int`
- You are not allowed to use any header file other than `stdio.h`

**Constraints**: `-1000 <= a, b, c, d <= 1000`

**Input**: Four integers in the order `a, b, c, d`.

**Output**: If the integral part of the decimal formed by converting `a/b + c/d` is less than -2, then print "Smaller than -2". If it is greater than 2, print "Greater than 2". In all other cases, print a number that is the integeral part itself.

**Sample Input 1**: `1 1 2 2`

**Sample Output 1**: `2`

**Sample Input 2**: `-1 2 -3 2`

**Sample Output 2**: -2

```
#include <stdio.h>
int main() {
 int num1 = 0, den1 = 0, num2 = 0, den2 = 0;
 scanf("%d %d %d %d ", &num1, &den1, &num2, &den2);
 int num = num1 * den2 + num2 * den1;
 int den = den1 * den2;
 if (num < -2 * den) {
 printf("Smaller than -2\n");
 } else if (num < -1 * den && num >= -2 * den) {
 printf("-2\n");
 } else if (num < 0 && num >= -1 * den) {
 printf("-1\n");
 } else if (num < den && num >= 0) {
 printf("0\n");
 } else if (num < 2 * den && num >= den) {
 printf("1\n");
 } else if (num < 3 * den && num >= 2 * den) {
 printf("2\n");
 } else if (num >= 3 * den) {
 printf("Greater than 2\n");
 }
 return 0;
}
```

# Practice 1 part 2

# Pound To Kg

Write a C Program to convert Pounds to Kilograms (1 pound = 0.454 kg).Your C program should input a single float value, this would be the value in Pounds. And the output should be a single

float value rounded upto three decimal place, this would be the value in Kilograms. For Example:
**Sample Input 1**:
```
10.0
```
**Sample Output 1**:
```
4.540
```
**Sample Input 2**:
```
2.202
```
**Sample Output 2**:
```
1.000
```

```
#include<stdio.h>

int main(){

        float pound;

        scanf("%f",&pound);


        float kg = pound * 0.454;

        printf("%.3f\n", kg);

        return 0;

}
```

# Odd Arjun

Your friend Arjun is a unique person. He believes only odd numbers are the natural numbers. He counts in the way as '1,3,5,7...'. So if you ask him what number comes after 1, he will say 3. His teacher ask **you** two questions,

1. What is nth natural number according to him?
2. What is the sum of the first n natural numbers according to him?

**Note** : Assume you believe in the conventional natural number system.

**Input** : An integer n.
**Output** : Two lines containing one integer each, corresponding to answer of first question and second question respectively.
**Sample Input 1**:
```
3
```
**Sample Output 1**:
```
5
9
```
**Sample Input 2**:
```
5
```
**Sample Output 2**:
```
9
25
```

```
#include<stdio.h>

int main(){

 int n,ans1,ans2;

 scanf("%d",&n); // scanning the n
 ans1 = 2*n - 1; // nth number is 2n-1

 ans2 = n*n; // sum of first n odd natural number is n square

 printf("%d\n",ans1); //printing answers \n for line

 printf("%d",ans2);

 return 0;

}
```

# Circle and point

Coordinates `(x, y)` of the center of a circle and its radius (say `r`) are given as input. Another point, say `(x1, y1)`, is provided as input. Write a program to find out whether the point is inside the circle, on the circle, or outside the circle. Assume `x, y, r, x1, y1` are of float data type. Print `Yes` if the point is inside or on the circle and print `No` otherwise.

There should be no newline or spaces after the `Yes` or `No` words.

Input Format: `x y r x1 y1` are separated by a single space.

**Sample Input 1:**
```
3.2 4.3 2.3 4.3 5.6
```

**Output**:
```
Yes
```

**Sample Input 2:**
```
1.2 2.3 2.0 5.3 7.6
```
**Output**:
```
No
```

```
#include<stdio.h>

int main(){

 float x,y,r,x1,y1;

 scanf("%f %f %f %f %f", &x, &y, &r, &x1, &y1);

 float d = (x1-x)*(x1-x) + (y1-y)*(y1-y);

 if(d <= r*r)

 printf("Yes");

 else
```

```
 printf("No");

return 0;

}
```

# Compound Interest

Bob has deposited `b` amount of money in a certain bank. The bank offers `r%` interest on the amount of money that is at start of the year each year. Bob wants to calculate the bank balance after `y` years have passed but he cannot figure out and needs your help. Help Bob calculate his bank balance after `y` years.

- Be careful of extra/missing spaces and extra/missing lines
- Use the visible test cases to guide yourself
- You are not allowed to use any header file other than `stdio.h` and any built-in functions

**Constraints**: `0<=b<=100000`, `0<=r<=100`, `0<=y<=4`

**Input**: Three integers in the order `b, r, y`.

**Output**: Print the amount of money rounded upto 5 decimal places after `y` years assuming Bob does not do any transaction in that time period.

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

**Sample Output 1**: `102.00000`

**Sample Input 2**: `100 50 2`

**Sample Output 2**: `225.00000`

```
#include<stdio.h>

int main() {

 float b, r;

 int y;

 scanf("%f %f %d ", &b, &r, &y);

 if(y>0){

 b = (b*(1+r/100));

 y--;

 }

 if(y>0){

 b = (b*(1+r/100));

 y--;

 }
```

```
if(y>0){

b = (b*(1+r/100));

y--;

}

if(y>0){
b = (b*(1+r/100));

y--;

}

printf("%0.5f ", b);

}
```

# Practice 2

# Bases and Bases

Write a program that converts a decimal (base-10) number "A" to a base-B number. Input will be given in the format of two integers separated by a space. Output will be base-B number.
Constraints: 2<=B<=10 and A is a whole number.

INPUT :
```
A B
```
OUTPUT :
```
C
```

Sample Input 1:
```
5 2
```

Sample Output 1:
```
101
```

Sample Input 2:
```
10 6
```

Sample Output 2:
```
14
```

```
#include <stdio.h>

int main()

{ int a=1,b=1;

 scanf("%d %d", &a,&b);

 int s=0;

 int f=1;
```

```
int rem=0;

while(a!=0){

rem=a%b;

s=s+f*(rem);

f=f*10;

a=(a-rem)/b;
}

printf("%d",s);

return 0;

}
```

# Maximum Product Pair

*Note*: Usage of arrays is disallowed.

Given a list of `n` number of **non-zero** integers (`n > 1`), find and print the pair of integers such that their product is the maximum among all possible pairs in the list.

**Input Format**:
The first line denotes the integer `n`.
The second line contains `n` non-zero integers separated by spaces.

**Output Format**:
The pair of numbers in the input list with the maximum product possible separated by a space.
**The numbers should be printed in non-decreasing order.**

**Constraints**:

- In the list of numbers given, it is guaranteed that there is a *unique* pair with the maximum product.
- The given `n` is greater than `1`.
- The given numbers are non-zero integers.

## Example 1

**Input**:
```
4
312 26 312 1
```

**Output**:
```
312 312
```

## Example 2

**Input**:
```
5
59 -49 14 123 -213
```
**Output**:

```
-213 -49
```

```
#include<stdio.h>
int main() {
 int n;
 scanf("%d", &n);
 // if there are just two numbers we compare them and print them in order accordingly
if(n == 2) {
 int a, b;
 scanf("%d %d", &a, &b);
 if(a > b)
 printf("%d %d", b, a);
 else
 printf("%d %d", a, b);
 }
 else {
 int max1 = 0, max2 = 0, min1 = 0, min2 = 0;
 int a, i = 1;
 while(i <= n) {
 // we input one number after the other while tracking the two maximum and two minimum
numbers
 scanf("%d", &a);
 if(a >= max1) {
 // if current number is greater than the maximum number update max1 and max 2
accordingly
 max2 = max1;
 max1 = a;
 }
 else if(a >= max2) {
 // if current number is greater than the second maximum number update max2 accordingly  max2 =
a;
 }
```

```
// minimum analogous to maximum
if(a <= min1) {
min2 = min1;
min1 = a;
}
else if(a <= min2) {
min2 = a;
}
i = i + 1;
}
// the maximum product pair can either be the minimum pair or the maximum pair  //
compare them and print accordingly in non-decreasing order
if(min1 * min2 > max1 * max2)
printf("%d %d", min1, min2);
else
printf("%d %d", max2, max1);
}
return 0;}
```

# Sum of Digits

Given a non-negative integer `n`, find and print the sum of the digits of

`n`.

## Example 1

**Input**:
```
5239
```

**Output**:
```
19
```

## Example 2

**Input**:
```
21301
```
**Output**:
```
7
```

```
#include<stdio.h>
```

```
int main()

{

 int n, sum = 0;

 scanf("%d", &n);

 // invariant: in the ith iteration d is the ith digit of the original number from the right hand side

while(n > 0)

 {

 // d is the current digit

 int d = n % 10;
 sum = sum + d;

 n = n / 10;

 }

 printf("%d", sum);

 return 0;

}
```

# Adjacent difference

You are given a series of integers terminated by -1. Print the maximum adjacent absolute difference between two consecutive numbers. You will always be provided with at least two numbers in the sequence.

## Example 1

Input : 1 -3 4 9 12 -3 12 33 -2 34 22 41 -1
Output : 36

Explanation : The maximum absolute difference is between -2 and 34

## Example 2

Input : 55 23 76 55 22 11 0 32 54 88 -1
Output : 53

```
#include<stdio.h>

int main(){

        int curr, prev, diff, max = 0;

        scanf("%d", &prev);
```

```
	scanf("%d", &curr);
	while(curr != -1){
		if( curr < prev )
			diff = prev - curr;
		else
			diff = curr - prev;
		if( max < diff )
			max = diff;
		prev = curr;
		scanf("%d", &curr);
}
	printf("%d", max);
	return 0;}
```

# Quadruples

You are given a series of numbers terminated by -1. Print the sum of product of quadruples formed by taken 4 in sequence.

