https://progcont.hu/progcont/100129/?pid=200777

1. Arithmetic Operations #1

Write a program that reads integers and operators of the C programming language from the standard input line by line, performs the operation indicated by the operator on the scanned/read values and writes the result to the standard output.

Input Specification

The input contains several test cases. The first line of the input contains a positive integer *n* indicating the number of test cases. Each of the following lines describes a test case. Each test case is a line in the form "*x op y*", where *x* and *y* are integers, while *op* is one of the following integer arithmetic operators of the C programming language: `+`, `-`, `*`, `/`, `%`.

The operands are separated from the operator by a space character.

Output Specification

For each test case, the program must write a single line containing the result of the operation *x op y* specified in the test case.

Sample Input

```
5
5 + -5
165 - -10
-5 * -2
23 / 8
25 % 3
```

Output for Sample Input

```
0
175
10
2
1
```

Solution 1

```
#include <stdio.h>
int main()
{
    int i, n, a, b;
    char op;
    scanf("%d", &n);
    for(i = 0; i < n; i++)
    {
        scanf("%d %c %d",&a,&op,&b);
        if(op == '+')
            printf("%d\n", a + b);
        if(op == '-')
            printf("%d\n", a - b);
        if(op == '*')
            printf("%d\n", a * b);
        if(op == '/')
```

```
            printf("%d\n", a / b);
        if(op == '%')
            printf("%d\n", a % b);
    }
    return 0;
}
```

Solution 2

```
#include <stdio.h>
int main()
{
    int n, i, a, b;
    char op;
    scanf("%d", &n);
    for(i = 0; i < n; i++)
    {
        scanf("%d %c %d", &a, &op, &b);
        switch(op)
        {
        case'+':
            printf("%d\n",a+b);
            break;
        case'-':
            printf("%d\n",a-b);
            break;
        case'*':
            printf("%d\n",a*b);
            break;
        case'/':
            printf("%d\n",a/b);
            break;
        case'%':
            printf("%d\n",a%b);
            break;
        }
    }
    return 0;
}
```

https://progcont.hu/progcont/100129/?pid=200778

2. Arithmetic Operations #2

Write a program that reads integers and operators of the C programming language from the standard input line by line, performs the operation indicated by the operator on the scanned/read values and writes the result to the standard output.

Input Specification

Each of the input lines contains a test case until end-of-file (EOF). Each test case is a line in the form "*x op y*", where *x* and *y* are integers, while *op* is one of the following integer arithmetic operators of the C programming language: `+`, `-`, `*`, `/`, `%`.

The operands are separated from the operator by a space character.

Output Specification

For each test case, the program must write a single line containing the result of the operation *x op y* specified in the test case.

Sample Input

```
5 + -5
165 - -10
-5 * -2
23 / 8
25 % 3
```

Output for Sample Input

```
0
175
10
2
1
```

Solution 1

```
#include <stdio.h>
int main()
{
    int a, b;
    char op;

    while(scanf("%d %c %d",&a,&op,&b)!=EOF){
        if(op == '+')
            printf("%d\n", a + b);
        if(op == '-')
            printf("%d\n", a - b);
        if(op == '*')
            printf("%d\n", a * b);
        if(op == '/')
            printf("%d\n", a / b);

        if(op == '%')
            printf("%d\n", a % b);
    }
    return 0;
}
```

Solution 2

```
#include <stdio.h>
int main()
{
    int a, b;
    char op;
    while(scanf("%d %c %d",&a,&op,&b)!=EOF){
        switch(op)
        {
```

```
        case'+':
            printf("%d\n",a+b);
            break;
        case'-':
            printf("%d\n",a-b);
            break;
        case'*':
            printf("%d\n",a*b);
            break;
        case'/':
            printf("%d\n",a/b);
            break;
        case'%':
            printf("%d\n",a%b);
            break;
        }
    }
    return 0;
}
```

https://progcont.hu/progcont/100129/?pid=200779

3. Arithmetic Operations #3

Write a program that reads integers and operators of the C programming language from the standard input line by line, performs the operation indicated by the operator on the scanned/read values and writes the result to the standard output.

Input Specification

Each of the input lines describes a test case. Each test case is a line in the form "*x op y*", where *x* and *y* are integers, while *op* is one of the following integer arithmetic operators of the C programming language: `+`, `-`, `*`, `/`, `%`. The operands are separated from the operator by a space character. Scanning should continue until the value of *x* or *y* is 0.

The last test case should not be processed.

Output Specification

For each test case, the program must write a single line containing the result of the operation *x op y* specified in the test case.

Sample Input

```
5 + -5
165 - -10
-5 * -2
23 / 8
25 % 3
6 * 0
```

Output for Sample Input

```
0
175
10
```

```
2
1
```

Solution 1

```
#include <stdio.h>
int main()
{
    int a, b;
    char op;

    while(1){
      scanf("%d %c %d",&a,&op,&b);
      if(a == 0 || b == 0)
           break;
       if(op == '+')
            printf("%d\n", a + b);
        if(op == '-')
            printf("%d\n", a - b);
        if(op == '*')
            printf("%d\n", a * b);
        if(op == '/')
            printf("%d\n", a / b);
        if(op == '%')
            printf("%d\n", a % b);
    }
    return 0;
}
```

Solution 2

```
#include<stdio.h>
int main()
{
    int a, b;
    char op;
    while(1){
      scanf("%d %c %d",&a,&op,&b);
      if(a == 0 || b == 0)
           break;
      switch(op)
        {
        case'+':
            printf("%d\n",a+b);
            break;
        case'-':
            printf("%d\n",a-b);
            break;
        case'*':
            printf("%d\n",a*b);
            break;
        case'/':
            printf("%d\n",a/b);
            break;
```

```
        case'%':
            printf("%d\n",a%b);
            break;
        }
    }
    return 0;
}
```

https://progcont.hu/progcont/100129/?pid=200780

4. Average of Integers #1

Write a program that reads integers from the standard input and computes their average.

Input specification

The first line of the input contains a positive integer *n*, which is followed by *n* integers (one per line).

Output specification

The program should write a single line to the standard output containing the average of the *n* numbers with a precision of two decimal places.

Sample Input

```
5
-2
-1
0
1
2
```

Output for Sample Input

```
0.00
```

Solution

```
#include <stdio.h>
int main()
{
    int n, i, cases, sum=0;
    scanf("%d", &cases);
    for(i = 0; i < cases; i++)
    {
        scanf("%d",&n);
        sum+=n;
    }
    printf("%.2f\n",(float)sum/cases);
    return 0;
}
```

https://progcont.hu/progcont/100129/?pid=200781

5. Average of Integers #2

Write a program that reads integers from the standard input and computes their average.

Input specification

Each of the input lines contains an integer. The input is terminated by end-of-file (EOF).

Output specification

The program should write a single line to the standard output containing the average of the numbers with a precision of two decimal places.

Sample Input

```
-1
-2
0
1
2
```

Output for Sample Input

```
0.00
```

Solution

```
#include <stdio.h>
int main()
{
    int n, sum=0, cases=0;
    while(scanf("%d", &n)!=EOF)
    {
        sum+=n;
        cases++;
    }

    printf("%.2f\n",(float)sum/cases);
    return 0;
}
```

https://progcont.hu/progcont/100129/?pid=200782

6. Average of Integers #3

Write a program that reads integers from the standard input and computes their average.

Input specification

Each of the input lines contains an integer. The reading should continue until the scanned value is 0. Do not take into account this 0 value (denoting the end of the input) when calculating the average.

Output specification

The program should write a single line to the standard output containing the average of the numbers with a precision of two decimal places.

Sample Input

```
-2
-1
1
2
0
```

Output for Sample Input

```
0.00
```

```
#include <stdio.h>
int main()
{
    int n, sum=0, cases=0;
    while(1)
    {
        scanf("%d", &n);
        if (n==0)
            break;
        sum+=n;
        cases++;
    }

    printf("%.2f\n",(float)sum/cases);
    return 0;
}
```

https://progcont.hu/progcont/100129/?pid=200783

7. Minimum of Integer Numbers #1

Write a program that reads integers from the standard input and determines their minimum.

Input specification

The first line of the input contains a positive integer *n*, which is followed by *n* integers (one per line).

Output specification

The program should write a single line to the standard output containing the minimum of the *n* numbers.

Sample Input

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

Output for Sample Input

```
-4
```

Solution

```
#include <stdio.h>
int main()
{
    int cases, n, i, min;
    scanf("%d", &cases);
    scanf("%d", &n);
    min=n;
    for(i = 1; i < cases; i++)
    {
        scanf("%d", &n);
        if(n < min)
            min = n;
    }
    printf("%d\n", min);
    return 0;
}
```

https://progcont.hu/progcont/100129/?pid=200784

8. Minimum of Integer Numbers #2

Write a program that reads integers from the standard input and determines their minimum.

Input specification

Each of the input lines contains an integer. The input is terminated by end-of-file (EOF).

Output specification

The program should write a single line to the standard output containing the minimum of the numbers.

Sample Input

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

Output for Sample Input

```
-4
```

Solution

```
#include <stdio.h>
int main()
{
    int n, min;
    scanf("%d", &n);
    min=n;
    while(scanf("%d", &n)!=EOF){
```

```
        if(n < min)
            min = n;
    }
    printf("%d\n", min);
    return 0;
}
```

https://progcont.hu/progcont/100129/?pid=200785

9. Maximum of Integer Numbers #1

Write a program that reads integers from the standard input and determines their maximum.

Input specification

The first line of the input contains a positive integer *n*, which is followed by *n* integers (one per line).

Output specification

The program should write a single line to the standard output containing the maximum of the *n* numbers.

