**Lab Exam 07**

A – A

An array is a container object that holds a fixed number of values of a single type. To create an array in C, we can do int arr[n];. Here, arr, is a variable array which holds up to integers. The above array is a static array that has memory allocated at compile time.

In this challenge, create an array of size n , and read the values. Iterate the array calculating the sum of all elements. Print the sum.

While it is true that you can sum the elements as they are read, without first storing them to an array, but you will not get the experience working with an array. Efficiency will be required later.

**Input Format**

The first line contains an integer, n .  
The next line contains space-separated integers.

**Constraints**

1 <= n <= 1000

1 <= arr[i] <= 1000

**Output Format**

Print the sum of the integers in the array.

**Sample Input 0**

6

16 13 7 2 1 12

**Sample Output 0**

51

**Sample Input 1**

7

1 13 15 20 12 13 2

**Sample Output 1**

76

#include <stdio.h>

int main()

{

int n;

scanf("%d", &n);

int a[n];

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

{

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

}

int sum=0;

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

{

sum=sum+a[i];

}

printf("%d", sum);

return 0;

}

B – B

Given an array, of size n, reverse it.

Example: If array, arr=[1,2,3,4,5], after reversing it, the array should be, arr= [5,4,3,2,1] .

**Input Format**

The first line contains an integer, n, denoting the size of the array. The next line contains  n space-separated integers denoting the elements of the array.

**Constraints**

1<=n<=1000

1<=arr[i]<=1000  
, where arr[i]  is the  [i] th element of the array.

**Output Format**

The output is handled by the code given in the editor, which would print the array.

**Sample Input 0**

6

16 13 7 2 1 12

**Sample Output 0**

12 1 2 7 13 16

**Explanation 0**

Given array, arr  = [16,13,7,2,1,12]. After reversing the array, arr = [12,1,2,7,13,16]

**Sample Input 1**

7

1 13 15 20 12 13 2

**Sample Output 1**

2 13 12 20 15 13 1

**Sample Input 2**

8

15 5 16 15 17 11 5 11

**Sample Output 2**

11 5 11 17 15 16 5 15

#include <stdio.h>

int main()

{

int n;

scanf("%d", &n);

int a[n];

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

{

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

}

for(int i=0, j=n-1; i<j; i++, j--)

{

int temp=a[i];

a[i]=a[j];

a[j]=temp;

}

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

{

printf("%d", a[i]);

if(i != n-1)

{

printf(" ");

}

}

return 0;

}

C – C

Given a string, s, consisting of alphabets and digits, find the frequency of each digit in the given string.

**Input Format**

The first line contains a string, num  which is the given number.

**Constraints**

1<=len(num)<=1000  
All the elements of num are made of english alphabets and digits.

**Output Format**

Print ten space-separated integers in a single line denoting the frequency of each digit from  0 to 9.

**Sample Input 0**

a11472o5t6

**Sample Output 0**

0 2 1 0 1 1 1 1 0 0

**Explanation 0**

In the given string:

* 1 occurs two times.
* 2,4,5,6 and 7 occur one time each.
* The remaining digits  0,3,8 and  9 don't occur at all.

**Sample Input 1**

lw4n88j12n1

**Sample Output 1**

0 2 1 0 1 0 0 0 2 0

**Sample Input 2**

1v88886l256338ar0ekk

**Sample Output 2**

1 1 1 2 0 1 2 0 5 0

#include <stdio.h>

#include <string.h>

int main()

{

char s[1001], c[11]={0};

char a[11]="0123456789";

//gets(s);

scanf("%[^\n]", s);

int len=strlen(s);

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

{

for(int j=0; j<10; j++)

{

if(s[i]==a[j])

{

c[j]++;

}

}

}

for(int i=0; i<10; i++)

{

printf("%d", c[i]);

if(i != 9)

{

printf(" ");

}

}

return 0;

}

D – D

Given an array a*a* of n*n* elements, print any value that appears at least three times or print -1 if there is no such value.

**Input**

The first line contains an integer t*t* (1 \leq t \leq 10^41≤*t*≤104) — the number of test cases.

The first line of each test case contains an integer n*n* (1 \leq n \leq 2\cdot10^51≤*n*≤2⋅105) — the length of the array.

The second line of each test case contains n*n* integers a\_1, a\_2, \dots, a\_n*a*1​,*a*2​,…,*an*​ (1 \leq a\_i \leq n1≤*ai*​≤*n*) — the elements of the array.

It is guaranteed that the sum of n*n* over all test cases does not exceed 2\cdot10^52⋅105.

**Output**

For each test case, print any value that appears at least three times or print -1 if there is no such value.

**Sample 1**

| **Inputcopy** | **Outputcopy** |
| --- | --- |
| 7  1  1  3  2 2 2  7  2 2 3 3 4 2 2  8  1 4 3 4 3 2 4 1  9  1 1 1 2 2 2 3 3 3  5  1 5 2 4 3  4  4 4 4 4 | -1  2  2  4  3  -1  4 |

**Note**

In the first test case there is just a single element, so it can't occur at least three times and the answer is -1.