## Example 1

Input:
1 2 3 4 5 6 7 8 9 10 11 12 -1

Output: 13584

Explanation: (1 x 2 x 3 x 4) + (5 x 6 x 7 x 8) + (9 x 10 x 11 x 12) = 13584

## Example 2

Input:
1 3 2 4 12 11 9 10 8 7 6 5 -1

Output: 13584

```
#include<stdio.h>
int main(){
	int ans = 0;
	int cur = 1;
```

```
        int x;
        scanf("%d", &x);
        int count = 0;
        while( x != -1 ){
                count = count + 1;
                cur = cur*x;
                if( count == 4 ){
                        count = 0;
                        ans = ans + cur;
                        cur = 1;
                }
                scanf("%d", &x);
        }
        printf("%d", ans);
        return 0;
}
```

# Prime Mod

You are given a prime `p >= 2`, and a positive number `N < p`.

You have to output the smallest non-negative integer `k` such that `(2 ^ k) mod p` is `N`, if such a `k` exists. If such a `k` does not exist, you have to output `-1`.

## Constraints

```
2 <= p <= 200
N < p
```

## Input Format

The input consists of two numbers, a prime `p >= 2` and a positive number `N < p` separated by a single space. (You can assume that `p` is a prime, and that `0 < N < p`, you **do not** have to check whether these conditions are true.)

## Output Format

A single integer `k`

| # | INPUT | EXPECTED OUTPUT | ACTUAL OUTPUT | |
|---|---|---|---|---|
| 1 | 11 4 | 2 | 2 | |
| 2 | 11 1 | 0 | 0 | |

```
#include <stdio.h>


int main()
{
        int p;
        int N;
        int current_power=1;
        int num_iterations=0;
        scanf("%d",&p);
        scanf("%d",&N);
/*
The key to the solution is to know that there can be atmost p different remainders.  So, we
need to check only powers of 2 from 0 to p - 1. If not found here, then answer is -1.   */
int found = 0;
        while(num_iterations < p && found == 0){
                if(current_power == N){
                        printf("%d\n",num_iterations);
                        found = 1;
                }
                current_power = (current_power * 2) % p;
                num_iterations = num_iterations + 1;
        }
if(found == 0) printf("-1");
        return 0;
```

}

# Fibonacci Sequence

A Fibonacci Sequence is a sequence such that the current number is sum of previous two numbers in the sequence. Write a program to print the `n-th` number in the sequence. Take the first Fibonacci number to be `0` and second to be `1`.

## Input Format

A single integer `n`

## Output Format

A single integer denoting `n-th` Fibonacci number

| # | INPUT | EXPECTED OUTPUT | ACTUAL OUTPUT | |
|---|---|---|---|---|
| 1 | 1 | 0 | 0 | |
| 2 | 2 | 1 | 1 | |

```
#include <stdio.h>

int main()

{

 int n;

 scanf("%d", &n);

 if(n == 1){

 printf("0\n");

 }

 else if(n == 2){

 printf("1\n");

 }

 else{
```

```
int a = 0;

int b = 1;

int c;

n = n - 2; // left iterations

while(n != 0){

c = a + b;

a = b;

b = c;

n = n - 1;

}

printf("%d\n", c);

}

return 0;

}
```

# Running Variance

Write a program that calculates variance of a stream of numbers where the stream will end in 0.
Input will be given in the format of a stream of integers separated by a space and ending in 0.
The output will be a real number which will be a variance of the numbers. The output must be printed upto two decimal points only.
Constraints: Assume that stream ends when 0 is encountered.
Note: This question can be solved without arrays.

Sample Input 1:

```
-1 1 0
```

Sample Output 1:

```
1.00
```

Sample Input 2:

```
1 2 3 4 5 0
```

Sample Output 2:

```
2.00
```

```
#include <stdio.h>

int main()

{ int sum=0,sum_sq=0,val=0;

 float num=0;

 scanf("%d ", &val);

 while(val!=0){
```

```
num++;
sum=sum+val;
sum_sq=sum_sq + val*val;
scanf("%d", &val);
}
if (num==0)printf("0");
else printf("%0.2f",(sum_sq/num)-((sum/num)*(sum/num)));
return 0;
}
```

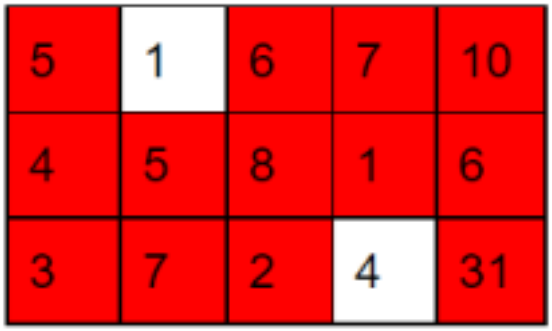
# Practice 3

# Hail Swastika

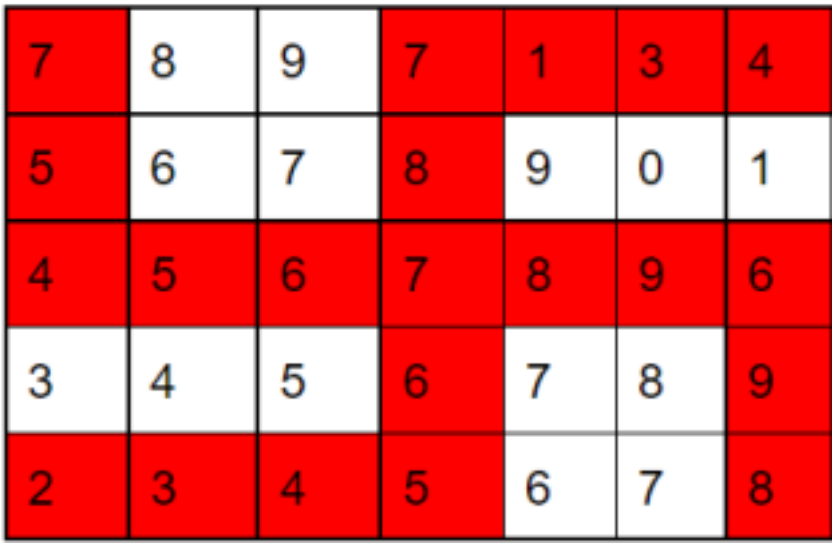
## Hail Swastika

You are given a matrix with odd number of columns and rows. You have to print the sum of elements that appear in a `swastika` pattern.

for 3*5 matrix given below, the swastika pattern is:




for 5*7 matrix given below, the swastika pattern is:




Note: Swastika pattern involves middle row and middle column.

# Input Format

first line two odd natural numbers `n` and `m` separated by space. Next `n` lines contain `m` integers separated by space.

It is guaranteed that $n \geq 3$ and $m \geq 3$.
For eg.
3 5
1 2 3 4 5
6 7 8 9 10
11 12 13 14 15

# Output Format

A single integer: sum of elements that appear in a swastika pattern.

# Sample Input

3 5 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

# Sample Output

104

| # | INPUT | EXPECTED OUTPUT | ACTUAL OUTPUT | |
|---|---|---|---|---|
| 1 | 3 5 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 | 104 | 104 | |

| | | | | |
|---|---|---|---|---|
| 2 | 5 5 3 -5 3 4 1 44 1 3 -45 1 -789 9999 10 1 1 -981918 9028 333 -9 0 45 45 0 -45 -45 | 9658 | 9658 | |
| 3 | 3 7 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 | 187 | 187 | |

```
#include <stdio.h>
int main(){
 int n,m;
 scanf("%d %d",&n,&m);
 int sum = 0;
 for(int i=0;i<n;i++){
 for(int j=0;j<m;j++){
 int element;
 scanf("%d",&element);
 if(j==m/2 || i == n/2){
 sum += element;
 continue;
 }
 if(j==0 && i<n/2){
 sum += element;
 continue;
 }
 if(i==0 && j>m/2){
sum += element;
 continue;
 }
 if(i==n-1 && j<m/2){
 sum += element;
 continue;
```

```
}
if(j == m-1 && i>n/2){
sum += element;
continue;
}
}
}
printf("%d\n",sum);
return 0;
}
```

# Trace-Trace

# Trace-Trace

You are given a square matrix with `n` rows and `n` columns. Your task is to calculate the absolute difference between the sum of square of trace elements of the matrix and the sum of square of anti-trace elements of the matrix.

## Input Format

First line contains a single integer `n` denoting the number of rows and columns of the matrix. Next `n` lines contain `n` integers separated by space.

## Output Format

A single integer: absolute difference of sum of square of trace elements and sum of the square of anti-trace elements.

## Sample Input

```
3
1 2 3
4 5 6
7 8 9
```

## Sample Output

```
24
```

## Explanation:

`1*1+ 5*5 + 9*9 - 3*3 - 5*5 - 7*7 = 24`

```
#include <stdio.h>
#include <stdlib.h>
int main(){
 int n;
 scanf("%d", &n);
 int tr_sq = 0, anti_tr_sq = 0;
 for(int i=0;i<n;i++){
 for(int j=0;j<n;j++){
 int x;
 scanf("%d", &x);
 if(i==j) tr_sq += x*x;
 if(i+j==n-1) anti_tr_sq += x*x;
 }
 }
 printf("%d", abs(tr_sq-anti_tr_sq));
 return 0;
}
```

# Factorial Fun

# Rwit Panda

# Factorial Fun

We know that factorial of a non-negative number `n` is defined as `n! = 1*2*3....*n`, which is the product of all natural numbers till `n`. As a special case, `0!` is defined to be `1`. You are given a sequence of non-negative integers terminated with `-1`. Find the sum of the squares of the sum of digits of factorial of each number. Do not account the terminating `-1` as a part of the sequence. For ease of understanding, see the example input.