Sample Input

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

Output for Sample Input

```
5
```

Solution

```
#include <stdio.h>
int main()
{
    int cases, n, i, max;
    scanf("%d", &cases);
    scanf("%d", &n);
    max=n;
    for(i = 1; i < cases; i++)
    {
        scanf("%d", &n);
        if(n > max)
            max = n;
    }
    printf("%d\n", max);
    return 0;
}
```

https://progcont.hu/progcont/100129/?pid=200786

10. Maximum of Integer Numbers #2

Write a program that reads integers from the standard input and determines their maximum.

Input specification

Each of the input lines contains an integer. The input is terminated by end-of-file (EOF).

Output specification

The program should write a single line to the standard output containing the maximum of the numbers.

Sample Input

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

Output for Sample Input

```
5
```

Solution

```
#include <stdio.h>
int main()
{
    int n, max;
    scanf("%d", &n);
    max=n;
    while(scanf("%d", &n)!=EOF){

        if(n > max)
            max = n;
    }
    printf("%d\n", max);
    return 0;
}
```

https://progcont.hu/progcont/100129/?pid=200787

11. Greatest Common Divisor

Write a program that reads two positive integers from the standard input and determines their greatest common divisor.

Input specification

The input contains two positive integers separated by a space character.

Output specification

The program should write a single line to the standard output containing the greatest common divisor of the two input number.

Sample Input 1

```
15 10
```

Output for Sample Input 1

```
5
```

Sample Input 2

```
2 7
```

Output for Sample Input 2

```
1
```

Solution 1

```
#include <stdio.h>
int main()
{
    int a, b;
    scanf("%d %d", &a, &b);

    while(a!=b)
    {
        if(a>b)
            a-=b;
        else
            b-=a;
    }
    printf("%d\n", a);
    return 0;
}
```

Solution 2

```
#include <stdio.h>
int main()
{
    int a, b, r;
    scanf("%d %d", &a, &b);

    while(r=a%b)
    {
        a=b;
        b=r;
    }
    printf("%d\n",b);
    return 0;
}
```

https://progcont.hu/progcont/100129/?pid=200788

12. Relative Primes

Write a program that reads two positive integers from the standard input and decides whether they are relative primes or not.

Input specification

The input contains two positive integers separated by a space character.

Output specification

The program should write a single line to the standard output containing the string "IGEN" ("yes" in Hungarian) or "NEM" ("no" in Hungarian) depending on whether the two input numbers are relative primes.

Sample Input 1

```
15 10
```

Output for Sample Input 1

```
NEM
```

Sample Input 2

```
2 7
```

Output for Sample Input 2

```
IGEN
```

Solution 1

```
#include <stdio.h>
int main()
{
    int a, b;
    scanf("%d %d", &a, &b);

    while(a!=b)
    {
        if(a>b)
            a-=b;
        else
            b-=a;
    }
    if(b==1)
        printf("IGEN\n");
    else
        printf("NEM\n");

    return 0;
}
```

Solution 2

```
#include <stdio.h>
int main()
{
    int a, b;
    scanf("%d %d", &a, &b);

    while(r=a%b)
```

```
    {
        a=b;
        b=r;
    }
    if(b==1)
        printf("IGEN\n");
    else
        printf("NEM\n");

    return 0;
}
```

https://progcont.hu/progcont/100129/?pid=200789

13. Prime Test

Write a program that reads a positive integer from the standard input and decides whether it is a prime or not.

Input specification

The input contains one positive integer number.

Output specification

The program should write a single line to the standard output containing the string "`IGEN`" ("yes" in Hungarian) or "`NEM`" ("no" in Hungarian) depending on whether the scanned number is a prime.

Sample Input 1

```
2
```

Output for Sample Input 1

```
IGEN
```

Sample Input 2

```
4
```

Output for Sample Input 2

```
NEM
```

Solution

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

int main()
{
    int n, i, flag=0;
    scanf("%d", &n);

    for(i=2; i<=sqrt(n); i++)
    {
        if(n%i==0)
        {
            flag=1;
            break;
        }
```

```
    }

    if(flag==1)
        printf("NEM\n");
    else
        printf("IGEN\n");

return 0;
}
```

https://progcont.hu/progcont/100129/?pid=200790

14. Grading #1

Write a program that will help you grade the students' achievements. The maximum score available on the test is 100 points, and the student's mark is

- **excellent** if he scored at least 80 points;
- **good** if he scored at least 70 points but did not reach 80 points;
- **satisfactory** if he scored at least 60 points but did not reach 70 points;
- **pass** it he scored at least 50 points but did not reach 60 points;
- **fail** if he did not reach 50 points.

Input specification

The first line of the input contains a positive integer *n*. Each of the following *n* lines contains a student's score between 0 and 100.

Output specification

For each student, the program should write exactly one line to the standard output, which contains the mark he achieved, that is, one of "jeles", "jo", "kozepes", "elegseges", and "elegtelen" (the Hungarian counterparts of "excellent", "good", "satisfactory", "pass", and "fail", respectively).

Sample Input

```
2
45
90
```

Output for Sample Input

```
elegtelen
jeles
```

Solution

```
#include <stdio.h>
int main()
{
        int pont,n,i;
        scanf("%d",&n);
        for(i=1; i<=n; i++)
        {
                scanf("%d",&pont);
                if (pont>=80)
                 printf("jeles\n");
                else if (pont>=70)
```

```
                printf("jo\n");
        else if (pont>=60)
                printf("kozepes\n");
        else if (pont>=50)
                printf("elegseges\n");
        else
                printf("elegtelen\n");
    }
    return 0;
}
```

https://progcont.hu/progcont/100129/?pid=200791

15. Grading #2

Write a program that will help you grade the students' achievements. The maximum score available on the test is 100 points, and the student's mark is

- **excellent** if he scored at least 80 points;
- **good** if he scored at least 70 points but did not reach 80 points;
- **satisfactory** if he scored at least 60 points but did not reach 70 points;
- **pass** it he scored at least 50 points but did not reach 60 points;
- **fail** if he did not reach 50 points.

Input specification

Each line of the input contains a student's score between 0 and 100.

The input is terminated by end-of-file (EOF).

Output specification

For each student, the program should write exactly one line to the standard output, which contains the mark he achieved, that is, one of "jeles", "jo", "kozepes", "elegseges", and "elegtelen" (the Hungarian counterparts of "excellent", "good", "satisfactory", "pass", and "fail", respectively).

Sample Input

```
45
90
```

Output for Sample Input

```
elegtelen
jeles
```

Solution

```
#include <stdio.h>
int main()
{
    int pont;
    while(scanf("%d", &pont) != EOF)
    {

        if(pont >= 80)
            printf("jeles\n");
```

```
        else if(pont >= 70)
            printf("jo\n");
        else if(pont >= 60)
            printf("kozepes\n");
        else if(pont >= 50)
            printf("elegseges\n");
        else
            printf("elegtelen\n");
    }
    return 0;
}
```

https://progcont.hu/progcont/100129/?pid=200792

16. Grading #3

Write a program that will help you grade the students' achievements. The maximum score available on the test is 100 points, and the student's mark is

- **excellent** if he scored at least 80 points;
- **good** if he scored at least 70 points but did not reach 80 points;
- **satisfactory** if he scored at least 60 points but did not reach 70 points;
- **pass** it he scored at least 50 points but did not reach 60 points;
- **fail** if he did not reach 50 points.

Input specification

Each line of the input contains a student's score between 0 and 100.

The input is terminated by a line containing a negative integer (this test case should not be processed).

Output specification

For each student, the program should write exactly one line to the standard output, which contains the mark he achieved, that is, one of "jeles", "jo", "kozepes", "elegseges", and "elegtelen" (the Hungarian counterparts of "excellent", "good", "satisfactory", "pass", and "fail", respectively).

Sample Input

```
45
90
-1
```

Output for Sample Input

```
elegtelen
jeles
```

Solution

```
#include <stdio.h>
int main ()
{
    int pont;
    while (1)
    {
        scanf("%d", &pont);
```

```
        if (pont <0)
            break;
        if (pont >= 80)
            printf("jeles\n");
        else if ( pont >= 70)
            printf("jo\n");
        else if ( pont >= 60)
            printf("kozepes\n");
        else if ( pont >= 50)
            printf("elegseges\n");
        else
            printf("elegtelen\n");
    }

    return 0;
}
```

https://progcont.hu/progcont/100033/?pid=200517

17. The Biggest

Write a program that reads integers from the standard input.

The first number of each line determines how many additional numbers are given in that line.

For each line, the program should write to the standard output how many times the biggest number occurs from the second number to the end of the line. The input is terminated by a line containing only one 0, this line should not be processed by the program.

Sample Input

```
10 5 2 3 5 4 1 3 5 1 3
1 1
5 5 5 5 5 5
3 1 2 3
0
```

Output for Sample Input

```
3
1
5
1
```

Solution

```
#include <stdio.h>
int isMax(int *number, int size)
{
    int max = 0, i;
    for(i=1; i<size; i++)
        if(number[i]>number[max])
            max = i;
    return number[max];
}

int countTimes(int *number, int size)
```

```
{
    int count=0, i;
    int max = isMax(number, size);
    for(i=0; i<size; i++)
        if(number[i]==max)
            count++;
    return count;
}

int main()
{
    int size, i;
    scanf("%d", &size);
    while(size)
    {
        int number[size];
        for(i=0; i<size; i++)
            scanf("%d", &number[i]);
        printf("%d\n", countTimes(number,size));
        scanf("%d", &size);
    }
    return 0;
}
```

https://progcont.hu/progcont/100033/?pid=200518

18. Which Day?

Write a program that reads dates in "*MM*.*DD*." format from the standard input until either *MM* or *DD* is zero. The program should write the name of the day to the standard output for each line.

On which day does the given date fall? We can assume that the first day of the year falls on Monday, and the year is not a leap year. See the English names of the days in the sample output.

Sample Input