In the second test case, all three elements of the array are equal to 22, so 22 occurs three times, and so the answer is 22.

For the third test case, 22 occurs four times, so the answer is 22.

For the fourth test case, 44 occurs three times, so the answer is 44.

For the fifth test case, 11, 22 and 33 all occur at least three times, so they are all valid outputs.

For the sixth test case, all elements are distinct, so none of them occurs at least three times and the answer is -1.

#include <stdio.h>

#include <string.h>

int main()

{

char s[1001], c[11]={0};

char a[11]="0123456789";

//gets(s);

scanf("%[^\n]", s);

int len=strlen(s);

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

{

for(int j=0; j<10; j++)

{

if(s[i]==a[j])

{

c[j]++;

}

}

}

for(int i=0; i<10; i++)

{

printf("%d", c[i]);

if(i != 9)

{

printf(" ");

}

}

return 0;

}

E – E

**Task**

In this challenge, you have to input a five digit number and print the sum of digits of the number.

**Input Format**

The input contains a single five digit number, *n*.

**Constraints**

*10000 ≤ n ≤ 99999*

**Output Format**

Print the sum of the digits of the five digit number.

**Sample Input**

*10564*

**Sample Output**

*16*

#include<stdio.h>

int main()

{

int n, sum=0;

scanf("%d",&n);

while(n>0)

{

sum=sum+n%10;

n=n/10;

}

printf("%d", sum);

return 0;

}

F – F

A telephone number is a sequence of exactly 11 digits, where the first digit is 8. For example, the sequence 80011223388 is a telephone number, but the sequences 70011223388 and 80000011223388 are not.

You are given a string s*s* of length n*n*, consisting of digits.

In one operation you can delete any character from string s*s*. For example, it is possible to obtain strings 112, 111 or 121 from string 1121.

You need to determine whether there is such a sequence of operations (possibly empty), after which the string s*s* becomes a telephone number.

**Input**

The first line contains one integer t*t* (1 \le t \le 1001≤*t*≤100) — the number of test cases.

The first line of each test case contains one integer n*n* (1 \le n \le 1001≤*n*≤100) — the length of string s*s*.

The second line of each test case contains the string s*s* (|s| = n∣*s*∣=*n*) consisting of digits.

**Output**

For each test print one line.

If there is a sequence of operations, after which s*s* becomes a telephone number, print YES.

Otherwise, print NO.

**Sample 1**

| **Inputcopy** | **Outputcopy** |
| --- | --- |
| 2  13  7818005553535  11  31415926535 | YES  NO |

**Note**

In the first test case you need to delete the first and the third digits. Then the string 7818005553535 becomes 88005553535.

#include <stdio.h>

#include <string.h>

int main()

{

int t;

scanf("%d", &t);

for(int k=0; k<t; k++)

{

int n;

scanf("%d", &n);

char arr[n];

scanf("%s", &arr);

int flag=0;

for(int i = 0; i <=n-11; i++)

{

if(arr[i]=='8')

{

flag=1;

break;

}

}

if(flag==1)

{

printf("YES\n");

}

else

{

printf("NO\n");

}

}

return 0;

}

G – G

As it's the first of April, Heidi is suspecting that the news she reads today are fake, and she does not want to look silly in front of all the contestants. She knows that a newspiece is fake if it contains heidi as a subsequence. Help Heidi assess whether the given piece is true, but please be discreet about it...

**Input**

The first and only line of input contains a single nonempty string *s* of length at most 1000 composed of lowercase letters (a-z).

**Output**

Output YES if the string *s* contains heidi as a subsequence and NO otherwise.

**Sample 1**

| **Inputcopy** | **Outputcopy** |
| --- | --- |
| abcheaibcdi | YES |

**Sample 2**

| **Inputcopy** | **Outputcopy** |
| --- | --- |
| hiedi | NO |

**Note**

A string *s* contains another string *p* as a subsequence if it is possible to delete some characters from *s* and obtain *p*.

#include <stdio.h>

#include <string.h>

#include <stdbool.h>

bool isSubSequence(char s1[1001], char s2[1001]);

int main()

{

char a[]="heidi";

char b[1001];

scanf("%s", &b);

if(isSubSequence(a, b))

{

printf("YES");

}

else

{

printf("NO");

}

return 0;

}

bool isSubSequence(char s1[1001], char s2[1001])

{

int len1= strlen(s1);

int len2= strlen(s2);

int i=0, j=0;

while(i<len1 && j<len2)

{

if(s1[i]==s2[j])

{

i++;

}

j++;

}

return i==len1;

}

H – H

The students of the sophomore year from the CSE department of SUST decided to build a humanoid robot that can talk like human beings. Within a short period of time, they finished making the amazing robot. But there was a problem. The robot was not responding to any human language. It was being instructed by only using binary strings. But the head of the robot developer team was cool about it. She came and whispered to my ears and said that the robot will only respond in human language if the binary string contains 1 or 0 simultaneously seven times in a single row.  
  
For example, giving instruction 001111111010100 to the robot will make it understand the human language. But if it is 1110000111 then it will not understand.