## Input Format

A space separated stream of non-negative integers terminated with a `-1`.

## Output Format

A single integer: sum of squares of sum of digits of factorial of each number in the stream (except the `-1`).

## Sample Input

```
4 5 1 6 1 -1
```

## Sample Output

```
128
```

## Explanation

``` 4! = 24 Sum of digits = 6 5! = 120 Sum of digits = 3 1! = 1 Sum of digits = 1 6! = 720 Sum of digits = 9 1! = 1 Sum of digits = 1

So, our required answer is 6^2 + 3^2 + 1^2 + 9^2 + 1^2 = 36 + 9 + 1 + 81 + 1 =

128 #include <stdio.h>

int main()

{

int x;

int ans = 0;

scanf("%d", &x);

while(x != -1)

{

int factorial = 1;

for(int i=1;i<=x;i++) factorial *= i;

int sum_dig = 0;

while(factorial)

{

sum_dig += factorial % 10;
factorial /= 10;

}
```

```
ans += sum_dig * sum_dig;

scanf("%d", &x);

}

printf("%d", ans);

return 0;

}
```

# Aye-Aye Matrix

# Aye-Aye Matrix

You are given an `n*m` matrix with `n` rows and `m` columns. The rows and columns are 0-indexed [top left cell is referred to in this question as (0,0)]
Pirate Captain Black-Mat gives you, his most trusted underling, 2 positive integers `k` and `r`. He asks you to make a list of all the special elements in the matrix in the row-wise traversal order. A special element is an element which is in the cell (i,j) of the matrix such that `(i+j)%k = r`. By row-wise traversal, we mean that we first travel along all cells of first row (row number `0`), then along all cells of second row (row number `1`), and so on.. Also, for each row, we start travelling from the first column (columns number `0`) and move right.
Once you have made the list of all special elements, he wants you to find their interleaving sum. For the list `{1, 2, 5, 3, -1, 3, 4, 5}`, we have the interleaving sum as `1 - 2 + 5 - 3 + (-1) - 3 + 4 - 5 = -4`.

## Input Format

The first line has 2 positive integers `n` and `m`, denoting the number of rows and columns respectively.
The second line has 2 non-negative integers `k` and `r` (`r < k`, `k > 0`).
The next `n` lines each have `m` space separated integers.

## Output Format

A single integer, denoting the interleaved sum

## Sample Input

```
5 6

3 1

3 1 2 4 -4 54

-44 23 1 2 33 0

-10 2 32 2 7 666
98 11 -76 21 4 -1
```

```
77 -77 101 -65 -6 -2
```

## Sample Output

```
-526
```

## Explanation

```
For k = 3, r = 1, we have the list of special elements in row-wise traversal as
:  {1, -4, -44, 2, 32, 666, 11, 4, 77, -65}

So, the interleaving sum is : 1 - (-4) + (-44) - 2 + 32 - 666 + 11 - 4 + 77 -
(- 65) = -526
```

| | | | | |
|---|---|---|---|---|
| 2 | 1 4 8 7 1 2 -1 3 | 0 | 0 | |
| 3 | 5 4 2 0 2 33 1 90 -1 -2 -3 -4 123 -90 90 4561 132 0 0 0 -1000 999 11 1 | -975 | -975 | |

```
#include <stdio.h>

int main()

{

 int n, m, k, r;

 scanf("%d %d\n%d %d\n", &n, &m, &k, &r);

 int positive = 1;

 int x, ans = 0;

 for(int i=0;i<n;i++)

 {

 for(int j=0;j<m;j++)

 {

 scanf("%d", &x);

 if( (i+j) % k != r ) continue;

 if(positive)
```

```
{
ans += x;

positive = 0;

}

else

{

ans -= x;

positive = 1;

}

}

}

printf("%d", ans);

return 0;

}
```

# Unlucky Number 7

# Unlucky Number 7

You are given an `n*m` matrix with `n` rows and `m` columns. You are supposed to calculate the sums of each of the rows of the matrix. Then you are supposed to take summation over these sums starting from the first row and stop whenever you reach a row whose sum is `7` as it is considered bad luck. The sums for all following rows, including the row whose sum is `7` will not be added to the final answer.

## Input Format

First line contains `2` space separated integers `n` and `m` which denote the size of the `n*m` matrix. Then next `n` lines contain `m` space separated integers.

## Output Format

A single integer: The required summation over the valid rows

## Sample Input

```
3 3
1 2 3
2 3 2
4 2 4
```

## Sample Output

```
6
```

## Explanation

```
Sum of first row = 6
Sum of second row = 7
Since the sum of the second row is 7, we need only consider the sum over rows  before itself in our final answer
```

| | | | | |
|---|---|---|---|---|
| 2 | 2 2 5 3 1 4 | 13 | 13 | |
| 3 | 3 4 2 0 5 2 -1 2 3 2 -1 -2 -3 -4 | 5 | 5 | |
| 4 | 5 2 6 -8 5 4 7 0 7 0 7 0 | 7 | 7 | |

```
#include <stdio.h>
int main()
{
 int n, m;
 scanf("%d %d", &n, &m);
 int x;
 int totsum = 0;
 for(int i=0;i<n;i=i+1)
 {
 int rowsum = 0;
```

```
for(int j=0;j<m;j=j+1)

{

scanf("%d", &x);

rowsum = rowsum + x;

}

if(rowsum == 7){
break;

}

else{

totsum = totsum + rowsum;

}

}

printf("%d", totsum);

return 0;

}
```

# Caged Robot

# Caged Robot

A robot is constrained to move along a one dimensional line. It receives input commands in the form of a stream of integers. A positive integer denotes that the robot must take that many steps towards `+ve` direction and a negative integer denotes that the robot must take that many steps towards `-ve` direction. The robot starts at 0 and terminates its operations on encountering the integer `-7`. Also the robot is constrained to always remain between the positions `-10` and `+10`. That is, say if the robot is already at `+9` and it receives a command to go 3 steps in the positive direction, it will only go till `+10` for that command. You are supposed to find the final position of the robot.

## Input Format

The first line has space separated integers terminated by a `-7`.

## Output Format

A single integer, denoting the final position of the robot.

## Sample Input

```
2 3 12 -4 -7
```

## Sample Output

```
6
```

## Explanation

```
First on encountering 2, the robot reaches the position 2.

Then on encountering 3, it reaches the position 5.

Then, on encountering 12, due the constraints on itself it only reaches

10. Finally, on encountering -4, it reaches 6.

Then -7 command terminates the robot's operations cementing 6 as its
final  position.
```

```
#include <stdio.h>

int main()

{

 int x;

 int pos = 0;

 scanf("%d", &x);

 while(x != -7)

 {

 pos = pos + x;

 if(pos > 10){

 pos = 10;

 }

 else if (pos < -10){

 pos = -10;

 }

 scanf("%d", &x);

 }
```

```
printf("%d", pos);

return 0;

}
```

# Practice 4

# Word compressor

You are given a string of lowercase characters (without any spaces). You have to compress the word using the method described below.

The basic idea is to compress multiple consecutive occurences of a character to a single character and a number specifying the occurences. So, for instance the substring `aaaa` gets compressed to `a4`. Note that if a character occurs only once (consecutively), then there is no need of compression.

For example, suppose the given string is `aaaabbcdddeee`. Then the compressed output using above method will be `a4b2cd3e3`. Your task is to read the word (in the format specified below) and output the compressed string.

## Input Format

The input consists of a single line. The line is a string of lowercase letters ending with a period (`.`). Note that period is not the part of the word.

## Output format

Output the compressed word as a single string.

## Note

- All the characters are lowercase alphabet except the last `.`.
- There will be at least one letter in the word.

## Examples

- **Sample 1**

Input

```
aaaabbcdddeee.
```

Output

```
a4b2cd3e3
```

```
#include <stdio.h>
```

```
int main(){
	char c, prev;
	scanf("%c", &c);
	prev = c; // Initializing first previous character to current
	int cnt = 0; // Storing count of occurences
	// Initialized cnt to 0 as c==prev will definitely match
	// in first iteration below
	while (c != '.') { // Input remaining
		if (c == prev){
			cnt++;
		}
		else{ // Character changed
			printf("%c", prev);
			if (cnt>1) printf("%d", cnt);
			cnt = 1;
		}
		prev = c;
		scanf("%c", &c);
	}
	printf("%c", prev); // Print the last character before encountering period if (cnt>1)
	printf("%d", cnt);
	cnt = 1;
}
```

# Identify the smallest

Given a sequence of `N` number, identify the smallest number among them. Each positive integer will be smaller than `1000`.

*Input:*

Each test case will span across 2 lines. First line with specify `N` the length of sequence. Second line will contain `N` positive numbers.

*Output:*

The output of the program will be the smallest number in the sequence.

*Sample Testcase*

- Testcase1:

Input

```
10
8 34 52 98 123 10 2 19 11 23
```

Output

2

```
#include <stdio.h>
int main() {
 int N = 0, i = 0;
 int smallest = 1000;
 int curr_num = 0;
 scanf("%d", &N);
 while (i < N) {
 scanf("%d", &curr_num);
 if (curr_num < smallest) {
 smallest = curr_num;
 }
 i = i + 1;
 }
 printf("%d\n", smallest);
 return 0;
}
```

# Find Code

You are given a three-letter word (the 3 letters will be distinct from each other), after which a string of letters follow. Determine whether the three-letter word is present in the string or not. Note that all letters are uppercase alphabets and print the location of the first occurence.

## Input Format

The first line contains three letters (uppercase) without spacing.

The next line contains a single integer `n`.

The third line contains a string of `n` uppercase alphabet without spacing.

# Output format

Print a single integer as the location of the first occurence of the code (the first letter) in the string if it exists.

(Consider the first letter to be at location 1.)

Print -1 if the code is not present.