```
01.01.
03.13.
10.17.
12.27.
08.17.
04.21.
07.15.
00.05.
```

Output for Sample Input

```
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
Sunday
```

Solution 1

```
#include <stdio.h>

int whichMonth(int month)
{
    if(month == 1 || month == 3 || month == 5 || month == 7 || month == 8
            || month == 10 || month == 12)
        return 31;
    else if(month == 4 || month == 6 || month == 9 || month == 11)
        return 30;
    else
        return 28;
}

void whichDay(int day)
{
    day %= 7;
    switch(day)
    {
    case 1:
        printf("Monday\n");
        break;
    case 2:
        printf("Tuesday\n");
        break;
    case 3:
        printf("Wednesday\n");
        break;
    case 4:
        printf("Thursday\n");
        break;
    case 5:
        printf("Friday\n");
        break;
    case 6:
        printf("Saturday\n");
        break;
    case 0:
        printf("Sunday\n");
        break;
    }
}

int main()
{
    int month, day, i;
    scanf("%d.%d.", &month, &day);
    while(month != 0 && day != 0)
    {
        int sum = 0;
        if(month > 1)
        {
            for( i = 1; i < month; i++)
                sum += whichMonth(i);
        }
        sum += day;
        whichDay(sum);
        scanf("%d.%d.", &month, &day);
    }
    return 0;
```

```
}
```

Solution 2

```
#include <stdio.h>
int main()
{
    int month, day, i;
    int T[12]= {31,28,31,30,31,30,31,31,30,31,30,31};

    while(scanf("%d.%d.",&month,&day))
    {
        int sum=0;
        if(month==00 || day==00)
            break;
        if(month==1)
            sum=sum;
        else
            for(i=0; i<month-1; i++)
                 sum=sum+T[i];

        sum=sum+day;

        if(sum%7==1)
            printf("Monday\n");
        if(sum%7==2)
            printf("Tuesday\n");
        if(sum%7==3)
            printf("Wednesday\n");
        if(sum%7==4)
            printf("Thursday\n");
        if(sum%7==5)
            printf("Friday\n");
        if(sum%7==6)
            printf("Saturday\n");
        if(sum%7==0)
            printf("Sunday\n");

    }
    return 0;
}
```

https://progcont.hu/progcont/100033/?pid=200519

19. SMS

Write a program that reads English forenames from the standard input until end-of-file (EOF), made up of arbitrary number of English uppercase letters, one forename per line. The program should write to the standard output which numeric keys are to be pressed in a traditional mobile phone with a numeric keypad, and in what order, if we want to send the strings scanned by the program as SMS messages with as few keypresses as possible. (For example, the name ANDREW can be entered as 2663777339.)

Sample Input

```
ANDREW
JOHN
ADAM
SARAH
```

Output for Sample Input

```
2663777339
56664466
2326
77772777244
```

Solution

```
#include<stdio.h>

int main()
{
    char c =' ';
    while((c=getchar())!=EOF )
       {
          if(c!='\n')
          {
            if(c=='A')printf("2");
            if(c=='B')printf("22");
            if(c=='C')printf("222");
            if(c=='D')printf("3");
            if(c=='E')printf("33");
            if(c=='F')printf("333");
            if(c=='G')printf("4");
            if(c=='H')printf("44");
            if(c=='I')printf("444");
            if(c=='J')printf("5");
            if(c=='K')printf("55");
            if(c=='L')printf("555");
            if(c=='M')printf("6");
            if(c=='N')printf("66");
            if(c=='O')printf("666");
            if(c=='P')printf("7");
            if(c=='Q')printf("77");
            if(c=='R')printf("777");
            if(c=='S')printf("7777");
```

```
            if(c=='T')printf("8");
            if(c=='U')printf("88");
            if(c=='V')printf("888");
            if(c=='W')printf("9");
            if(c=='X')printf("99");
            if(c=='Y')printf("999");
            if(c=='Z')printf("9999");

          }
          else printf("\n");
       }

    return 0;
}
```

https://progcont.hu/progcont/100033/?pid=200520

20. Three is the Hungarian Justice

Write a program that reads an English word of up to 24 characters per line from the standard input until you the word “justice” is read for the third time. For each scanned word, including the third “justice”, determine whether it is shorter than 10 characters, longer than 10 characters, or exactly 10 characters long, and write to the standard output in a separate line a < (less than), > (greater than) or = (equal) character, respectively.

Sample Input

```
program
justice
programming
justice
programmer
justice
programmability
programmable
```

Output for Sample Input

```
<
<
>
<
=
<
```

Solution

```
#include<stdio.h>
#include<string.h>
int main()
{
    int count=0;
    char s[24];
    while(scanf("%s",s))
    {
        if (strcmp(s,"justice")==0)
            count++;
        if (strlen(s)<10)
```

```
            printf("<\n");
        else if (strlen(s)>10)
            printf(">\n");
        else if (strlen(s)==10)
            printf("=\n");

        if (count==3)
            break;
    }
    return 0;
}
```

https://progcont.hu/progcont/100032/?pid=200513

21. Monotonicity

Write a program that reads integers from the standard input. The first number of each line determines how many additional numbers are given in that line. For each line, determine whether the sequence from the second number to the end of the line is monotonous, and if so, write "`monoton`" (monotonous in Hungarian), otherwise "`nem monoton`" (nonmonotonous in Hungarian) to the standard output.

One-element and two-element sequences are considered monotonous. The input is terminated by a line containing only one 0, this line should not be processed.

Sample Input

```
10 1 1 1 1 1 2 2 2 2 2
3 2 3 2
3 3 3 2
1 0
4 -1 0 1 0
0
```

Output for Sample Input

```
monoton
nem monoton
monoton
monoton
nem monoton
```

Solution

```
#include<stdio.h>
int isMonoIncrease(int a[],int n)
{
    for(int i=1; i<n; i++)
        if(a[i]<a[i-1])
            return 0;
    return 1;
}
int isMonoDecrease(int a[], int n)
{
    for(int i=1; i<n; i++)
        if(a[i]>a[i-1])
            return 0;
```

```
    return 1;
}
int main()
{
    int n;
    scanf("%d",&n);

    while(n)
    {
        int a[n];
        for(int i=0; i<n; i++)
             scanf("%d",&a[i]);

        if(isMonoDecrease(a,n) || isMonoIncrease(a,n))
            printf("monoton\n");
        else
            printf("nem monoton\n");

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

    return 0;
}
```

https://progcont.hu/progcont/100032/?pid=200516

22. Palindromes

Write a program that reads strings of up to 29 characters from the standard input, one per line, until the scanned string is a palindromic string. Palindromes have the same characters in the same order when read backwards as if read forwards. With the exception of the last string scanned, the program should write to the standard output each string and its length, separated by a space character.

Sample Input

```
alma
eper retek cseresznye
ribizli
komor romok
```

Output for Sample Input

```
alma 4
eper retek cseresznye 21
ribizli 7
```

Solution

```
#include<stdio.h>
#include<string.h>
int main(){
     int i=0,len,c=0;
     char s[50];
```

```
      while(1){
            gets(s);
            c=0;
            len=strlen(s);
            for(i=0;i<len;i++)
                  if(s[i]==s[len-i-1])
                        c++;
            if (c==len)return 0;
            printf("%s %d\n",s,strlen(s));
      }
      return 0;
}
```

https://progcont.hu/progcont/100034/?pid=200521

23. Median

Write a program that reads three integers per line from the standard input until three zeros are read. Except for the last scanned line, the program should write the median of the three scanned numbers (the middle of the sorted elements) for each line.

Sample Input

```
3 2 5
3 3 4
-1 0 8
0 5 0
0 0 0
```

Output for Sample Input

```
3
3
0
0
```

Solution 1

```
# include <stdio.h>
int main()
{
    int a,b,c;
    while(1)
    {
        scanf("%d %d %d", &a, &b, &c);
        if (a==0 && b==0 && c==0)
            break;
        if ((b<=a && b>=c) || (a<=b && b<=c))
            printf("%d\n", b);
        else if((c>=a && b>=c) || (a>=c && c>=b))
            printf("%d\n", c);
        else if ((a<=b && a>=c) || (a>=b && a<=c))
            printf("%d\n", a);
    }
    return 0;
```

```
}
```

Solution 2

```
#include <stdio.h>

void sort(int a[], int size)
{
    int i, j, tmp;
    for(i = 0; i < size -1 ; ++i)
        for(j = i + 1; j < size; ++j)
        if (a[i] > a[j])
    {
        tmp = a[i];
        a[i] = a[j];
        a[j] = tmp;
    }
}

int main()
{
    int t[3], a,b,c;

    while(1)
    {
       scanf("%d %d %d", &a, &b, &c);
        if ( a == 0 && b == 0 && c == 0 )
            break;
        t[0] = a;
        t[1] = b;
        t[2] = c;

       sort(t,3);

       printf("%d\n",t[1]);
    }

    return 0;
}
```

https://progcont.hu/progcont/100034/?pid=200522

24. Divisibility Testing

Write a program that reads two positive integers per line from the standard input until the first number is greater than the second. For each line, the program should output those numbers between the two integers scanned that are divisible by 3, but not divisible by 9, separated by exactly one space character. If there are no such numbers between the two integers, –1 should be displayed in the output. Immediately after the last number in each line, the newline character should appear (without an extra space).

Sample Input

```
    10 20
```

```
10 10
12 12
1 2
2 1
```

Output for Sample Input

```
12 15
-1
12
-1
```

Solution 1

```
#include <stdio.h>
int main()
{
    int a, b, i, count;
    while(1)
    {
        scanf("%d %d", &a, &b);
        if(a > b)
            break;
        count = 0;
        for(i=a ; i <= b ; i++)
        {
            if(i % 3 == 0 && i % 9 != 0)
            {
                count++;
                if(count == 1)
                    printf("%d", i);
                else
                    printf(" %d", i);
            }
        }
        if(count == 0)
            printf("-1");
        printf("\n");
    }
    return 0;
}
```

Solution 2

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

int main()
{
    int a, b, i, count, diff;
    while(1)
    {
        scanf("%d %d", &a, &b);
        if(a > b)
```

```
            break;
        count = 0;
        diff=b-a;
        int t[diff+1];
        for(i=a ; i <= b ; i++)
        {
            if(i % 3 == 0 && i % 9 != 0)
            {
                t[count]=i;
                count++;
            }
        }
        if(count == 0)
            printf("-1\n");
        else if (count !=0 )
        {
        for(i=0; i<count-1 ; i++)
                printf("%d ",t[i]);
        printf("%d\n",t[i]);
        }
    }
    return 0;
}
```

https://progcont.hu/progcont/100034/?pid=200523

25. 12-Hour Clock

Write a program that reads times in 12-hour format from the standard input until end-of-file (EOF), one per line. The program should write to the standard output the 24-hour times corresponding to the given times. If the hours are less than 10, display the hours with one digit. Minutes should always be displayed with two digits. For example, 12.02am should be converted to 0.02, 11.58am to 11.58, 12.32pm to 12.32, 1.29pm to 13.29, and 10.17pm to 22.17.

