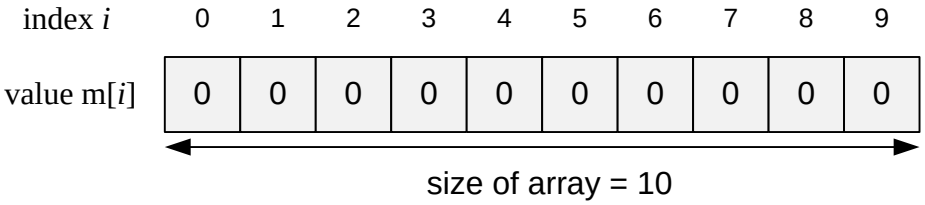
Arrays

An array is a series of elements of the same type placed in contiguous memory locations that can be individually referenced by adding an index to a unique identifier.

For example, an array containing 10 integer values of type ***int*** called ***m*** could be represented as:



Each cell represents an element of array. The elements of type ***int*** are numbered from 0 to 9, being 0 the first and 9 the last. In C, the first element in an array is always numbered with a ***zero*** (not a one), no matter its length.

Like a regular variable, an array must be declared before it is used. A typical declaration for an array in C++ is:

```
type name [elements];
```

where ***type*** is a valid type (such as int, float...), ***name*** is a valid identifier and the ***elements*** field (which is always enclosed in square brackets []), specifies the length of the array in terms of the number of elements.

Therefore, the m array, with ten elements of type int, can be declared as:

```
int m[10];
```

Accessing the values of an array

The values of any of the elements in an array can be accessed just like the value of a regular variable of the same type. The syntax is:

```
name[index]
```

For example, the following statement stores the value 34 in the third element of m:

```
m[2] = 34;
```

and, for example, the following copies the value of the third element of m to a variable called *x*:

```
x = m[2];
```

Notice that the third element of m is specified m[2], since the first one is m[0], the second one is m[1], and therefore, the third one is m[2]. By this same reason, its last element is m[9]. Therefore, if we write m[10], we would be accessing the eleventh element of m, and therefore actually exceeding the size of the array.

E-OLYMP 8953. Print array Array of *n* integers is given. Print all its elements in column, do not change their initial order.

► The problem can be solved using a ***loop***. Read and immediately print the elements of the array, one per line. But we'll solve the problem with an ***array***.

```
#include <stdio.h>

int a, i, n;
int m[101];

int main(void)
{
  // read the value of n, the size of array
  scanf("%d", &n);
  // read the array itself
  for (i = 0; i < n; i++)
    scanf("%d", &m[i]);

  // print the elements of array in specified order
  for (i = 0; i < n; i++)
    printf("%d\n", m[i]);

  return 0;
}
```

E-OLYMP **7829. Sum of elements** Given sequence of *n* real numbers. Find the sum of all its elements. First line contains number *n*. Next line contains *n* real numbers. Print the sum of all sequence elements.

Sample input	**Sample output**
5 1.2 1.3 5.7 1.8 12.4	22.4

► Let's divide the solution of the problem into the next parts:

1. Read the sequence of *doubles* into array;
2. Find the sum of array elements using ***for*** loop;
3. Print the resulting sum.

```
#include <stdio.h>

int i, n;
double s, m[100];

int main(void)
{
  scanf("%d",&n);

  // read array
  for(i = 0; i < n; i++)
    scanf("%lf",&m[i]);

  // find the sum of array elements
  s = 0;
  for(i = 0; i < n; i++)
    s = s + m[i];
```

```
  // print the answer
  printf("%.1lf\n",s);
  return 0;
}
```

E-OLYMP 4730. Fibonacci Fibonacci numbers is a sequence of numbers F(n), given by the formula:

$$F(0) = 1, F(1) = 1, F(n) = F(n-1) + F(n-2)$$

Given the value of n. Print the n-th Fibonacci number.

Sample input	**Sample output**
4	5

► Let's declare array fib of size 45. Value of fib[i] will contain the i-th Fibonacci number, so fib[i] = F(i).

index i	0	1	2	3	4	5	6	7	8	9
fib[i]	1	1	2	3	5	8	13	21	34	55

To answer the question its enough to read n and print the value of fib[n].

```
#include <stdio.h>
#define MAX 46

int i, n;
int fib[MAX];

int main(void)
{
  // initialize the base case n = 0, n = 1
  fib[0] = 1; fib[1] = 1;

  // calculate the values of fib[i] = F(i). Preprocessing
  for(i = 2; i < MAX; i++)
    fib[i] = fib[i-1] + fib[i-2];

  // read the value of n and print F(n) = fib[n]
  scanf("%d",&n);
  printf("%d\n",fib[n]);
  return 0;
}
```

E-OLYMP 928. The sum of the largest and the smallest Find the sum of the smallest and the largest element in array.