# Note

- All the characters are uppercase alphabet.
- `n` will be at least 3.
- HINT: You might need to read extra characters (new line etc.)

# Examples

- **Sample 1**

Input

```
ESC
19
ILOVEESCPROGRAMMING
```

Output

```
6
```

```
#include<stdio.h>

int main()

{

 char word[3];

 int i,n,count=0;

 gets(word);

 scanf("%d\n",&n);

 char name[n];

 gets(name);
```

```
for(i=0;i<n;i++)

{

if(name[i]==word[0] && name[i+1]==word[1] && name[i+2]==word[2]){

printf("%d",i+1);

break;}

else{

count++;

}

}

if(count==n)

printf("-1");

return 0;
}
```

# Error-checksum

Error Detection mechanism are widely employed in industry to identify the error in data communication. You have to have design a simple error detection mechanism, which uses parity digit. A parity digit is appended to the decimal representation of the number to make the total number of 9's even. Given an integers A . Your task is identify the parity bit.

## *Constraint:*

1. The integer A is always positive
2. Refrain from using floating point variables for storing A.

## *Input:*

- The input will consist of a number.

## *Output:*

- The program should print:
  - `YES`, if the parity digit is required to make the total number of digit 9's even.
  - `NO`, if parity digit is not required
- The output should be followed by a newline.

## *Sample TestCase:*

- TestCase1:
  Input

• 919

Output:

NO

• TestCase2:

Input

• 94

Output:

YES

• Explanation:

TEST CASE1: The number of 9's is 919 is even. Parity digit is not required. TEST CASE2: The number of 9's is 94 is odd. Parity digit is required.

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {

 int N;

 scanf("%d", &N);

 int num_nines = 0, mod = 0;

 while (N > 0) {

 mod = N % 10;

 N = N / 10;

 if (mod == 9) {

 num_nines++;

 }

 }

 if (num_nines % 2)

 printf("YES\n");
```

```
else
printf("NO\n");
return 0;
}
```

# Generate Email

## Problem Description:

You are given an integer n. In the next n lines you have been given the first-name, surname, birth year, and institute name for n students. You have generate an email from that data, with the following format.

(the first 3 letters of the first-name in uppercase)_(the first 3 letters of the surname in uppercase)(last 2 digits of the birth year)@(institute name in lowercase).ac.in

### *Input:*

First line with specify `n` the number of students. the next n lines will have the data as mentioned above in the following format:

first-name,surname,birthyear,institute-name,

Note : The first-name and surname will have atleast 3 characters each, and will comprise of only alphabetical characters. The institute name will comprise only of alphabetical characters. The birthyear will comprise only of numerical characters. The first-name, surname, and institue-name may not have consistent uppercase or lowercase maintained, for ex. this is a valid input:
AbcD,bACf,1999,IItK
Note : There will be no spaces in any of the lines.

### *Output:*

The emails as mentioned as mentioned above, each in new line.

Note : Do NOT add any extra spaces. There should not be a newline after the last email. *Visible Test Case*

- Testcase1:

Input

```
2
abcd,efgd,1989,IITK,
HIJK,LmNO,1999,IIITH,
```

Output

ABC_EFG89@iitk.ac.in HIJ_LMN99@iiith.ac.in

```
#include <stdio.h>
int main(){
 int n;
 scanf("%d", &n);
 int i = 0;
 int j = 0;
 char currChar;
 int emailSectionPrinted = 0;
 int emailSectionScanned = 0;
 for(i = 0; i < n; i++){
 for(j = 0; j < 4; j++){
 emailSectionPrinted = 0;
 emailSectionScanned = 0;
 do{
 //putting a space in scanf, before the format specifier, skips the whitespaces characters like  newline and space
 scanf(" %c", &currChar);
 emailSectionScanned++;
 if(currChar == ','){
 break;
 }
 if(j == 0 || j == 1){
 // the first name and the surname
 if(emailSectionPrinted < 3){
 if('a' <= currChar && currChar <= 'z'){
 currChar = ((currChar - 'a') + 'A');
 }
 printf("%c", currChar);
 emailSectionPrinted++;
 }
 if(j == 0 && emailSectionPrinted == 3){
```

```
printf("_");
emailSectionPrinted++;
}
}
else if(j == 2){
//the birth year
if(emailSectionScanned > 2){
//when the first 2 things of the birth year have been scanned  //we print
currChar
printf("%c", currChar);
emailSectionPrinted++;
}
if(emailSectionPrinted == 2){
//we print @, when the last two chars of the birthyear have  //been
scanned
printf("@");
}
}
else{
// the college name
if('A' <= currChar && currChar <= 'Z'){
currChar = ((currChar - 'A') + 'a');
}
printf("%c", currChar);
}
}while(1);
}
//at the end we need to print .ac.in followed by a newline
printf(".ac.in");
if(i < n-1){
```

```
printf("\n");

}

}

return 0;

}
```

# Compare Strings

You are given two words. You have to compute the following functions for each word, and based on the output you have to tell which is larger. The function is defined as follows. f(c), where c is a character f(a) = 0, f(b) = 1, .... f(z) = 25. f(S), where s is a word = sum over all c in S(f(c)). For example f(abc) = f(a) + f(b) + f(c) = 0 + 1 + 2 = 3.

## *Constraint:*

The word will only consist of lowercase letters.

## *Input:*

- The input will span accross 4 lines:
- n1
- word of length n1
- n2
- word of length n2

## *Output:*

- The program should print:
  - `first`, if the function for the first word is larger
  - `second`, if the function for the second word is larger
  - `tie`, if the function for both is same

There should be any spaces or newline chars in the outputs.

## *Sample TestCase:*

- TestCase1:

Input

```
1
a
2
aa
```

Output:

```
tie
```

- TestCase2:

Input

- 1
- a
- 1
- b

Output:

```
second
```

- Explanation:

TEST CASE1: The function for both the words is 0, hence it is a tie.

TEST CASE2: The function for the first is 0 and for the second is 1, hence it is second

```
#include<stdio.h>

int main()

{

 char s1[10],s2[10];

 int i,j,n,m,sum1=0,sum2=0;

 scanf("%d\n",&n);

 for(i=0;i<n;i++)

 {

 scanf("%c",&s1[i]);

 }

 scanf("%d\n",&m);
 for(i=0;i<m;i++)

 {

 scanf("%c",&s2[i]);

 }

 for(i=0;i<n;i++)

 {

 sum1=sum1+s1[i]-97;
```

```
}
for(i=0;i<m;i++)
{
sum2=sum2+s2[i]-97;
}
if(sum1==sum2)
printf("tie");
else if(sum1>sum2)
printf("first");
else if(sum2>sum1)
printf("second");
return 0;
}
```

# Practice 5

# Candy Crush

Your friendly neighborhood shopkeeper has decided to reward your excellent performance at JEE and give you some candies. She gives you a list of 10 integers (not all may be positive) and tells you to select two locations in the list (i.e. two numbers between 1 and 10 -- the first number being smaller than or equal to the second number).

Once you tell her these two locations, the shopkeeper will to give as many candies as the sum of all the numbers occurring between the locations indicated by you (including the locations specified by you). Write an algorithm to calculate what is the maximum number of candies you can get this way. Your input will be a list of 10 integers and your output should be the maximum number of candies you can get, the starting location, and the ending location, all separated by a single space with no trailing spaces at the end.

**Caution**

- Be careful about extra/missing lines and extra/missing spaces.
- Be careful that array locations start from zero whereas in your output, you should give locations that start from one, just as we do in day to day conversations.

**Constraints**

All numbers lie between -1000 and 1000.

**Sample Input**

1 3 5 2 6 -7 1 1 -3 1

**Sample Output**:

17 1 5

*Explanation* : If you select the locations 1 and 5, then you get will get 1+3+5+2+6 = 17 candies since the numbers between the first and fifth locations (including both ends) in the list are 1, 3, 5, 2, 6. You can verify that this is the maximum number of candies you can get.

| | | | | |
|---|---|---|---|---|
| 1 | 1 2 3 4 5 6 7 8 9 10 | 55 1 10 | 55 1 10 | |
| 2 | 18 -560 -187 617 -592 476 730 188 105 - 635 | 1524 4 9 | 1524 4 9 | |
| 3 | 484 -44 -812 596 821 -488 -530 -253 -709 141 | 1417 4 5 | 1417 4 5 | |
| 4 | 78 15 97 74 84 84 -77 -9 55 48 | 449 1 10 | 449 1 10 | |

```
#include <stdio.h>


int main(){
 int n = 10, i, j, k, arr[10];
 for(i = 0; i < n; i++)
 scanf("%d", &arr[i]);
 int sum = 0, max = -1001;
 int start, end;
 for(i = 0; i < n; i++){
 for(j = i; j < n; j++){
 sum = 0;
 for(k = i; k <= j; k++)
sum += arr[k];
 // Can get rid of this third inner loop
 if(sum > max){
 max = sum;
 start = i;
 end = j;
```

```
}

}

}

printf("%d %d %d", max, start+1, end+1);

return 0;

}
```

# OddExists

Write a program that checks if there exists an odd integer in an array of integers.

**Input Format**:
The first line of input contains the size `n` of the array. Then `n` lines of input follow giving the contents of the array.

**Output Format**:
You must output `Yes` or `No` if there exits an element with the desired property or not respectively in the given array

**Variable Constraints**: The size of the array is smaller than `20`. The contents of the array are integers in the range from `-32768` to `32767`.

**Sample Input 1**:
```
5
2 3 4 6 102
```

**Sample Output 1**:
```
Yes
```

**Sample Input 2**:
```
3
74 18 64
```

**Sample Output 2**:
```
No
```

```
#include<stdio.h>

void scan_array(int* array , int no_of_elements ) // Scanning the array {

        int i;
        for( i = 0; i < no_of_elements ; i++)

        {

                scanf("%d",&array[i]);