Sample Input

```
12.02am
11.58am
12.32pm
1.29pm
10.17pm
```

Output for Sample Input

```
0.02
11.58
12.32
13.29
22.17
```

Solution

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

```
int main()
{
    int hour,minute;
    char time[3];
    while(scanf("%d.%d%s",&hour,&minute,time)!=EOF)
    {
        switch(time[0])
        {
        case 'a':
            if(hour==12)
                printf("0.%.2d\n",minute);
            else
                printf("%d.%.2d\n",hour,minute);
            break;
        case 'p':
            if(hour==12|| hour==0)
                printf("%d.%.2d\n",hour,minute);
            else if(hour>0 && hour<12)
                printf("%d.%.2d\n",hour+12,minute);
        }
    }
    return 0;
}
```

https://progcont.hu/progcont/100034/?pid=200524

26. String Reversion

Write a program that reads strings of up to 15 characters from the standard input until the "THE END" string is read, one string per line. The program should write the reverse of the scanned strings to the standard output, each in a separate line.

Sample Input

```
komor romok
almamag
apa a pap
talpra magyar
THE END
```

Output for Sample Input

```
komor romok
gamamla
pap a apa
raygam arplat
```

Solution

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

int main()
{
```

```
    char line[16];
    while ( 1 )
    {
        gets(line);
        if ( strcmp(line,"THE END") == 0 )
            break;
        for( int i = strlen(line) - 1; i >= 0; --i )
            printf("%c", line[i]);
        printf("\n");
    }
    return 0;
}
```

https://progcont.hu/progcont/100102/?pid=200688

27. Pizzas

Write a program that reads pizza data from a text file given as the first command line argument until the end of the file. Each line contains data for one pizza and is structured as follows:

pizza_name;*ingredient*[;*ingredient*]…;*32cm_price*;*45cm_price*

The pizza name is a string of up to 32 characters, containing only English letters, digits, hyphens (minus signs), and space characters. An ingredient is a string of up to 32 characters, containing only English letters and space characters. The ingredients are followed by two positive integers: the *32cm_price* shows the price of a pizza with a diameter of 32 cm, while the *45cm_price* is the price of a pizza with a diameter of 45 cm. Two consecutive data in the line are separated by a semicolon character.

The program should determine and write to the first line of the standard output the number of ingredients of the pizza(s) with the most ingredients. In the following lines, the names of these pizzas should be written in the order of their occurrence in the text file, one by line.

Sample Input

```
    margareta;paradicsom;mozzarella;840;1640
    szalamis;szalami;mozzarella;940;1900
    sonkas-gombas;sonka;gomba;mozzarella;1040;2100
```

Output for Sample Input

```
    3
    sonkas-gombas
```

Solution

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

int main(int argc, char const *argv[])
{
    int db=0,i=0,max=0,j=0;
    FILE *fp;
    char c,nev[40];
```

```
    fp=fopen(argv[1],"r");

    while((c=fgetc(fp))!=EOF)
    {
        if(c==';')
            db++;

        if(c=='\n')
            db=0;

        if(db-2>max)
            max=db-2;
    }
    printf("%d\n",max);
    rewind(fp);
    db=0;
    while((c=fgetc(fp))!=EOF)
    {

        if(c!=';' && db<1)
            nev[j++]=c;

        if(c==';')
            db++;

        if(c=='\n')
        {
            if(db-2==max)
            {
                nev[j]='\0';
                printf("%s\n",nev);
                memset(nev,0,40);
            }
            db=0;
            j=0;
        }
    }

    fclose(fp);
    return 0;
}
```

https://progcont.hu/progcont/100102/?pid=200689

28. The Curse of Greatness

Write a program that receives at least one but otherwise any number of negative integers as command line arguments and writes the value of the largest of these numbers to the standard output.

Sample Command Line Arguments

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

Output for Sample Input

```
-1
```

Solution

```
#include<stdio.h>
int main()
{
    int n, max, first_value=0;
    while(scanf("%d",&n)!=EOF)
    {
        if (first_value==0)
        {
        max=n;
        first_value=1;
        }
        if(n>max)
            max=n;
    }
    printf("%d\n",max);
    return 0;
}
```

https://progcont.hu/progcont/100091?pid=200660

29. Zoo

Write a program that reads zoo data from the standard input until end-of-file. Each line contains data for one zoo and is structured as follows:

zoo_city;*animal_species*[;*animal_species*]…;*foundation_year*;*number_of_visitors*

The *zoo_city* is a string of up to 40 characters, containing only English letters, hyphens (minus signs), and space characters. An *animal_species* is a string of up to 32 characters, containing only English letters, hyphens (minus signs), and space characters. The animal species are followed by two positive integers: the *foundation_year* represents the foundation year of the zoo, while the *number_of_visitors* indicates how many visitors visited the zoo during the past year. Two consecutive data in the line are separated by a semicolon character.

The program should determine and write to the standard output the city of the zoo that hosts the most animal species. If there are multiple zoos with the same maximum number of species, then output the city of the most recent one.

Sample Input

```
Debrecen;degu;tengerimalac;feherkezu gibbon;1958;171612
Veszprem;vari;huszarmajom;orrszarvu;1958;255538
Budapest;aranyhasu mangabe;zsiraf;1866;997057
```

Output for Sample Input

```
Veszprem
```

Solution

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

int main()
{
    char c, zoo[40], maxzoo[40];
    int db=0, i, max=0;
    while((c=getchar())!=EOF)
    {
        zoo[0]=c;
        i=1;
        while((c=getchar())!=';')
        {
            zoo[i]=c;
            i++;
        }
        zoo[i]='\0';
                    while((c=getchar())!='\n')
        {
            if (c==';')
                    db++;
        }
    if (db>=max)
        {
            strcpy(maxzoo, zoo);
            max=db;
        }
        db=0;
    }
    puts(maxzoo);

    return 0;
}
```

https://progcont.hu/progcont/100125?pid=200771

30. Minions

The history of the minions comes from the beginning of time. The minions started as a yellow single-celled organism and evolved through ages and served the most gru-like masters. Since these masters have been lost continuously from T-Rex to Napoleon, the minions now have nobody to serve and have fallen into deep depression.

But a minion named Kevin has a plan. With two companions on his side, he wants to search the world for the new evil leader whom their race can follow. In order to succeed, however, he has to consider thoroughly which two partners to take.

It is your task to write a program for Kevin that reads the data of minions from the standard input until end-of-file, at least 2, up to 20, one per line. A line is structured as follows:

minion_name`;`*hunger*`;`*enthusiasm*`;`*pants_size*

minion_name is a unique string of up to 30 characters, with only English letters. *hunger* and *enthusiasm* are both integers from 0 to 100, while *pants_size* is one of "`S`", "`L`", "`XL`", and "`XXL`". Data in the line are separated from each other by semicolon (`;`) characters.

The better companion of two minions is the one who is more enthusiastic, in case of equally enthusiastic minions, Kevin will rank them based on the lexicographical order of their names.

You have to print exactly two lines to the standard output, both of them containing the name of a minion, his hunger, and the size of his pants in the following format:

minion_name hunger (*pants_size*)

The best travel companion should appear in the first line, and write the second best minion in the second line.

Sample Input

```
Bob;87;100;S
Dave;43;10;L
Stuart;50;30;L
Jerry;40;20;XL
```

Output for Sample Input

```
Bob 87 (S)
Stuart 50 (L)
```

Solution

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

typedef struct
{
    char name[30];
    int hunger;
    int enthusiasm;
    char pants_size[4];
} MINYON;


int compar(const void *a1, const void *b1)
{
    MINYON *a=(MINYON*)a1;
    MINYON *b=(MINYON*)b1;
```

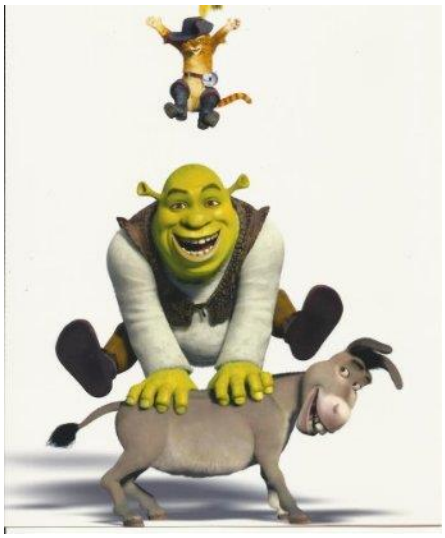
```
    if(a->enthusiasm > b->enthusiasm)
        return -1;
    else if(a->enthusiasm < b->enthusiasm)
        return 1;
    else
        return strcmp(a->name, b->name);
}


int main()
{

    char line[100], *token;
    int i, db=0;
    MINYON minions[20];

    while( fgets(line, 100, stdin) )
    {
        token = strtok(line, ";");
        strcpy(minions[db].name, token);

        token = strtok(NULL, ";");
        minions[db].hunger = atoi(token);

        token = strtok(NULL, ";");
        minions[db].enthusiasm = atoi(token);

        token = strtok(NULL, "\n");
        strcpy(minions[db].pants_size, token);

        db++;
    }

    qsort(minions, db, sizeof(MINYON), compar);

    for(i=0; i<2; i++)
        printf("%s %d (%s)\n", minions[i].name, minions[i].hunger,
minions[i].pants_size);

    return 0;
}
```

https://progcont.hu/progcont/100126/?pid=200774

31. Shrek

Once upon a time, in a far marsh, lived a grumpy ogre called Shrek in cosy loneliness. But the silence and the life of the green monster was disturbed one day by a strange event: fairy-tale creatures swarned the marsh – and surprised our unsuspecting hero.