Sample input	**Sample output**
4 1 2 3 4	5

► Find minimum and maximum in array. Print their sum.

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

```
int i, n, mn, mx;
int m[100];

int main(void)
{
  // read array
  scanf("%d", &n);
  for (i = 0; i < n; i++)
    scanf("%d", &m[i]);

  // find min and max in array
  mn = 100; mx = -100;
  for (i = 0; i < n; i++)
  {
    if (m[i] < mn) mn = m[i];
    if (m[i] > mx) mx = m[i];
  }

  // print the answer
  printf("%d\n", mn + mx);
  return 0;
}
```

E-OLYMP 7833. More than average Given array of *n* integers. Find the sum and the amount of numbers, greater than the arithmetic average of array elements.

Sample input	**Sample output**
5 1 6 2 6 3	12 2

► Find the sum *s* of all elements. The arithmetic average equals to *s* / *n*. Now we need to find the sum and amount of such m[*i*] that m[*i*] > *s* / *n*. To avoid integer division, we can rewrite an equality as m[*i*] * *n* > *s*.

```
#include <cstdio>
#include <algorithm>
using namespace std;

int i, s, n, sum, cnt;
int m[101];

int main(void)
{
  scanf("%d", &n);
  // read array, find the sum s of elements
  for (i = 0; i < n; i++)
  {
    scanf("%d", &m[i]);
    s += m[i];
  }

  // find the sum and amount of elements greater than average
  sum = cnt = 0;
  for (i = 0; i < n; i++)
```

```
    if (m[i] * n > s) // m[i] > s / n
    {
      sum += m[i];
      cnt++;
    }

  printf("%d %d\n", sum, cnt);
  return 0;
}
```

E-OLYMP **8963. Minimums to the left** Given array of n integers. Move all minimum elements to the beginning of the array without changing the order of others.

► Find the minimum element *min*. Move along the array from right to left and move non-minimal elements to the right. The part of the array that remains to the left is filled with the minimum element.

Consider an example given in problem statement.

Declare an array.

```
int m[101];
```

Read the input array. Find the minimum element *min*.

```
scanf("%d", &n);
min = 100;
for (i = 0; i < n; i++)
{
  scanf("%d", &m[i]);
  if (m[i] < min) min = m[i];
}
```

Declare two indices i and j. Move through the array from right to left. If m[j] is not equal to the minimum, then copy this number to m[i].

```
i = n - 1;
for (j = n - 1; j >= 0; j--)
  if (m[j] != min)
  {
    m[i] = m[j];
    i--;
  }
```

The rest of the array elements from index 0 to i should be filled with minimum element.

```
  while (i >= 0)
  {
    m[i] = min;
```

```
    i--;
  }
```

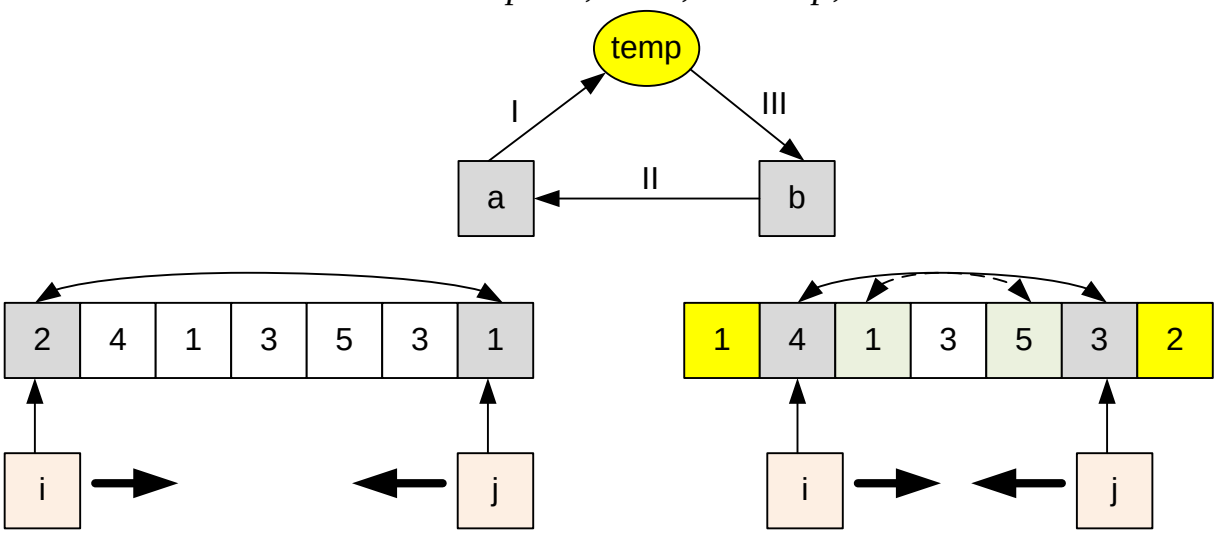
Print the resulting array.

```
for (i = 0; i < n; i++)
  printf("%d ", m[i]);
printf("\n");
```

E-OLYMP 2098. Invertor Print array elements in the reverse order.