        }

}

int main()

{
```

```
int no_of_elements;
int array[20]; //Define array
int i; //iterators
scanf("%d",&no_of_elements);
scan_array(array , no_of_elements); //Function call to scan the array int
flag = 0;
//Iterating over all the elements of the array
for( i = 0; i<no_of_elements ; i++)
{
        if(array[i]%2 != 0 )
        {
                flag = 1;//flg variable is made 1 if the condition is true.
        }
}
if(flag == 1) //checking existence of the desired element. {
        printf("Yes\n");
}
else
{
        printf("No\n");
}
return 0;
}
```

# Lucky Draw

You, along with your two best friends, go to a party, where a tile of 7 coupons is kept, one on top of the other. Each coupon has a **unique** non-negative integer number, which denotes how many gifts does someone win at the end of the party. You know that you and your friends will be the first to select the coupons, so you decide to cheat and keep the coupons with maximum, second maximum and third maximum at the top three places on the pile.

You will be given a list of 7 non-negative integers separated by a space as input. Write a code to find the position of the maximum element in the list, and swap it with the first element of the list. Then find the position of the second largest element of the list and swap it with the second element in the list. Finally, find the position of the third largest element in the list and swap it with the third element of the list. Output the resulting list with one space between two numbers. There

should be no trailing spaces in your output.

**Note:**
Be careful about extra/missing lines and extra/missing spaces.
Be very careful, even though the evaluation may give you marks for extra spaces and newlines, the autograder will give you zero marks for any extra spaces or new lines.

**Hint:**
You will require the use of arrays in this question.

**Examples:**

*Input 1*
```
1 2 5 4 6 7 8
```

*Output 1*
```
8 7 6 4 5 2 1
```

*Explanation 1*
The maximum element is 8, which gets swapped with the first element 1. The second maximum element is 7, which gets swapped with the second element 2. The third largest element is 6, which gets swapped with the third element 5.

*Input 2*
```
6 5 4 3 2 1 7
```

*Output 2*
```
7 6 5 3 2 1 4
```

*Explanation 2*
First we swap 6 (first element) and 7 (largest element) and we get `7 5 4 3 2 1 6` and then we swap 5 (second element) and 6 (second largest element) and we get `7 6 4 3 2 1 5` and finally we swap 4 (third element) and 5 (third largest element) and we get `7 6 5 3 2 1 4`

```
#include <stdio.h>

int main()

{

 int n = 7;

 int ar[n];

 for(int i=0;i<n;i++)

 scanf("%d", &ar[i]);

 int max1, pos;
 for(int i=0;i<3;i++){

 max1 = ar[i];

 pos = i;

 for(int j=i;j<n;j++){

 if(max1<ar[j]){

 pos = j;
```

```
max1 = ar[j];
}
}
int temp = ar[pos];
ar[pos] = ar[i];
ar[i] = temp;
}
for(int i=0;i<n;i++){
printf("%d", ar[i]);
if(i < n-1) printf(" ");
}
return 0;
}
```

# Sum Divisibility

You are given an array of size of `n`. All the elements of the array are non-negative integers. Check if the sum of all the elements present at even indices of the array is even.

**Input**: You will be given a positive integer `n`, followed by `n` non-negative integers. **Output**: Print `YES` if the required sum is even and `NO` otherwise.

**Examples**:

*Input 1*

```
4 1 2 3 4
```

*Output 1*

```
YES
```

*Explanation 1*

`1 + 3 = 4` which is even.

*Input 2*

```
9 1 2 3 4 5 6 7 8 9
```

*Output 2*

```
NO
```

*Explanation 2*

`1 + 3 + 5 + 7 + 9 = 25` which is odd.

```
#include<stdio.h>
int main(){
        int n;
```

```
        int arr[1000];
        scanf("%d", &n);
        int i;
        int sum = 0;
        for(i=0; i<n; i++){
                scanf("%d", &arr[i]);
                if(i%2 == 0){
                        sum = sum + arr[i];
                }
        }
        if(sum % 2 == 0){
                printf("YES");
        }
        else{
                printf("NO");
        }
        return 0;
}
```

# Prime Queries

Your friend, Shyam recieved a problem from his teacher. The problem goes like this You will be provided an array of non-negative integers. Next, you will be asked some queries. There will be two types of queries - type 'A' and type 'B'.

For query of type 'A', you will be provided with two valid indices of array, you need to report how many numbers between these indices in the array (including these numbers) are prime. For query of type 'B', you will be provided with two valid indices of array, you need to report how many numbers between these indices in the array (including these numbers) are not prime.

**Input Format**

- The first line contins a integer equal to n, i.e the number of elements. •
  Next line contains 'n' space separated non-negative integers.
- Next line contians q, which represent the number of queries
- Each of the next 'q' lines contains one character c, and two integers i and j all separated by integers.

**Constraints**

- 1<= n <= 1000
- 1<= q <=500
- c is either 'A' or 'B'
- i <=j < n

**Output Format**

- Print the answer of the above queries. Answer each query in a separate line.

**Sample Input**

```
5
5 3 1 2 4
3
A 0 4
A 0 2
B 2 4
```

**Sample Output**

```
3
2
2
```

```
#include<stdio.h>


int isPrime(int n){
        if(n<=1)
                return 0;
        int i=2;
        while(i*i<=n){
                if(n%i==0)
                        return 0;
                i++;
        }
        return 1;
}


int primeCount(int s[], int i, int j){
 int count = 0;
 int index,r;
```

```
for(index = i; index<=j; index++){
if(isPrime(s[index]))
                                count++;
}
return count;
}
int main(){
int n,q,array[1000],i,j,index;
char c;
scanf("%d",&n);
for(index = 0; index < n; index++){
scanf("%d",&array[index]);
}
scanf("%d\n",&q);
for(index = 0; index < q; index++){
scanf("%c %d %d",&c, &i,&j);
if(c=='A'){
printf("%d\n", primeCount(array,i,j));  }
else{
printf("%d\n", j-i+1 - primeCount(array,i,j));  }
if(i!=q-1) scanf("\n");
}
return 0;
}
```

# Drone Control

A drone is floating in space. Its initial coordinates are `(X,Y,Z)`. You send a series of `C` movement commands to the drone.There are `6` movement commands `1`, `2`, `3`, `4`, `5` and `6`.

1--UP

For Example, if the drone is at `(0,0,0)`, then after the command it will be at `(0,0,1)` 2--Down

For Example, if the drone is at `(0,0,0)`, then after the command it will be at

`(0,0,-1)` 3--North

For Example, if the drone is at `(0,0,0)`, then after the command it will be at

`(1,0,0)` 4--South

For Example, if the drone is at `(0,0,0)`, then after the command it will be at

`(-1,0,0)` 5--East

For Example, if the drone is at `(0,0,0)`, then after the command it will be at

`(0,1,0)` 6--West

For Example, if the drone is at `(0,0,0)`, then after the command it will be at `(0,-1,0)`

Time Elapsed=0, just before the first command is issued.
Consider that each command takes `1` time unit to execute.
For Example, At Timestamp `T=5` if the drone is at `(1,5,3)` and we issue the command 1(UP), then at Timestamp `T=6` it will be at `(1,5,4)`.

Given `Q` queries where each query is a timestamp `t`, you will have to output the coordinate of the drone at each of the timestamps.

To sum up: `(X,Y,Z)` is the initial position of the drone. `C` is the number of commands issued `Q` is the number of queries

**Input Format:**
The first line contains `5` space separated integers `X`, `Y`, `Z`, `C`, `Q`.
The second line contains `C` space separated integers describing the commands. Each of these `C` commands can be `1`, `2`, `3`, `4`, `5` or `6`.
The third line contains `Q` space separated integers specifying the queries. Each of these `Q` queries specifies a timestamp `t` at which the coordinate of the drone is required. `0 ? t ? C`.

**Output:**
Print `Q` lines corresponding to each of the `Q` queries.
In each line print `3` space separated integers `a,b,c` . Where `(a,b,c)` is the coordinate of the drone at the given timestamp.

**Constraints:**
`1 <= N <= 200 1 <= Q <= 200`

**Example Input:**
```
0 0 0 3 3
1 1 4
0 3 2
```

**Example Output:**
```
0 0 0
-1 0 2
0 0 2
```

**Explanation:**
At Timestamp T=0, position of Drone: (0,0,0)
At Timestamp T=1, position of Drone: (0,0,1)
At Timestamp T=2, position of Drone: (0,0,2)
At Timestamp T=3, position of Drone: (-1,0,2)

| | | | | |
|---|---|---|---|---|
| 1 | 0 0 0 3 3 1 1 4 0 3 2 | 0 0 0 -1 0 2 0 0 2 | 0 0 0 -1 0 2 0 0 2 | |
| 2 | 1 1 1 4 1 2 4 6 2 3 | 0 0 0 | 0 0 0 | |
| 3 | 2 0 -1 6 3 1 2 3 4 5 6 2 4 6 | 2 0 -1 2 0 -1 2 0 -1 | 2 0 -1 2 0 -1 2 0 -1 | |
| 4 | -2 -1 0 1 2 5 0 1 | -2 -1 0 -2 0 0 | -2 -1 0 -2 0 0 | |
| 5 | 8 -1 1 8 3 1 3 6 2 2 2 5 4 0 5 7 | 8 -1 1 9 -2 0 9 -1 -1 | 8 -1 1 9 -2 0 9 -1 -1 | |

```
#include <stdio.h>
int main()
{
 int X,Y,Z,C,Q;
 scanf("%d%d%d%d%d",&X,&Y,&Z,&C,&Q);
 int commands[200];
 int queries[200];
 int pos_x[201];
 int pos_y[201];
 int pos_z[201];
 for(int i=0;i<C;i++)
 {
 scanf("%d",&commands[i]);
 }
 for(int i=0;i<Q;i++)
 {
 scanf("%d",&queries[i]);
 }
 pos_x[0]=X;
```