The three blind mice were running over his dinner, the big wicked wolf was lying in his bed, the seven dwarves put Snow White’s coffin on his desk, and three homeless little pigs were lurking in front of his hut and all the other magical figures that had been chased away by the evil Farquaad lord.



So Shrek is going to the lord to reclaim his marsh.

In the great adventure, the ogre also finds a companion: a talkative Donkey, who would do anything for Shrek, except for one thing: he is not willing to keep his mouth closed for all the treasures of the world.

But the Donkey likes to put his nose into everything, so he offers a deal: he's willing to be slient for a few minutes if Shrek shares with him the characteristics of the three most irritating creatures.

Write a program that reads data of fairy tale creatures from the standard input until end-of-file, at least 3, up to 30, one per line. A line is structured as follows:

name_of_creature`;`*irritation_factor*`;`*note*

name_of_creature is a unique string of at most 30 characters, containing only English letters and spaces. *irritation_factor* is a real number between 0 and 100, while *note* is a string containing up to 30 characters, describing the grievance caused by the creature, with only English letters and spaces. Data in the line are separated from each other by semicolon (`;`) characters.

More annoying of two fairy tale creatures is the one having a higher irritation factor. In case of two equally annoying creatures – to try Shrek's and your knowledge – the Donkey asks them in reversed lexicographical order of their names. Exactly three lines have to be written to the standard output, each of them containing the name, irritation factor, and note of a fairy tale creature in the following format:

creature_name (*irritation_factor*) : *note*

Values describing the irritation factor of the creatures are to be displayed with one decimal precision.

Sample Input

```
A gonosz farkas;47.01;elfoglalta a fekhelyemet
Hofeherke;30.75;jatssza itt a halottat
Egy kismalac;50.22;egyfolytaban rofog
Masik kismalac;60.323;sokat beszel
Harmadik kismalac;50.22;pusztan idegesit
```

Output for Sample Input

```
    Masik kismalac (60.3): sokat beszel
    Harmadik kismalac (50.2): pusztan idegesit
    Egy kismalac (50.2): egyfolytaban rofog
```

Solution

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
typedef struct
{
     char name[30];
     float irritation;
     char note[30];
} SHREK;

int compar(const void *a1, const void *b1)
{
SHREK *a=(SHREK*)a1;
SHREK *b=(SHREK*)b1;

if ( (a->irritation) > (b->irritation) )
     return -1;
else if ( (a->irritation) < (b->irritation) )
     return 1;

else

     return (-1)*strcmp(a->name, b->name);
}

int main()
{

char sor[1000], *token;
SHREK shreks[30];
int i, db=0;

while( fgets(sor,1000,stdin) != NULL )
{
     token=strtok(sor,";");
     strcpy(shreks[db].name, token);

     token=strtok(NULL,";");
     shreks[db].irritation=atof(token);

     token=strtok(NULL,"\n");
     strcpy(shreks[db].note, token);

     db++;
}

qsort(shreks, db, sizeof(SHREK), compar);
```

```
for(i=0; i<3; i++)
     printf("%s (%.1f): %s\n", shreks[i].name, shreks[i].irritation,
shreks[i].note);

return 0;
}
```

https://progcont.hu/progcont/100143/?pid=200843

32. Hiking Trail

Consider the following header file:

myheader.h

```
#ifndef _MYHEADER_H
#define _MYHEADER_H 1

int foo(char *, int);

#endif /* myheader.h */
```

Write the function `foo()` declared in `myheader.h` that takes a string and a nonnegative integer *n* as parameters. The string describes the current state of a hiking trail. In the string, '`!`' (exclamation mark) characters indicate the start, the finish, and the checkpoints of the trail, the lowercase letters of the English alphabet indicate the hikers performing the trail, and '`.`' (period) characters denote other parts of the route.

The function should return the number of sections where more than *n* hikers are staying.

Note

Place the function in file `foo.c` and submit this file as a solution to the evaluation system. You can test your solution using the following files. The evaluation system does not necessarily perform the evaluation using these files.

main.c

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

int main()
{
    char line[1000];
    int n;
    while (scanf("%s %d", line, &n) != EOF)
        printf("%d\n", foo(line, n));
    return EXIT_SUCCESS;
}
```

Makefile

```
SRCS = main.c foo.c
OBJS = $(SRCS:%.c=%.o)
TARGETS = main
.PHONY: clean
```

```
all: $(TARGETS)
main: $(OBJS)
$(CC) $(OBJS) -o main
%.o: %.c
$(CC) -Wall -c $< -o $@
clean:
rm -rf $(OBJS) *~ $(TARGETS)
```

Sample Input

```
!.........! 1
!...ab..c.!..d..e.!...! 1
!...abcde..!....!.....!......!.......! 5
```

Output for Sample Input

```
0
2
0
```

Solution

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

int foo(char *s, int n)
{
      int hikers=0,i,count=0;
      for(i=0;i<strlen(s);i++)
      {
            if(isalpha(s[i]))
            {
                  hikers++;
            }
            else if(s[i] == '!')
            {
                  if(hikers>n)
                        count++;
                  hikers = 0;
            }
      }
      return count;
}
```

https://progcont.hu/progcont/100143/?pid=200846

33. Working Hours

Write a program that helps you calculate the employees' time spent on work.

The input contains several test cases. Each line of a test case gives the length of a time period spent on work in *H.MM* format, where *H* denotes the number of hours ($H \geq 0$), and *MM* denotes the minutes, the latter always given with two digits ($0 \leq MM < 60$).

Test cases are terminated by a line containing the string "END". You may assume that all employees spend at least a short time on work.

The program should write to the standard output the total time spent on work by each employee, in the same format as in the input.

Sample Input

```
    3.20
    4.30
    5.00
    END
    10.00
    END
```

Output for Sample Input

```
    12.50
    10.00
```

Solution 1

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
int main()
{
    char input[5];
    int hour=0,sum=0;
    float time=0,minute=0;
    while(scanf("%s",input)!=EOF)
    {
        if(strcmp(input,"END")!=0)
        {
            hour=atoi(input)*60;
            minute=(atof(input)-atoi(input))*100;
            sum=sum+hour+minute;
            time=(float)(sum/60*100+sum%60)/100.0;
        }
        else
        {
            printf("%.2f\n",time);
            time=0;
            sum=0;
            hour=0;
            minute=0;
        }
    }
    return 0;
}
```

Solution 2

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

```
int main()
{
    char c[50];
    int min,hour,minutes=0;
    while(fgets(c,50,stdin)!=NULL)
    {
        if(strcmp(c,"END\n")!=0)
        {
            sscanf(c,"%d.%d",&hour,&min);
            minutes+=(min+hour*60);
        }
        else
        {
            printf("%d.%02d\n",minutes/60,minutes%60);
            minutes=0;
        }
    }

    return 0;
}
```

https://progcont.hu/progcont/100144/?pid=200850

34. Even Number or Consonant

Each card in a deck has an uppercase letter of the English alphabet on one side, and an integer number on the other side. There is exactly one letter and one number on each card. Somebody makes the following statement about this card deck:

"*If there is an even number on the number side of a card, then its letter side shows a vowel.*"

To determine whether this statement is true, we obviously have to have a look at the other side of certain cards.

Your task is to write a program that reads data describing one side of the cards and writes to the standard output the minimum number of cards that have to be flipped over in the worst case to be able to determine without any doubt whether the above statement is true for all cards in the deck.

The input sonsists of several test cases. Each line of a test case contains either an uppercase letter of the English alphabet or an integer. Test cases are terminated by a line containing the string "`END`". You may assume that each test case contains at least one line describing a card.

For each test case, your program should write a single line with a single integer to the standard output: the minimum number of cards that need to be flipped over.

Sample Input

```
E
K
4
7
END
-1
END
```

Output for Sample Input

```
2
0
```

Solution 1

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

int main()
{
    char input[20];
    int count=0;
    while(scanf("%s",input)!=EOF)
    {
        if(strcmp(input,"END")==0)
        {
            printf("%d\n",count);
            count=0;
        }

        else if((input[0]>='A')&&(input[0]<='Z'))
        {
            switch(input[0])
            {
            case 'A': break;
            case 'E': break;
            case 'I': break;
            case 'O': break;
            case 'U': break;
            default: count++;  break;
            }
        }
        else if(atoi(input)%2==0)
        {
            count++;
        }
    }

    return 0;
}
```

Solution 2

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

int main()
{
    char input[50];
    int count=0;
```

```
    while(scanf("%s", input)!=EOF)
    {
        if(strcmp(input, "END"))
        {
           if(isupper(input[0]))
                {
                    if(input[0]!='A'      &&      input[0]!='O'      &&
input[0]!='I' && input[0]!='U' && input[0]!='E' )
                        count++;
                }
            else
            {
                if(atoi(input)%2==0)
                    count++;
            }
        }
        else
        {
            printf("%d\n", count);
            count=0;
        }
    }
    return 0;
}
```

https://progcont.hu/progcont/100144/?pid=200847

35. If There Is No Cat at Home, the Mice Squeak

Consider the following header file:

myheader.h

```
    #ifndef _MYHEADER_H
    #define _MYHEADER_H 1

    int foo(char *, int);

    #endif /* myheader.h */
```

Write the function `foo()` declared in `myheader.h` that takes a string and a positive integer *n* as parameters. The string describes the wall of a room with mouse holes, where '`O`' (uppercase letter O) indicates a mouse hole, '`>`' denotes a mouse looking to the right, '`<`' a mouse looking to the left, and '`.`' (period) characters denote the rest of the wall. At each end of the wall, in the corners of the room, there are mouse holes.