**Input**

The first input line contains a non-empty string, consisting of characters "0" and "1", which represents a instructions.  
The length of the string is less than or equal to 100 characters.  
The string contains at least one "0" and one "1".

**Output**

Print "YES" if the can understand human language. Otherwise, print "NO".

**Examples**

**Input**

00111001

**Output**

NO

**Input**

1001000000001

**Output**

YES

Input

101111111001

**Output**

YES

#include <stdio.h>

#include <string.h>

#include <stdbool.h>

bool isSubSequence(char s1[1001], char s2[1001]);

int main()

{

char a[]="heidi";

char b[1001];

scanf("%s", &b);

if(isSubSequence(a, b))

{

printf("YES");

}

else

{

printf("NO");

}

return 0;

}

bool isSubSequence(char s1[1001], char s2[1001])

{

int len1= strlen(s1);

int len2= strlen(s2);

int i=0, j=0;

while(i<len1 && j<len2)

{

if(s1[i]==s2[j])

{

i++;

}

j++;

}

return i==len1;

}

I – I

Sezu was learning numbers and letters. She just learned to write **"zero"** and **"one"**. One day she wrote some numbers in some cards sequentially where a card contains only one letter. Unfortunately when she was going to her mother to show her, those cards got shuffled. Can you help her mother to find what numbers she wrote on those cards?

**Input**

The first line contains a single integer n*n* (1 \leqslant n \leqslant 10^51⩽*n*⩽105) — the length of the string. The second line contains a string consisting of English lowercase letters: 'z', 'e', 'r', 'o' and 'n'.

It is guaranteed that it is possible to rearrange the letters in such a way that they form a sequence of words, each being either "zero" which corresponds to the digit 00 or "one" which corresponds to the digit 11.

**Output**

Print the maximum possible number in binary notation means 1's will be first. Print binary digits separated by a space. The leading zeroes are allowed.

**Sample 1**

| **Inputcopy** | **Outputcopy** |
| --- | --- |
| 4  ezor | 0 |

**Sample 2**

| **Inputcopy** | **Outputcopy** |
| --- | --- |
| 10  nznooeeoer | 1 1 0 |

**Note**

In the first example, the correct initial ordering is "zero".

In the second example, the correct initial ordering is "oneonezero".

#include <stdio.h>

#include <string.h>

int main()

{

char s[101];

scanf(" %s", s);

int len=strlen(s);

int flag=0;

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

{

int count=1;

for(int j=i+1; j<len; j++)

{

if(s[i]!=s[j])

{

break;

}

count++;

if(count>2)

{

i++;

}

}

if(count>=7)

{

flag=1;

break;

}

}

if(flag)

{

printf("YES\n");

}

else

{

printf("NO\n");

}

return 0;

}

J – J

Caps Lock is a button on a computer keyboard that causes all letters of Latin and Cyrillic based scripts to be generated in capital letters. It is a toggle key: each press reverses the previous action. Some keyboards also implement a light, to give visual feedback about whether it is on or off. Exactly what Caps Lock does depends on the keyboard hardware, the operating system, the device driver, and the keyboard layout. Usually, the effect is limited to letter keys; letters of Latin-based scripts are capitalized, while letters of other texts (e.g. Arabic, Hebrew, Hindi) and non-letter characters are generated normally. Whenever the key gets engaged, the shift keys may be used to type lowercase letters on many operating systems, but not on an macOS.

Let's consider that a word has been typed with the Caps lock key accidentally switched on, if:

* either it only contains uppercase letters;
* or all letters except for the first one are uppercase.

In this case we should automatically change the case of all letters. For example, the case of the letters that form words "hELLO", "HTTP", "z" should be changed.

Write a program that applies the rule mentioned above. If the rule cannot be applied, the program should leave the word unchanged.

**Input**

The first line of the input data contains a word consisting of uppercase and lowercase Latin letters. The word's length is from 1 to 100 characters, inclusive.

**Output**

Print the result of the given word's processing.

**Examples**

**Input**

cAPS

**Output**

Caps

**Input**

Lock

**Output**

Lock

#include <stdio.h>

#include <string.h>

int main()

{

char s[101];

scanf("%[^\n]", s);

int count=0;

int len=strlen(s);

for(int i=1; i<len; i++)

{

if(s[i]>='A' && s[i]<='Z')

{

count++;

}

}

if(count==len-1)

{

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

{

if(s[i]>='A' && s[i]<='Z')

{

s[i]= s[i]+32;

}

else if(s[i]>='a' && s[i]<='z')

{

s[i]= s[i]-32;

}

}

}

printf("%s", s);

return 0;

}

#include <stdio.h>

#include <string.h>

#include <stdbool.h>

int main()

{

char s[101];

scanf("%[^\n]", s);

char ch;

bool c= true;

int len=strlen(s);

for(int i=1; i<len; i++)

{

if(islower(s[i]))

{

c= false;

}

}

if(c==true)

{

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

{

if(islower(s[j]))

{

ch= toupper(s[j]);

}

else

{

ch= tolower(s[j]);

}

printf("%c", ch);

}

}

else

{

printf("%s", s);

}

return 0;

}