Sample input	**Sample output**
7 2 4 1 3 5 3 1	1 3 5 3 1 4 2

► To swap the values of two variables *a* and *b* we need additional variable *temp*:
temp = *a*; *a* = *b*; *b* = *temp*;




```
#include <stdio.h>

int n, i, j, temp;
int m[110];

int main(void)
{
  // read array
  scanf("%d", &n);
  for (i = 1; i <= n; i++)
    scanf("%d", &m[i]);

  // reverse array
  i = 1; j = n;
  while (i < j)
  {
    // swap (m[i], m[j])
    temp = m[i]; m[i] = m[j]; m[j] = temp;
    i++; j--;
  }
```

```
  // print array
  for (i = 1; i <= n; i++)
    printf("%d ", m[i]);
  printf("\n");
  return 0;
}
```

E-OLYMP 4751. Diagonals Find the sum of elements on the main and the secondary diagonal.

Sample input

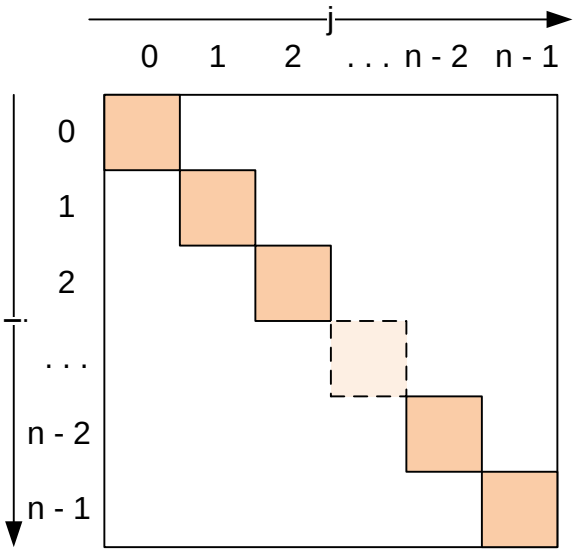
```
4
134 475 30 424
303 151 419 235
248 166 90 42
318 237 184 36
```

Sample output

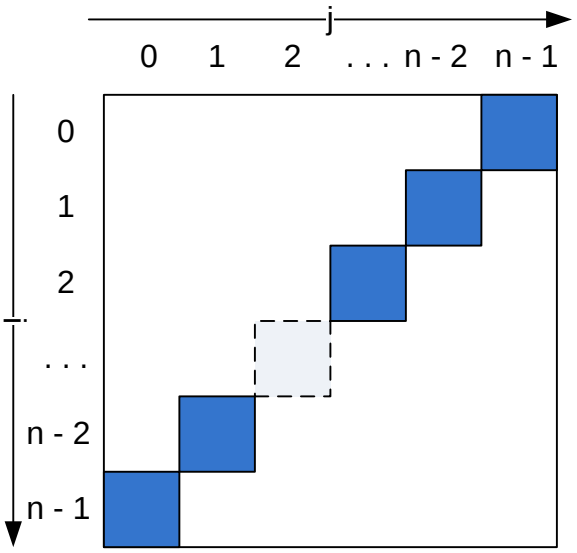
```
411 1327
```

► Let m[0..*n* – 1][0..*n* – 1] be two dimentional array. Element m[*i*][*j*] belongs to:

- main diagonal if *i* = *j*;
- secondary diagonal if *i* + *j* = *n* – 1;



i = j



i + j = n - 1

```
#include <stdio.h>

int i, j, n;
int a, b;
int m[510][510];

int main(void)
{
  // read two dimentional array
  scanf("%d", &n);
  for (i = 0; i < n; i++)
  for (j = 0; j < n; j++)
    scanf("%d", &m[i][j]);

  // find the sum of elements on the main and secondary diagonal
```

```
  a = b = 0;
  for (i = 0; i < n; i++)
  for (j = 0; j < n; j++)
  {
    if (i == j) a += m[i][j];
    if (i + j == n - 1) b += m[i][j];
  }

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