British scientists have proved that if mice smell a cat in the vicinity, they start running in the direction they are facing at the moment. Each mouse travels one unit per second, and as soon as they reach a mouse hole, they hide in there. If two mice meet along the wall while fleeing, they pass by each other neatly without time loss and run further.

The function should return true if a cat arriving after *n* seconds does not find any mice beside the wall, and false otherwise.

Note

Place the function in file `foo.c` and submit this file as a solution to the evaluation system. You can test your solution using the following files. The evaluation system does not necessarily perform the evaluation using these files.

main.c

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


int main()
{
    char line[1000];
    int n;
    while (scanf("%s %d", line, &n) != EOF)
        puts(foo(line, n) ? "YES" : "NO");
    return EXIT_SUCCESS;
}
```

Makefile

```
SRCS = main.c foo.c
OBJS = $(SRCS:%.c=%.o)
TARGETS = main
.PHONY: clean
all: $(TARGETS)
main: $(OBJS)
$(CC) $(OBJS) -o main
%.o: %.c
$(CC) -Wall -c $< -o $@
clean:
rm -rf $(OBJS) *~ $(TARGETS)
```

Sample Input

```
O..>.O...<..O 5
O..>.<..O 3
O..<.O 3
```

Output for Sample Input

```
YES
NO
YES
```

Solution

```
#include <string.h>
int foo(char *line, int sec)
{
    int i,j,hole=0,c=0;
    for(i=0; i<strlen(line); i++)
    {
        hole=0;
```

```
        if(line[i]=='>')
            for(j=1; j<=sec; j++)
            {
                if(line[i+j]=='O')
                    hole++;
            }
        if(line[i]=='<')
            for(j=1; j<=sec; j++)
            {
                if(line[i-j]=='O')
                    hole++;
            }

        if((line[i]=='>') || (line[i]=='<'))
            if(hole==0)
            {
                c++;
                break;
            }
    }
    if(c==0)
        return 1;
    else
        return 0;

}
```

https://progcont.hu/progcont/100196/?pid=201028

36. The Turtle and the Snail

The turtle and the snail are playing the following game: they stand at the two ends of a small clearing, facing each other. The midpoint of the imaginary straight line joining their positions is designated as the goal, and for a given signal, they begin to run towards that point.

Write a program that processes the data of such a race. The first line of the input contains a positive integer (*n*) that specifies the number of minutes that this noble competition lasted. Each of the next two lines contains *n* nonnegative integers. The numbers in a line are separated by exactly one space character. The first *n* numbers specify how many centimeters the turtle travelled each minute during the race, and the second *n* numbers give the same information for the snail. The race ends after *n* minutes, then one (or both) of them reaches the goal.

Your program should write in the first line of the standard output the total length of the race track in centimeters. In the second line, depending on which competitor has reached the midpoint, write the word "`TURTLE`", "`SNAIL`", or "`DRAW`".

Sample Input 1

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

Output for Sample Input 1

```
12
```

```
TURTLE
```

Sample Input 2

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

Output for Sample Input 2

```
12
DRAW
```

Sample Input 3

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

Output for Sample Input 3

```
12
SNAIL
```

Solution

```
#include <stdio.h>
int main() {
    int minute, number, i, turtle=0, snail=0;
    scanf("%d",&minute);
    for(i=0;i<minute;i++)
    {
        scanf("%d",&number);
        turtle+=number;
    }
    for(i=0;i<minute;i++)
    {
        scanf("%d",&number);
        snail+=number;
    }
    if(turtle==snail)
    {
        printf("%d\n",snail+turtle);
        printf("DRAW\n");
    }

    else if(turtle>snail)
    {
        printf("%d\n",turtle*2);
        printf("TURTLE\n");
    }
    else
    {
        printf("%d\n",snail*2);
        printf("SNAIL\n");
    }
    return 0;
```

```
}
```

https://progcont.hu/progcont/100161/?pid=200568

37. Super Stars of the Majors

Write a program that writes to the standard output the data of the three best-performing students of a particular major or all of them if there are at most three students in the given major. The data of the students should be presented in the standard output in descending order of their study average. Students having the same study average should be listed in lexicographical order of their Neptun IDs. The study averages should be displayed with one decimal precision.

The data to be processed must be read from the standard input. The first line of the input contains an acronym of a major made up of at most 10 characters. An additional up to 1000 lines of the input are of the following format:

surname first_name neptun_id major study_average

The *surname* and *first_name* are strings of up to 30 characters in length, the *neptun_id* is exactly 6 characters long, the *major* is an acronym of up to 10 characters, and the *study_average* is a real number. None of the strings contain space characters.

Sample Input

```
PTI
Orchidea Dora 123ABC GI 4.5
Ugyes Mark AFG12L GI 4.4
Horvath Csaba ART12M PTI 2.8
Mekk Elek ERTZ12 MI 2.7
Feher Adam ZTI98M GI 3.4
Keves Gabor PRE74W PTI 4.0
Pato Pal QWER99 MI 2.6
Feher Laszlo ASDF89 GI 2.5
Kiss Mark ACDC12 MI 3.8
Elo Arpad XYC78L PTI 4.9
Kemeny Szilard LUK65E PTI 4.0
Kantor Norbert WASD33 MI 3.8
```

Output for Sample Input

```
Elo Arpad XYC78L 4.9
Kemeny Szilard LUK65E 4.0
Keves Gabor PRE74W 4.0
```

Solution

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

typedef struct super
{
    char sname[35];
    char fname[35];
    char code[10];
```

```
    char major[12];
    float average;
} sz;

int compar(const void *a, const void *b)
{
    sz *x = (sz *) a;
    sz *y = (sz *) b;

    if (x->average < y->average)
        return 1;

    if (x->average > y->average)
        return -1;

    return strcmp (x->code, y->code);

}

int main()
{
    struct super *data;
    data=(struct super*)malloc(sizeof(struct super)*1000);
    int i = 0, count = 0,ki=0;
    char akt[10];

    scanf ("%s", akt);
    while (scanf("%s %s %s %s %f", data[i].sname, data[i].fname,
data[i].code,data[i].major, &data[i].average) > -1)
    {
        i++;
        count++;
    }

    qsort (data, count, sizeof (struct super), compar);

    if (count<=3)
    {
        for (i = 0; i < count; i++)
            printf ("%s %s %s %.1f\n", data[i].sname, data[i].fname,
data[i].code, data[i].average);
    }
    else
        for (i = 0; i < count; i++)
        {
            if (strcmp(akt, data[i].major)==0 && ki<3)
            {
                printf ("%s %s %s %.1f\n", data[i].sname,
data[i].fname, data[i].code, data[i].average);
                ki++;
            }
        }
    free(data);
    return 0;
}
```

https://progcont.hu/progcont/100196/?pid=201030

38. Prime or Not?

Write a program that helps you determine whether the elements of a sequence of numbers are all prime numbers. The input consists of several lines. Each line contains integers; if there are more numbers, they will be separated from each other by exactly one space character within the line. The first number of each line (*n*) specifies the number of numbers to be examined in that line. For each line, your program must determine whether the additional *n* numbers are all prime numbers. The end of the input is indicated by a line where *n* = 0. As a hint, we mention that the smallest prime number is 2.

For each line, your program should write "`YES`" to the standard output if all numbers in the line are prime numbers, or "`NO`" if there is at least one number among the numbers that is not prime.

Sample Input

```
1 1
1 2
5 2 3 5 7 11
5 2 3 4 5 7
5 3 11 5 7 2
5 11 7 3 9 5
0
```

Output for Sample Input

```
NO
YES
YES
NO
YES
NO
```

Solution

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
int prime(int n)
{
    int i;
    if (n<=1)
        return 1;
    for(i=2; i<=sqrt(n); i++)
        if(n%i==0)
            return 1;
    return 0;
}

int main()
{
    int number, i;
    while(scanf("%d",&number) && number!=0)
    {
```

```
        int a[number], count=0;
        for(i=0; i<number; i++){
            scanf("%d",&a[i]);
            if(prime(a[i])==0)
                count++;
                }

        if (number==count)
            printf("YES\n");
        else
            printf("NO\n");
    }

    return 0;
}
```

https://progcont.hu/progcont/100161/?pid=200595

39. Temperature Fluctuation

Write a program that reads lines in the following format until end-of-file from the text file given as the first command line argument:

settlement_name: *temperature*,*temperature*[,*temperature*]…

settlement_name is a string of up to 30 characters, the temperature data are signed integers, and the total length of the line does not exceed 1000 characters.

Each line contains the temperature data for a given settlement in a certain time interval. We do not know the exact number of measurements, but we know that at least two but no more than five hundred measurements are made during the time interval.

For each settlement, the program should write to the standard output the temperature fluctuation during the time interval, i.e., the difference between the largest and the smallest measured temperature.

The format of each line in the output should be as follows:

settlement_name: *temperature_fluctuation*

Sample Input