```
pos_y[0]=Y;
pos_z[0]=Z;
for(int i=0;i<C;i++)
{
if(commands[i]==1)  {
pos_z[i+1]=pos_z[i]+1;
pos_x[i+1]=pos_x[i];
pos_y[i+1]=pos_y[i];  }
else if(commands[i]==2)  {
pos_z[i+1]=pos_z[i]-1;
pos_x[i+1]=pos_x[i];
pos_y[i+1]=pos_y[i];  }
else if(commands[i]==3)  {
pos_x[i+1]=pos_x[i]+1;
pos_y[i+1]=pos_y[i];
pos_z[i+1]=pos_z[i];  }
else if(commands[i]==4)  {
pos_x[i+1]=pos_x[i]-1;
pos_y[i+1]=pos_y[i];
pos_z[i+1]=pos_z[i];  }
else if(commands[i]==5)
{
pos_y[i+1]=pos_y[i]+1;
pos_x[i+1]=pos_x[i];
pos_z[i+1]=pos_z[i];
}
else if(commands[i]==6)
{
pos_y[i+1]=pos_y[i]-1;
pos_x[i+1]=pos_x[i];
```

```
pos_z[i+1]=pos_z[i];

}

else

{

printf("Invalid Input");

}


}


for(int i=0;i<Q;i++)

{

printf("%d %d %d\n",pos_x[queries[i]],pos_y[queries[i]],pos_z[queries[i]]);  }

return 0;

}
```

# Peaks

Given an array $A_1$, $A_2$,... $A_n$ of `n` integers ( `2 < n < 100` ).
Find total number of peaks in the array. An element $A_i$ (`i > 1` and `i < n`) is a peak iff $A_i > A_{i-1}$ and $A_i > A_{i+1}$.

**Input Format:**
The first line of input contains the size n of the array.
The second line contains n space-separated integers denoting elements of the array.

**Output Format:**
You must output an integer which is the answer to the question.

*Sample Input 1:*

```
3
1 3 2
```

*Sample Output 1:*

```
1
```

*Explanation:*
$A_2 = 3$ is the only peak here.

*Sample Input 2:*

```
4
1 2 3 4
```

*Sample Output 2:*

```
0
```

*Explanation:*
There is no peak element in the given array.

```
#include <stdio.h>

int main()

{

 int a[200];

 int n;

 scanf("%d", &n);

 for(int i=0;i<n;i++)

 scanf("%d", &a[i]);

 int tot = 0;

 for(int i = 1; i < n-1; i++){

 if(a[i] > a[i-1] && a[i] > a[i+1]) tot++;

 }

 printf("%d", tot);

 return 0;

}
```

# Strongest Team

Alice is given a list of `n` players and their skill level. It is guaranteed that the skill level of each player is some non-negative number.
Now he wants to make a team of `k` players (`k <= n`) to participate in a tournament. But since he is lazy, he wants to select only adjacent players. He also wants to make the team which has maximum value so that the chance of winning is maximised. The value of a team is defined as the sum of skill level of each player in that team.

**Input Format:**
The input consists of three lines.
The first line contains the number of players, n.
The second line contains n space-separated non-negative integers denoting the skill level of each player.
The third line contains integer `k`.

**Output Format:**
Output an integer denoting the value of Alice's team.

*Sample Input:*

```
5
12 3 5 11 15
3
```

*Sample Output:*

```
31
```

*Explanation:*

All the possible teams that Alice can make are `{1, 2, 3}`, `{2, 3, 4}` and `{3, 4, 5}`.
The corresponding values of teams are `20`, `19` and `31` respectively.
So the value of Alice's team is 31.

| | | | | |
|---|---|---|---|---|
| 2 | 7 9 7 12 9 6 5 1 4 | 37 | 37 | |
| 3 | 10 14 8 0 12 1 12 3 4 6 7 4 | 34 | 34 | |
| 4 | 5 4 11 2 4 0 5 | 21 | 21 | |

```
#include <stdio.h>

int main()

{

 int a[200];

 int n;

 scanf("%d", &n);

 for(int i=0;i<n;i++)

 scanf("%d", &a[i]);

 int k;

 scanf("%d", &k);

 int mx = 0;

 for(int i = 0; i < n-k+1; i++){

 int curr = 0;

 for(int j = 0; j < k; j++){

 curr += a[j+i];
 }

 if(mx < curr)

 mx = curr;

 }

 printf("%d", mx);

 return 0;

}
```

Practice 6

# Vowel - Consonant Count

Given a string, you are required to calculate the number of vowels and consonants in the string. It is mandated that you use pointers and dynamic memory allocation to store the string (in the heap memory) and use a function of the following prototype for doing the computation.

`int* count_vowel_consonant(char* s, int n)`
The function takes a character pointer and an integer as arguments and returns an integer pointer with information on both vowel and consonant counts.

Do not change the given function prototype. You may write other helper functions if necessary.

You **should not** use statically allocated arrays for this question. You should allocate and free space using dynamic memory allocation and use only pointers referencing and dereferencing to access the string. Do not use array subscripting in your program. Using string library functions is also prohibited.

**Input Format**:
The first line is an integer `n` that denotes the length of the string.
The second line contains a string `str` consisting of `n` non-null characters.

**Output Format**:
The count of vowels in `str` followed by the count of consonants in `str` separated by a space. Example

**Input**:

```
4
Ck3A
```

**Output**:

```
1 2
```

## Public Test Cases

**Input Output**
**Input Output**

```
4
Ck3A 1 2
```

| 2 | 1 S |
|---|---|

| | |
|---|---|
| 3 | 1 : |
| 4 | 8 'NU"}xXo |
| 5 | 23 Tv*Y2E;LdWK2n7G&+.h6WTg |
| | |

0 1

0 0

2 3

1 13

```
#include<stdio.h>
#include<stdlib.h>
int* count_vowel_consonant(char* str, int n) {
 int i;
 int v = 0, c_=0;
 int* ans = (int*)malloc(2 * sizeof(int));
 char c;
 for(i = 0; i < n; i++) {
 c = *(str + i) ;
 if(((c >= 97) && (c <= 122)) || ((c >= 65) && (c <= 90))) {
 if((c == 'a') || (c == 'e') || (c == 'i') || (c == 'o') || (c == 'u'))
 v++;
 else if((c == 'A') || (c == 'E') || (c == 'I') || (c == 'O') || (c == 'U'))
 v++;
```

```
else

c_++;

}

}

*(ans) = v;
*(ans + 1) = c_;

return ans;

}


int main()

{

char* str;

int n;

scanf("%d\n", &n);

str = (char*)malloc((n + 1) * sizeof(char));

for(int i = 0; i < n; i++) {

char c;

scanf("%c", &c);

*(str + i) = c;

}

*(str + n) = '\0';

int* a = count_vowel_consonant(str, n);

printf("%d %d", *(a) , *(a + 1));

return 0;

}
```

# Extract Characters from Number

We know that an int variable occupies 4 bytes of space whereas a char variable occupies 1 byte. Therefore, one integer variable has the capacity to store four chracter values in it. Given an integer value `n`, extract and print the 4 characters in it. The extraction is done as follows. The binary representation of `n` has 32 bits. These are divided into 4 segments of 8 bits (1 byte) each and each byte is treated as the ASCII value of the character it represents. These characters are to be printed.

*Note*: It is guaranteed that the number `n` is given in such a way that the characters extracted are always alphabets or digits.

**Input Format**:
A positive integer `n`.

**Output Format**:
The 4 characters extracted from the binary representation of `n` with no separators.

# Example

**Input**:
`1094861636`

**Output**:
`ABCD`

**Explanation**:
Binary representation of `1094861636` is `01000001010000100100001101000100`. Separating it as 4 segments of 8 bits each, we get `01000001 = 65`, `01000010 = 66`, `01000011 = 67`, `01000100 = 68`. These values represent `A`, `B`, `C` and `D` respectively by the ASCII mapping.

# Public Test Cases

**Input Output**

`1094861636 ABCD`

| | | | | |
|---|---|---|---|---|
| 1 | 1094861636 | ABCD | ABCD | |
| 2 | 1751477356 | hell | hell | |
| 3 | 1466774376 | Wm3h | Wm3h | |
| 4 | 1400338278 | Swwf | Swwf | |
| 5 | 1144485995 | D7xk | D7xk | |
| 6 | 1867135330 | oJ9b | oJ9b | |
| 7 | 829257269 | 1mv5 | 1mv5 | |

```
#include <stdio.h>

#include <stdlib.h>
```

```
int main() {

 int* a = (int*)malloc(sizeof(int));

 char* b = (char*)a + 3;

 scanf("%d", a);

 for(int i = 0; i < 4; i++) {

 printf("%c", *b);
 b -= 1;

 }

 return 0;

}
```

# Even-Even

You are given an array of integers. Let us call it row A .You need to write a program to verify whether A is an even row. A row is said to be even if it has even number of even numbers. 0 is an even number and extend the definition of even numbers to negative integers also.

**Input Format**:

The first line of input contains the size `n` of the array. The next line of input contains the contents of the array.

**Output Format**:

You must output "Yes" or "No" in a separate line for each test case. Output "Yes" if the corresponding test case holds the required property else print "No".

**Note**:

You **should not** use statically allocated arrays for this question. You should allocate and free space using dynamic memory allocation and use only pointers referencing and dereferencing to access the array. Do not use array subscripting in your program.
You **should** define a function `void scan_array(int *arr, int n)` for scanning the array.
You **should** define a function `int cnt(int *arr, int n)` for finding the answer.

# Example

**Input**:
4
1 2 3 4

**Output**:
Yes

| | | | | |
|---|---|---|---|---|
| 2 | 6 11 0 0 -1 8 9 | No | No | |

| | | | | |
|---|---|---|---|---|
| 3 | 8 112 110 88 -99 86 -1 51 69 | Yes | Yes | |
| 4 | 1 6999 | Yes | Yes | |
| 5 | 5 -544 0 2 7 311 | No | No | |

