Formatted input / output : printf / scanf

Input / output functions `printf` / `scanf` are declared in

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

- Function `printf` prints the formatted output to ***stdout***;
- Function `scanf` reads the formatted input from ***stdin***;

TYPE DESCRIPTION	TYPE	FORMAT
integer, 4 bytes	int	%d
integer, 8 bytes	__int64	%I64d
integer, 8 bytes	long long	%lld
real, 4 bytes	float	%f
real, 8 bytes	double	%lf
character, 1 byte	char	%c
string, array of chars	char[]	%s

Let $x = 34$ is an integer. To print the value of the variable, we use the function

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

If we want to print the value in n positions, we use format %nd. If the number contains less than n digits, the spaces will be added befor the number. Next example prints 3 spaces before number 34 (we print 5 characters in total):

```
printf("%5d",x);
```

Formatting with spaces can be used for example in printing the multiplication table. Each number is printed in 2 positions. If number is one-digit, a space is printed before it.

```
#include <stdio.h>

int i, j;

int main(void)
{
  for(i = 1; i < 10; i++)
  {
    for(j = 1; j < 10; j++)
      printf("%2d ",i*j);
    printf("\n");
  }
  return 0;
}
```

E-OLYMP **8545. Multiplication table** Print multiplication table $n * n$ with alignment.

► Use a double loop to print the multiplication table. For alignment, each number should be printed in two positions, use the format %2d.

Printing the time. Let *h* contains the hours, *m* contains the minutes and *s* contains the seconds. We want to print the time in digital format like "12:45:23", printing each number in 2 positions. But if some number is less than 10, using the format `%2d` we'll get something like " 2: 5: 0" (2 hours, 5 minutes and 0 seconds). In a real digital clock we must print 0 instead of spaces: "02:05:00". To solve this problem we must use format `%02d`. Zero symbol before 2 means that instead of spaces at extra positions we must print zeroes.

```
#include <stdio.h>

int h = 2, m = 5, s = 0;

int main(void)
{
  printf("%02d:%02d:%02d\n",h,m,s);
  return 0;
}
```

If we want to read the data in digital clock format, we can use the format

```
scanf("%d:%d:%d",&h,&m,&s);
```

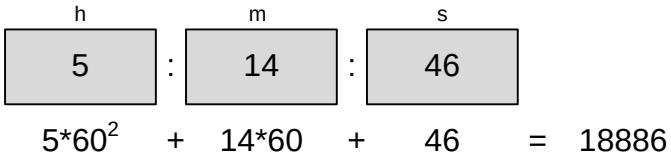
E-OLYMP 514. Time for Nicholas Find the difference between the times t_1 and t_2.

► Convert the start and end times to seconds. The time *hh* : *mm* : *ss* corresponds to 3600 * *hh* + 60 * *mm* + *ss* seconds.

Compute the difference $t_2 - t_1$ between them. If it is negative, then midnight belongs to the time when Nikolas delivers gifts. In this case, 3600 * 24 should be added to the negative difference – the number of seconds in a day. Next, compute how many hours, minutes and seconds the found difference is.

Let *d* be the time of gifts delivery in seconds. Convert it to hours *h*, minutes *m* and seconds *s* is the same as to represent the number *d* in sexagesimal notation:

- *h* = *d* / 3600;
- *m* = (*d* % 3600) / 60;
- *s* = *d* % 60;



For *d* = 18886 we have:

- *h* = 18886 / 3600 = 5;
- *m* = (18886 % 3600) / 60 = 14;
- *s* = 18886 % 60 = 46;

Single and multiple input / output

Solving the problems, one must distinguish the single input and multiple input. If the input contains only one test, we speak about "single input". If the input contains data for some tests, we say then the problem has "multiple input". Consider the examples.

Example. Two numbers are given. Find their sum.