```
Debrecen: 10,8,8,8,9,9,11,14,14,15,17,16,16,14,15,17,19,17,18,14
Hatvan: 18,3
Funchal: 18,18,18,18,18,18,18,18,18
Liverpool: 14,15,16,17,16,17,13,10,12,4,2,3,5,12,17
Murmanszk: 1,0,-1,-3,-3,-2,0,1,2
```

Output for Sample Input

```
Debrecen: 11
Hatvan: 15
Funchal: 0
Liverpool: 15
Murmanszk: 5
```

Solution

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main(int argc, char *argv[])
{
    FILE *f=fopen(argv[1], "r");
    char line[1000], name[35], *token;
    int max, min, n, diff;

    while(fgets(line, 1000,f))
    {
        token=strtok(line,":");
        strcpy(name,token);
        token=strtok(NULL, ",");
        max=atoi(token);
        min=atoi(token);
        while(token=strtok(NULL, ","))
        {
            n=atoi(token);
            if(n>max)
                max=n;
            if(n<min)
                min=n;
        }
        diff=max-min;
        printf("%s: %d\n", name,diff);

    }
    fclose(f);
    return 0;
}
```

https://progcont.hu/progcont/100115/?pid=200755

40. Airplanes

Write a program that reads a list of airplane types from the text file given as the first command line argument, one per line. Each line has the following structure:

manufacturer;type;length

manufacturer and *type* are strings of up to 50 characters without spaces and semicolons, and *length* is a real number of up to 8 characters. The program should write to the standard output those manufacturers and types, separated by a space, the length of which is at least the value specified in the second command line argument.

(Hint: see `atof()` or `sscanf()` functions for converting a string to a real number.)

Sample Command Line Arguments

```
prog airplanes.txt 50
```

Sample `airplanes.txt`

```
Airbus;A320;37.57
Airbus;A380;72.73
Boeing;737-800;39.5
Boeing;747-8;76.3
```

Output for Sample Input

```
    Airbus A380
    Boeing 747-8
```

Solution

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

int main( int argc, char *argv[] )
{
  FILE *f = fopen( argv[1], "r" );
  char line[ 200 ];
  double ertek = atof( argv[ 2 ] );
  while ( fgets( line, 200, f ) != NULL )
  {
    char *manufacturer = strtok( line, ";" );
    char *type = strtok( NULL, ";" );
    double length = atof( strtok( NULL, "\n" ) );
    if ( length >= ertek )
      printf( "%s %s\n", manufacturer, type );
  }
  fclose( f );
  return 0;
}
```

https://progcont.hu/progcont/100116/?pid=200759

41. High-Density Countries

Write a program that reads data of countries from the input file `bemenet.txt` until end-of-file, one country per line. Each line has the following structure:

name`;`*area*`;`*population*[`;`*population*]…

name is a string with no more than 30 characters, containing no semicolons, *area* and *population* are positive integers. The total population of the country is the sum of the *population* values in the line. The file consists of up to 200 lines, each at most 1000 characters long.

The program should write to the standard output the name, area, and total population of those countries whose population density (population to area ratio) exceeds the real value given in the first command line argument of the program, one country per line. Two consecutive data values in a line should be separated by a comma.

Sort the output in descending order of population densities. Countries with the same population density should be sorted in ascending order of country names.

(Hint: see `atoi()`, `atof()`, or `sscanf()` functions for converting a string to an integer or a real number, see the `strtok()` function for tokenizing a string, and see the `qsort()` function for sorting).

Sample Command Line Arguments

```
    prog 100
```

Sample `bemenet.txt`

```
    Magyarorszag;93030;4851354;5042545
    Amerikai Egyesult
Allamok;9826630;10000000;20000000;30000000;40000000;50000000;6000000
0;70000000;38697314
    Nemetorszag;357023;80500000
    Kanada;9984670;10000000;20000000;5702707
    Egyesult Kiralysag;244820;30000000;33705000
```

Output for Sample Input

```
    Egyesult Kiralysag,244820,63705000
    Nemetorszag,357023,80500000
    Magyarorszag,93030,9893899
```

Solution

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
typedef struct
{
    char name[31];
    int area;
    int population;
    float popular;
} DATA;

int compar(const void *a, const void *b)
{
    DATA *x=(DATA*)a;
    DATA *y=(DATA*)b;

    if(x->popular>y->popular)
        return -1;
    if(x->popular<y->popular)
        return 1;
    return strcmp(x->name,y->name);
}

int main(int argc, char *argv[])
{
    FILE *f = fopen("bemenet.txt","r");
    float a = atof(argv[1]);
    DATA t[200];
    char line[1000], *token;
    int index=0, i;
    while(fgets(line,1002,f))
    {
        int szum=0;
        t[index].population=0;
        token=strtok(line,";");
```

```
            strcpy(t[index].name,token);
            token=strtok(NULL,";");
            t[index].area=atoi(token);
            while(token=strtok(NULL,";"))
            {
                szum+=atoi(token);
            }
            t[index].population=szum;
            t[index].popular=(float)t[index].population/t[index].area;
            if (t[index].popular>a)
                index++;
        }
        qsort(t,index,sizeof(DATA),compar);
        for(i=0; i<index; i++)
        {
            printf("%s,%d,%d\n",t[i].name,t[i].area,t[i].population);
        }
        fclose(f);
        return 0;
}
```

https://progcont.hu/progcont/100248/?pid=201207

42. Drone Images

Many hikers are walking in the hiking trails of Hungary even in rainy weather. In this case, we can spot two things in an image of the pathway created by a drone: people and puddles. Given the current situation of the hikers and knowing that all hikers want to keep clear of the puddles, it is easy to determine the position of the hikers in the moment after the next step.

Hikers go from left to right on the pathways, and in the moment after the next step, they all move to the next place where there is no puddle. Thus, there will be hikers who take only a small step to the next adjacent section of the pathway, but there will be some who have to jump to the section of the pathway that follows the puddles in front of them.

Consider the following header file:

myheader.h

```
    #ifndef _MYHEADER_H
    #define _MYHEADER_H 1

    void foo(const char *, char *);

    #endif /* myheader.h */
```

Write the function `foo()` declared in `myheader.h` that takes the address of two strings as parameters. The first string describes a pathway, in which an asterisk character ('`*`') denotes a hiker, an at sign ('`@`') denotes a puddle, and an equal sign ('`=`') indicates a free passage of the pathway.

The `foo()` function should compute the position of the hikers in the next moment and store it in a string of the same format as the input. You may assume that no one in the starting position has reached the goal of the tour, nor has the target flag been set in a puddle, so there will be at least one free passage in front of each participant.

Note

Place the function in file `foo.c` and submit this file as a solution to the evaluation system. You can test your solution using the following files. The evaluation system does not necessarily perform the evaluation using these files.

main.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "myheader.h"

int main()
{
    char source[100], destination[100];
    while (fgets(source, 100, stdin) != NULL)
    {
        if (source[strlen(source) - 1] == '\n')
            source[strlen(source) - 1] = '\0';
        foo(source, destination);
        printf("%s\n", destination);
    }
    return EXIT_SUCCESS;
}
```

Makefile

```
SRCS = main.c foo.c
OBJS = $(SRCS:%.c=%.o)
TARGETS = main
.PHONY: clean
all: $(TARGETS)
main: $(OBJS)
$(CC) $(OBJS) -o main
%.o: %.c
$(CC) -Wall -c $< -o $@
clean:
rm -rf $(OBJS) *~ $(TARGETS)
```

Sample Input

```
==*====*====
===@===
**@@@==*@@=*@*=
*=*=*=*=*=*=
==*@===*=*@=
```

Output for Sample Input

```
===*====*===
===@===
=*@@@*==@@*=@**
=*=*=*=*=*=*
===@*===*=@*
```

Solution

```
void foo(char source[], char destination[])
{
    int i,j;
    strcpy(destination,source);
    for(i=strlen(destination)-1; i>=0; i--)
    {
        if(destination[i]=='='&&destination[i-1]=='*')
        {
            destination[i]='*';
            destination[i-1]='=';
        }
        else if(destination[i]=='@'&&destination[i-1]=='*')
        {
            for(j=i+1; j<strlen(destination); j++)
            {
                if(destination[j]=='=')
                {
                    destination[j]='*';
                    destination[i-1]='=';
                    break;
                }
            }
        }
    }
}
```

https://progcont.hu/progcont/100248/?pid=201210

43. Newspapers

Write a program that reads from the standard input names of newspapers of up to 20 characters, containing no spaces, as well as a string of exactly 7 characters, until end-of-file (EOF), one newspaper per line. The latter string contains only minus ('−') and plus ('+') characters. The 7 characters indicate the appearance of the newspaper on the seven days of the week: the first character represents Monday, the second Tuesday, the third Wednesday, the fourth Thursday, the fifth Friday, the sixth Saturday, and the seventh Sunday. A minus sign indicates that the newspaper is not published on that day, the plus sign indicates that it is.

Your program should write to the the standard output those days when no newspaper is published. If there is more than one day when no newspaper is published, these days should appear in the traditional Hungarian order of weekdays, but with their English names ("Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday").

Sample Input

```
Blikk -+-+-+-
Fules +------
Nepszava +------+
Metro ++-+-++
```

Output for Sample Input

```
Wednesday
```

```
    Friday
```

Solution

```
#include<stdio.h>
int main()
{
    char name[20],week[7];
    int w[7]= {0,0,0,0,0,0,0},i;
    while(scanf("%s %s",name,week)!=EOF)
    {
        for(i=0; i<7; i++)
        {
            if(week[i]=='+')
                w[i]=1;
        }
    }
    if(w[0]==0)
        printf("Monday\n");
    if(w[1]==0)
        printf("Tuesday\n");
    if(w[2]==0)
        printf("Wednesday\n");
    if(w[3]==0)
        printf("Thursday\n");
    if(w[4]==0)
        printf("Friday\n");
    if(w[5]==0)
        printf("Saturday\n");
    if(w[6]==0)
        printf("Sunday\n");
    return 0;
}
```

https://progcont.hu/progcont/100214/?pid=201085

44. Class Attendance

Consider the following header file:

myheader.h