```
#include<stdio.h>
#include<stdlib.h>
int cnt( int* arr , int n)
{
	int i, count;
	count = 0 ; //Initialising count
	for(i = 0; i < n; i++)
		{
		 int x = *(arr + i);
			if(x%2 == 0)
			{
				count++; // Increasing the count for each even number found in the
array.
			}
		}
	return ((count%2)==0); // Checking if count is even.
}
void scan_array(int* arr , int n) // Scanning the array
{
	int i;
	for(i = 0; i < n; i++)
	{
		scanf("%d", arr+i);
	}
}
```

```
int main()

{

        int n;

scanf("%d", &n);

        int *arr = (int *)malloc(n*sizeof(int)); //Define array

        int i; //iterators

        scan_array(arr, n); //Function call to scan the array

        if(cnt(arr, n))

        {
                printf("Yes\n");

        }

        else

        {

                printf("No\n");

        }

        return 0; }
```

# Brick Sort

There are several sorting algorithms that have been developed over the years. We saw some during the lectures and will explore others here. One simple algorithm is called Brick Sort. The algorithm got its name since it sorts alternating elements of the array in subsequent phases, so that the whole arrangement (see figure below) looks like bricks have been laid down. This variant was originally designed for parallel processing and may work very well on parallel computing architectures such as GPUs.

Brick sort proceeds in n phases. In the first phase, even indices of the array get compared to the element to their immediate right and if the pair is out of order (i.e. the element at the even index is strictly greater than the element to its immediate right), then the pair is swapped. In the second phase, odd indices of the array get compared (and swapped if out of order) to the element to their immediate right. The third phase again considers the even indices and so on. If the array has n elements, then the algorithm works for n phases. The array is guaranteed to be sorted at the end of the n-th phase.

The first line of the input will give you n, a strictly positive integer and the second line will give you n integers, separated by a space. Store these numbers in an array of size n. In your output, you have to print the array after each phase of brick sort, on a separate line. Print the array by printing each number in the array from left to right with two numbers separated by a single space. However, there should be no space at the end of a line.

# Note

If the array has an odd number of elements, say 5, then the index 4 element will have no element to its right in the phases when even elements are being compared to the element to their immediate right (for example the first phase). So nothing will need to be done for index 4.

The n numbers we give you may be positive, negative or zero. The same number may occur twice in the list too.
The number of elements n can be any strictly positive number, even 1. Your output must have exactly n lines. There should be no trailing newline character after the last line. Some phases may not require any work. However, you must still print the array after those phases.

# Example Input

```
4
3 4 1 2
```

# Example Output

```
3 4 1 2
3 1 4 2
1 3 2 4
1 2 3 4
```

# Explanation

First phase 3 4 1 2 (Checking for swap between 3 & 4 - nothing to be done) 3 4 1 2 (Checking for swap between 1 & 2 - nothing to be done) No other even index elements left.

Second phase 3 1 4 2 (Checking for swap between 4 & 1 - swap them) No other odd index elements left.

Third phase 1 3 4 2 (Checking for swap between 3 & 1 - swap them) 1 3 2 4 (Checking for swap between 4 & 2 - swap them) No other even index elements left.

Fourth phase 1 2 3 4 (Checking for swap between 3 & 2 - swap them) No other odd index elements left.

```
#include <stdio.h>

#include <stdlib.h>

void swap(int *a, int *b) {

 int t = *a;

 *a = *b;

 *b = t;

}
```

```
void prin(int *a, int n, int i) {
 for(int i = 0; i < n-1; i++) {
 printf("%d ",a[i]);
 }
 printf("%d",a[n-1]);
 if (i != n-1) {
 printf("\n");
 }
}
void snakeSort(int *a, int n) {
 for(int i = 0; i < n; i++) {
 int j = i%2;
 for(; j < n-1; j+=2) {
 if (a[j] > a[j+1]) {
 swap(a+j, a+j+1);
 }
 }
 prin(a,n,i);
 }
}
int main() {
 int n;
 scanf("%d",&n);
 int *a = (int *) malloc(n*sizeof(int));
 for(int i = 0; i < n; i++) {
 scanf("%d",&a[i]);
 }
 snakeSort(a,n);
}
```

# Bits and Bits

In the input, you will be given a strictly positive integer K denoting the length of the strings you have to print. In your output, on each line, you have to print a string of length K using only the characters '0' and '1' (without quotes). The strings must be printed in lexicographically increasing order i.e. if you think of these strings as numbers, the numbers should appear in increasing order.

The only property these strings must satisfy is that no two consecutive characters in the strings you generate can be the character '0'. Two or more consecutive characters can be '1' but two consecutive characters cannot be '0'.

# Note

The first character in the string can freely be 0 or 1 since there is no previous character to cause consecutive 0. However, second character onwards, we must have a 0 only if the previous character was not 0 to avoid consecutive 0. Be sure to print all leading 0 in the output. Every string you print must contain k characters.

There should be no extra spaces anywhere. There **MUST** be a trailing newline after the last string printed.

# Example Input

```
2
```

# Example Output

```
01
10
11
```

# Explanation

`00` is an illegal string.

```
#include<stdio.h>

int main(){

        int k;

        scanf("%d", &k);

        int total = 0;

        int curr = 1;

        for(int i = 0; i < k; i++){

                total = total + curr;

                curr = curr*2;
```

```
        }
        for(int i = 0; i <= total; i++){
                int a[k];
                int x = i;
                int flag = 1;
                for(int j = 0; j < k; j++){
                        a[j] = x%2;
                        x = x/2;
                }
                for(int j = k-1; j > 0; j--){
                        if(a[j] == 0 && a[j] == a[j-1])
                                flag = 0;
                }
                if(flag){
                        for(int j = k-1; j >= 0; j--){
                                printf("%d", a[j]);
                        }
                        printf("\n");
                }
        }
        return 0;
}
```

# Replace String

You are given three strings `s1`, `s2`and `s3` as input. You need to replace all the occurrences of `s2` in `s1` by `s3`. Print the string `s1` unchanged if `s2` does not occur in `s1` even once.

**Input Format**:

Three lines of input each of them are of the form "n s" where n is size of the input string s.

**Output Format**:

The new string `s1` after replacement.

# Example

**Input**:
18 abcxyzdefxyzghixyz
3 xyz
2 uv

**Output**:
abcuvdefuvghiuv

**Explanation**: `s2` occurs in `s1` at index 3, 9, 15. We replace the respective `xyz` by `uv`.

# Public Test Cases

| Input | Output |
| --- | --- |
| `18 abcxyzdefxyzghixyz`<br>`3 xyz`<br>`2 uv` | `abcuvdefuvghiuv` |

```
int get_len(char s[])
#include <stdio.h>
#include <stdlib.h>
{
        int i = 0 ;
        while(s[i] != '\0')
                i++;
        return i ;
}


  int number_occourences(char *s, char *t){ int s_len, t_len, i,j = 0;
        int ncount, count = 0 ;
        s_len = get_len(s);
        t_len = get_len(t);
        for(i = 0 ; i < s_len ; i++){
                ncount = 0 ;
                for(j = 0 ; i + j < s_len && j < t_len; j++)
                        if(s[i+j] == t[j])
                                ncount++;
```

```
                count = count + ncount/t_len;
        }

        return count;
}

int repace_substring(char *s,char *t,char
*w) {
        int j,i=0, k=0, index=-1,ncount=0;
        char *copy,*x,*y,*z;
        int s_len = get_len(s);
        int t_len = get_len(t);
        int w_len = get_len(w);
        int max_size_new = s_len + w_len*(number_occourences(s,t) +
        1); copy = (char *)malloc(max_size_new * sizeof(char));
        for(i = 0 ; i < s_len ; i++){
                ncount = 0 ;
                //checking if we have an occourence of a substring
                for(j = 0 ; i + j < s_len && j < t_len; j++)
                        if(s[i+j] == t[j])
                                ncount++;
                // we have found an occourence
                if (ncount/t_len) {
                        for(j = 0 ;j < w_len; j++)
                                copy[k + j] = w[j];
                        i = i + t_len - 1;
                        k = k + w_len;
                }
                else
```

```
                {
                        copy[k] = s[i];
                        k++ ;
                }
        }
        printf("%s\n",copy);
}
int main()
{
        int n1, n2, n3;
        char *s, *t, *w;
        scanf("%d" , &n1);
        s = (char *)malloc((n1 + 1) * sizeof(char));
        scanf("%s",s);
        scanf("%d" , &n2);
        t = (char *)malloc((n2 + 1) * sizeof(char));
        scanf("%s",t);
        scanf("%d" , &n3);
        w = (char *)malloc((n3 + 1) * sizeof(char));
        scanf("%s",w);
        replace_substring(s,t,w);
        return 0;
}
```

# Incremental Modification

Given an array A of N integers and M increment operations, you must print the final state of the array after all operations.

An increment operation takes two arguments - a pointer to an array element and an integer value and it increments the array element by this value. This operation must be implemented using a void function, whose template is given. Do not modify the arguments of the template, you are supposed to fill in the code for the function body only.

```
#include


void Increment( int *p1, int delta){
 // Insert your code here. DO NOT modify the arguments of the function.