Sample input	Sample output
3 4	7

Solution. To solve this problem, its enough to input two numbers, find their sum and output the result.

```
#include <stdio.h>

int a, b, res;

int main(void)
{
  scanf("%d %d",&a,&b);
  res = a + b;
  printf("%d + %d = %d\n", a, b, res);
  return 0;
}
```

E-OLYMP **518. Sum of two** Find the sum of two numbers. The first line contains number of test cases t ($1 \le t \le 100$). Each test consists of two integers a and b.

Sample input	Sample output
3	6
4 2	3
1 2	15
7 8	

► The input pair a and b does not have to be on the same line. Read the number of test cases t. Process the tests sequentially: read a couple of numbers and print their sum.

```
#include <stdio.h>

int i, t, a, b;

int main(void)
{
  scanf("%d", &t);
  for (i = 0; i < t; i++)
  {
    scanf("%d %d", &a, &b);
    printf("%d\n", a + b);
  }
```

```
  return 0;
}
```

The loop can be formed using the ***while*** operator. In this case we do not need to use an additional variable *i*. The ***while*** loop continues executing as long as the expression *t*-- stays true. And it stays true until *n* is not zero.

```
#include <stdio.h>

int t, a, b;

int main(void)
{
  scanf("%d", &t);
  while (t--)
  {
    scanf("%d %d", &a, &b);
    printf("%d\n", a + b);
  }
  return 0;
}
```

E-OLYMP **1000. a + b problem** Find the value of $a + b$. Each line contains two integers a and b ($|a|, |b| \leq 30000$).

Sample input	Sample output
4 2	6
1 2	3
7 8	15

► This problem statement differs from the previous because here we do not know the number of test cases. So we must read input data until the end of file. Writing program in C, we do not need to use the file operations.

The function ***scanf*** does not only read the data, but also returns the number of arguments that has been read. So if to run the expression

```
i = scanf("%d %d",&a,&b);
```

and to enter two numbers, the variable *i* will be assigned the value 2. This property of ***scanf*** function is convenient to use while reading data till the end of file. If the program read all the data and reached the end of file, in the next call ***scanf*** returns -1.

```
#include <stdio.h>

int a, b;

int main(void)
{
  while(scanf("%d %d",&a,&b) == 2)
    printf("%d\n",a + b);
```

```
  return 0;
}
```

In the ***while*** loop we read two numbers *a* and *b*. While we do not reach the end of file, ***scanf*** returns 2 and the body of the loop is executed (the sum of the numbers is printed). When the end of file is reached, the ***scanf*** can't read more data and returns -1. The loop stops.

Remember! If you read data from the console, it is possible to enter the symbol "end of file" pressing the keys ^Z.

Example. The input consists of multiple lines. Each line contains two nonnegative integers *a* and *b*. For each input line print the sum of its numbers. The last line contains two zeroes and must not be processed.

Sample input	Sample output
4 2	6
1 2	3
7 8	15
0 0	

Solution. This example differs from the previous because here we must process the data not till the end of file, but till the values $a = 0$, $b = 0$.

```
#include <stdio.h>

int a, b;

int main(void)
{
  while(scanf("%d %d", &a, &b), a + b)
    printf("%d\n", a + b);
  return 0;
}
```

The conditional expression of the loop ***while*** consists of two parts: the function ***scanf*** and the expression $a + b$. The loop continues until both expressions are true. It is obvious that ***scanf*** always returns 2 (because in this example we do not reach the end of file), and the value $a + b$ stays true until both variables *a* and *b* are not zero (by the condition of the problem *a* and *b* are nonnegative integers).

Remember! The arithmetic expression is ***true*** if it is not equal to 0.

E-OLYMP 520. The sum of all Find the sum of all given numbers.

► Since $n \le 10^5$, and each number does not exceed 10^9 in absolute value, then the sum of the initial numbers can be about 10^{14}. To compute the result, use ***long long*** type.

Read the input data till the end of file. Sum up the given numbers and print the result.