```
    #ifndef _MYHEADER_H
    #define _MYHEADER_H 1

    int foo(char *);

    #endif /* myheader.h */
```

Write the function `foo()` declared in `myheader.h` that takes a string as a parameter, consisting only of plus ('+') and minus ('−') characters, and describes a student's attendance in the semester classes: plus signs indicate presence, minus signs indicate absence. The function should determine and return the number of times the student missed class.

Note

Place the function in file `foo.c` and submit this file as a solution to the evaluation system. You can test your solution using the following files. The evaluation system does not necessarily perform the evaluation using these files.

main.c

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

int main()
{
    printf("%d\n", foo("++---+-+--+++-"));
    return EXIT_SUCCESS;
}
```

Makefile

```
SRCS = main.c foo.c
OBJS = $(SRCS:%.c=%.o)
TARGETS = main
.PHONY: clean
all: $(TARGETS)
main: $(OBJS)
$(CC) $(OBJS) -o main
%.o: %.c
$(CC) -Wall -c $< -o $@
clean:
rm -rf $(OBJS) *~ $(TARGETS)
```

Output for Sample Input

```
7
```

Solution

```
#include<string.h>

int foo(char *input){

    int count=0, i;

    for(i=0;i<strlen(input);i++)
        if(input[i]=='-')
            count++;
    return count;
}
```

https://progcont.hu/progcont/100214/?pid=201086

45. Over Hill and Valley

A tourist hiked through his favorite mountain range, Mátra, over the weekend. Several small settlements were involved in his journey. At the start and every time he arrived at a

settlement, he made a note of the number of kilometers he had walked since the last settlement and the population of the settlement he just reached, i.e., the number of people living there.

At the end of his tour, he was curious about the total distance he had taken and the difference between the number of inhabitants of the most populated settlement and the least populated settlement on his trip. Write a program that will make the calculations you need for this.

The input consists of at least two lines, each with two integers: the distance from the last settlement (in meters) and the number of inhabitants of the settlement reached. Your program must process the lines until end-of-file (EOF).

The program should write one line with two integers separated by a space character to the standard output: the length of the trip traveled by the tourist (in meters) and the difference between the number of inhabitants of the most populated settlement and the least populated settlement on his trip. Remember to terminate this line with a newline character.

Sample Input

```
0 32056
3628 617
6845 1068
1687 301
2598 54
3712 591
10025 112
3045 2513
2174 48
```

Output for Sample Input

```
33714 32008
```

Solution

```
#include <stdio.h>

int main()

{

    int dist, inhabitants ,max=0, min=0, distance=0, first_value=0;

    while(scanf("%d %d",&dist,&inhabitants )!=EOF)

    {

        if (first_value==0)

        {

        min = inhabitants;

        max = inhabitants;

        first_value=1;
```

```
        }
        distance+=dist;
        if(inhabitants >  max)
            max = inhabitants;
        if(inhabitants <min)
            min = inhabitants;
    }
    printf("%d %d\n", distance, max-min);
    return 0;
}
```

https://progcont.hu/progcont/100214/?pid=201087

46. Mastermind

In Mastermind, the player has to find a sequence of colored balls of a certain length in the least possible number of attempts. They always guess the entire ball sequence by giving a sequence of colors. After each guess, a feedback is given to them about the number of balls that are correct in both color and position, and the number of balls of a correct color in the wrong position.

For example, if the ball sequence to be found is blue-green-red-yellow, and the player guesses the sequence brown-green-blue-yellow, then two balls are correct in color and position, and one is correct in color but not in position. After the guess, this information is shared with the player.

The basic game is played with four balls, but there are many variations of the game, which can be obtained by changing the length of the ball sequence.

Write a program that reads two strings of the same length from each line of the standard input, each containing only uppercase letters of the English alphabet, up to 100 characters, until end-of-file (EOF): the first contains letters indicating the colors of the sequence to be found, and the second is the player's guess for this sequence. You may assume that each color is represented by a unique letter. For each line of the input, write two integers to the standard output based on the rules of the Mastermind game: first the number of balls that are correct in both color and position, then the number of balls of a correct color in the wrong position. In each line, the two numbers should be separated by a single space character.

Sample Input

```
KZPS BZKS
ABCDEF ABCDEF
FEDCBA ABCDEF
ABCD AAAA
ABCD XYAA
XYAA ABCD
AABBCC ABCABC
CBACBA ABCABC
ABCABC AACCBB
```

Output for Sample Input

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

Solution

```
#include <stdio.h>
#include <string.h>
int main()
{
    char correct[100], guess[100];
    int i, j, count1, count2;
    while(scanf("%s %s",correct,guess)!=EOF)
    {
        count1=count2=0;

        for(i=0; i<strlen(correct); i++)
        {
            if(correct[i]==guess[i])
            {
                count2++;
                correct[i]='0';
                guess[i]='1';
            }
        }

        for(i=0; i<strlen(correct); i++)
        {
            for(j=0; j<strlen(guess); j++)
            {
                if(correct[i]==guess[j])
                {
                    count1++;
                    correct[i]='0';
                    guess[j]='1';
                }
            }
        }

        printf("%d %d\n",count2,count1);
    }
    return 0;
}
```

https://progcont.hu/progcont/100214/?pid=201088

47. Divisors

Write a program that reads positive integers from each line of the standard input until end-of-file (EOF). The first number of each line determines how many additional numbers are given in that line.

For each line, the program should sort the numbers in ascending order of the (positive) number of their divisors. If two or more numbers have the same number of divisors, sort them in ascending order of magnitude.

Write each ordered sequence of numbers in a separate line, separating the elements of a sequence from each other with a single space character.

Sample Input

```
    10 1 2 3 4 5 6 7 8 9 10
    10 11 12 13 14 15 16 17 18 19 20
```

Output for Sample Input

```
    1 2 3 5 7 4 9 6 8 10
    11 13 17 19 14 15 16 12 18 20
```

Solution 1

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

int divisors(int n)
{
    int i, div=0;
    for (i=1; i<=n; i++)
        if (n%i==0)
            div++;
    return div;
}

int compar(const void *a1, const void *b1)
{
    int a = *(int *)a1;
    int b = *(int *)b1;
    if (divisors(a) < divisors(b))
        return -1;
    else if (divisors(a) > divisors(b))
        return 1;
    else
    {
        if (a< b)
            return -1;
        else if (a > b)
            return 1;
        else
            return 0;
    }
}
```

```
int main()
{
    int n, i;
    while (scanf("%d", &n)!=EOF)
    {
        int a[n];
        for (i=0; i<n; i++)
            scanf("%d", &a[i]);

        qsort(a, n, sizeof(int), compar);

        for (i=0; i<n; i++)
            if (i!=n-1)
                printf("%d ", a[i]);
            else
                printf("%d\n", a[i]);

    }
return 0;
}
```

Solution 2

```
#include<stdio.h>
void print_out(int *a,int n)
{
    int i;
    for(i=0; i<n-1; i++)
        printf("%d ",a[i]);
    printf("%d\n",a[i]);
}
void swap(int *a,int i,int j)
{
    int tmp;
    tmp=a[i];
    a[i]=a[j];
    a[j]=tmp;
}
int main()
{
    int n,i,j;
    while(scanf("%d",&n)!=EOF)
    {
        int a[n],b[n],sum=0, x;
        for(i=0; i<n; i++)
        {
            scanf("%d",&a[i]);
            sum=0;
            for(x=1; x<a[i]; x++)
                if(a[i]%x==0)
                    sum++;
            b[i]=sum;
        }
        for(i=n-1; i>0; i--)
```

```
            for(j=0; j<i; j++)
                if((b[j]>b[j+1])||((b[j]==b[j+1])&&(a[j]>a[j+1])))
                {
                    swap(a,j,j+1);
                    swap(b,j,j+1);
                }
        print_out(a,n);
    }
    return 0;
}
```

https://progcont.hu/progcont/100196/?pid=201027

48. At the Beginning of the Alphabet

Consider the following header file:

myheader.h

```
#ifndef _MYHEADER_H
#define _MYHEADER_H 1

char foo(const char *);

#endif /* myheader.h */
```

Write the function `foo()` declared in `myheader.h` that takes a string as a parameter containing only uppercase letters of the English alphabet. The function should determine and return the letter of the string that is the first in alphabetical order. Make sure that the original string does not change.

Note

Place the function in file `foo.c` and submit this file as a solution to the evaluation system. You can test your solution using the following files. The evaluation system does not necessarily perform the evaluation using these files.

main.c

```
#include <stdio.h>
#include <stdlib.h>
#include "myheader.h"
int main()
{
    char line[100];
    while (gets(line) != NULL)
        printf("%c\n", foo(line));
    return EXIT_SUCCESS;
}
```

Makefile

```
SRCS = main.c foo.c
OBJS = $(SRCS:%.c=%.o)
TARGETS = main
```

```
.PHONY: clean
all: $(TARGETS)
main: $(OBJS)
$(CC) $(OBJS) -o main
%.o: %.c
$(CC) -Wall -c $< -o $@
clean:
rm -rf $(OBJS) *~ $(TARGETS)
```

Sample Input

```
ALMA
MEGGY
MOTOR
```

Output for Sample Input

```
A
E
M
```

Solution

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

char foo(const char *s)
{

   char min=s[0];
   for (size_t i = 1; i < strlen(s); i++)
   {
       if(s[i]<min)min=s[i];
   }

  return min;
}
```