}


int main(){
 // Insert your code here.
 return 0;
}
```

Given an index i and a value delta, you must increment A[i] by d by passing the pointer of this array element to the void function.

Input Format: First line contains N and M, where N is the size of integer array A and M is the number of increment operations. Second line contains N integers of the array A. Each of the next M lines contain an index i and an integer value delta corresponding to the increment of A[i] by value delta.

Constraints: 1 < N < 10000 1 < M < 100

Output Format: You must print N comma separated integers corresponding to the final array A after all M increment operations.

# Public Test Cases

| Input | Output |
|---|---|
| 5 2<br>9 5 6 0 3<br>0 -4<br>2 2 | 5,5,8,0,3 |

```
#include <stdio.h>

void Increment( int *p1, int delta){

 *p1 = *p1 + delta;

}

int main(){

        int A[10009],N,M;

        scanf("%d%d",&N,&M);
```

```
        int i;
        for(i=0; i<N; i++) scanf("%d",A+i);
int index,delta;
while(M--){
scanf("%d%d",&index,&delta);
Increment(A+index,delta);
}
printf("%d",A[0]);
        for(i=1; i<N; i++) printf(",%d",A[i]);
return 0;
}
```

# Make Equal

You are given two strings, find out minimum number of character deletions required to make two strings equal. You can delete any character from any string. Two strings are said to equal if every character of one string is also present in other string at any place. Example : "abc" and "cab" are equal strings.

Note : String contains lower case alphabets only. Size of string will be given by the user and use dynamic memory allocation.

Input Format : First two lines contain the size of the first string and the string itself. Next two lines contain the size of the second string and the string itself.

Output Format : Print minimum number of character deletions required to make the strings equal.

## Public Test Cases

**Input Output**

| | |
|---|---|
| 3<br>cde 3 4<br>abc | |

```
#include <stdio.h>
#include <stdlib.h>
int makeEqual(char * s1, int n1, char * s2, int n2){
int a[26] = {0}, b[26] = {0}, i, r = 0;
 for(i = 0; i < n1; i++)
 *(a + *(s1 + i) - 'a') += 1;
```

```
 for(i = 0; i < n2; i++)
 *(b + *(s2 + i) - 'a') += 1;
 for(i = 0; i < 26; i++)
 r = r + abs(*(a + i) - *(b + i));
 return r;
}
int main(){
 int n1, n2;
 char *s1, *s2;
 scanf("%d\n", &n1);
 s1 = (char *)malloc(n1 * sizeof(char));
for(int i = 0; i < n1; i++)
 scanf("%c", (s1 + i));
 scanf("\n%d\n", &n2);
 s2 = (char *)malloc(n2 * sizeof(char));
for(int i = 0; i < n2; i++)
 scanf("%c", (s2 + i));
 printf("%d", makeEqual(s1, n1, s2, n2));
return 0;
}
```

# Practice 7

# Binary-Poem

A Noob-Robot is asking from you to write a program to print a binary poem which consists of only 1's and 0's. You're given a number `N`. And, your task is print all string of length `2*N` which consists of `N` 1's and `N` 0's in lexico-graphical order.

A string S is lexicographically smaller than another string T if S comes before T in the dictionary. For e.g if S = "1001" and T = "1010" then S is lexicographically smaller than T. if S = "10" and T = "01" then T is lexicographically smaller than S.

## Input Format

A single line containing an integer `N`.

## Output Format

Print all strings of length `2*N` which consist of `N` 1's and `N` 0's in lexico-graphical order. Example Input

3

## Example Output

```
000111
001011
001101
001110
010011
010101
010110
011001
011010
011100
100011
100101
100110
101001
101010
101100
110001
110010
110100
111000
```

```
#include <stdio.h>
void write_poem(int n, int a, int b, char poem[]) {
 if (a + b == 2 * n) {
 printf("%s\n", poem);
 } else {
```

```
if (a < n) {
poem[a + b] = '0';
write_poem(n, a + 1, b, poem);
}
if (b < n) {
poem[a + b] = '1';
write_poem(n, a, b + 1, poem);
}
}
}
int main() {
int n;
scanf("%d", &n);
char poem[2*n + 1];
poem[2*n] = '\0';
write_poem(n, 0, 0, poem);
return 0;
}
```

# Balanced Parentheses

You are given a number `N`. And, your task is print all string of length `2*N` which consist of `N` opening and `N` closing parentheses in lexico-graphical order.

## Input Format

A single line containing an integer `N`.

## Output Format

Print all strings of length `2*N` which consist of `N` opening and `N` closing parentheses in lexico graphical order.

## Example Input

3

## Example Output

```
((()))
(()())
(())()
()(())
()()()
```

```
#include <stdio.h>
void BalancedParenthesis(int n, int l, int r, int len, char str[]){
 if (len == 2 * n) {
 printf("%s\n", str);
 return ;
 }
 if (l < n) {
 str[len] = '(';
 BalancedParenthesis(n, l + 1,r, len+1, str);
 }
 if (l >r) {
 str[len] = ')';
 BalancedParenthesis(n, l ,r+1, len+1, str);
 }
}
int main(){
 int n;
scanf("%d", &n);
 char str[2*n + 1];
 str[2*n] = '\0';
 BalancedParenthesis(n, 0, 0,0, str);
 return 0;
}
```

# Scoring Rush

You are playing a game where you are given n tiles. Each tile has an associated cost C, and a score S. You have K amount of money with you. You need to pick up some tiles. Your total score for the game is the sum of scores of the tiles you have picked. The associated cost of your selection is the sum of costs of the tiles you have picked. You need to pick a set of tiles (without replacement) such that you maximize your score, subject to the constraint that the total cost of all tiles you have picked does not exceed K. Report the maximum score you can obtain.

*Note that you may select none of the tiles.

# Input Format

The first line contains a single integer `n`, which is the number of tiles.
The next line contains `n` positive integers, each denoting the cost of a tile.
The next line contains `n` non-negative integers, each denoting the associated score from each tile.
The next line contains a single non-negative integer `K`, denoting the amount of money you have.

# Output Format

A single integer, denoting the maximum score you can obtain.

# Sample Input

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

# Sample Output

```
3
```

# Explanation

We can see that with 2 units of money, we can either get the third or fourth tile. It is optimal to get the fourth tile, giving a score of `3`.

```
#include <stdio.h>

int solve(int* cost, int* score, int curr, int n, int money_left)

{

 if(curr == n) return 0;
```

```
 int ans = 0;
 if(money_left >= cost[curr])
 {
 int taken = solve(cost, score, curr + 1, n, money_left - cost[curr]) + score[curr];
if(ans < taken) ans = taken;
 }
 int not_taken = solve(cost, score, curr + 1, n, money_left);
 if(ans < not_taken) ans = not_taken;
 return ans;
}
int main()
{
 int n;
 scanf("%d", &n);
 int cost[n], score[n];
 for(int i=0;i<n;i++) scanf("%d", &cost[i]);
 for(int i=0;i<n;i++) scanf("%d", &score[i]);
 int k;
 scanf("%d", &k);
 int ans = solve(cost, score, 0, n, k);
 printf("%d", ans);
 return 0;
}
```

# That's some challenge

Your genius friend Kolmogorov always makes you feel inferior. This time you have a challenge for him. You give him a positive integer `n`, and ask him to find the number of different ways there are to form the given number `n` as a sum of positive integers in non-decreasing order. Now to pose this challenge to him, you need to solve it yourself :)

## Input Format

A single positive integer `n`

## Output Format

A single integer, denoting the number of ways

## Sample Input

```
3
```

## Sample Output

```
3
```

## Explanation

There are 3 ways to express `3` in non-decreasing order

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

Note that `2 + 1` is invalid, since it is not non-decreasing.

```
#include <stdio.h>
int solve(int n, int prev)
{
 if(n == 0) return 1;
 int ans = 0;
 for(int i=prev;i<=n;i++)
 {
 ans += solve(n - i, i);
 }
return ans;
}
int main()
{
 int n;
```

```
scanf("%d", &n);

printf("%d", solve(n, 1));

return 0;

}
```

# Money

Given the denominations of currencies available in the new system, find the number of ways an ATM machine can generate notes for an entered amount N.

## Input:

First line contains an integer n denoting the number of different denominations available in the new currency system. Second line contains n space-separated integers denoting the denominations in increasing order. Next line contains t denoting the number of test-cases. Next line contains t integers, each representing the amount entered in the ATM.

## Output:

For each of the t test-cases, in a new line output the number of ways the ATM machine can generate notes for the respective amount entered.

## Constraints:

- 1 <= n <= 10
- 1 <= denominations <= 100
- 1 <= t <= 10
- 1 <= amounts <= 1000

## Sample Input:

```
3

2 3 5

1

10
```

## Sample Output:

```
4
```

## Explanation:

The four possible note combinations which can be delivered by the ATM are as follows:

- 10 = 5 + 5
- 10 = 5 + 3 + 2
- 10 = 3 + 3 + 2 + 2
- 10 = 2 + 2 + 2 + 2 + 2

```
#include <stdio.h>


int ways(int n, int a[], int k)
{
 if(n<0 || k<0) return 0;
 if(n == 0) return 1;
 if(k == 0) return 0;
 return ways(n, a, k-1) + ways(n-a[k-1], a, k);
}
int main()
{
 int m; scanf("%d", &m);
 int coin[m], i;
 for(i=0; i<m; i++)
 scanf("%d", &coin[i]);
 int t;
 scanf("%d", &t);
 while(t--)
 {
 int n; scanf("%d", &n);
 printf("%d", ways(n, coin, m));
 if(t != 0)
 printf("\n");
 }
 return 0;
}
```

# Equal Splits

Given an array of integers, is it possible to divide them into two groups so that the sums of the two groups are the same.

## Input:

First Line of input will be an integer `n`, denoting the number of elements in array. Next line will consist of `n` space separated integers.

## Output:

Print "YES" if it's possible otherwise print "NO".

## Constraints:

- n <= 20

## Sample Input:

```
4
1 2 3 4
```

## Sample Output:

```
YES
```

## Explanation:

- 1 + 4 = 2 + 3 = 5

```
#include <stdio.h>

int n, array[20];

int areSplitsEqual(int len, int sum_split_A, int sum_split_B)

{

        if(len == n)
```