ARRAYS IN C/C++

array: **dane_liczbowe**

float **dane_liczbowe**[5];
dane_liczbowe[0]=12.5;
dane_liczbowe[1]=-0.2;
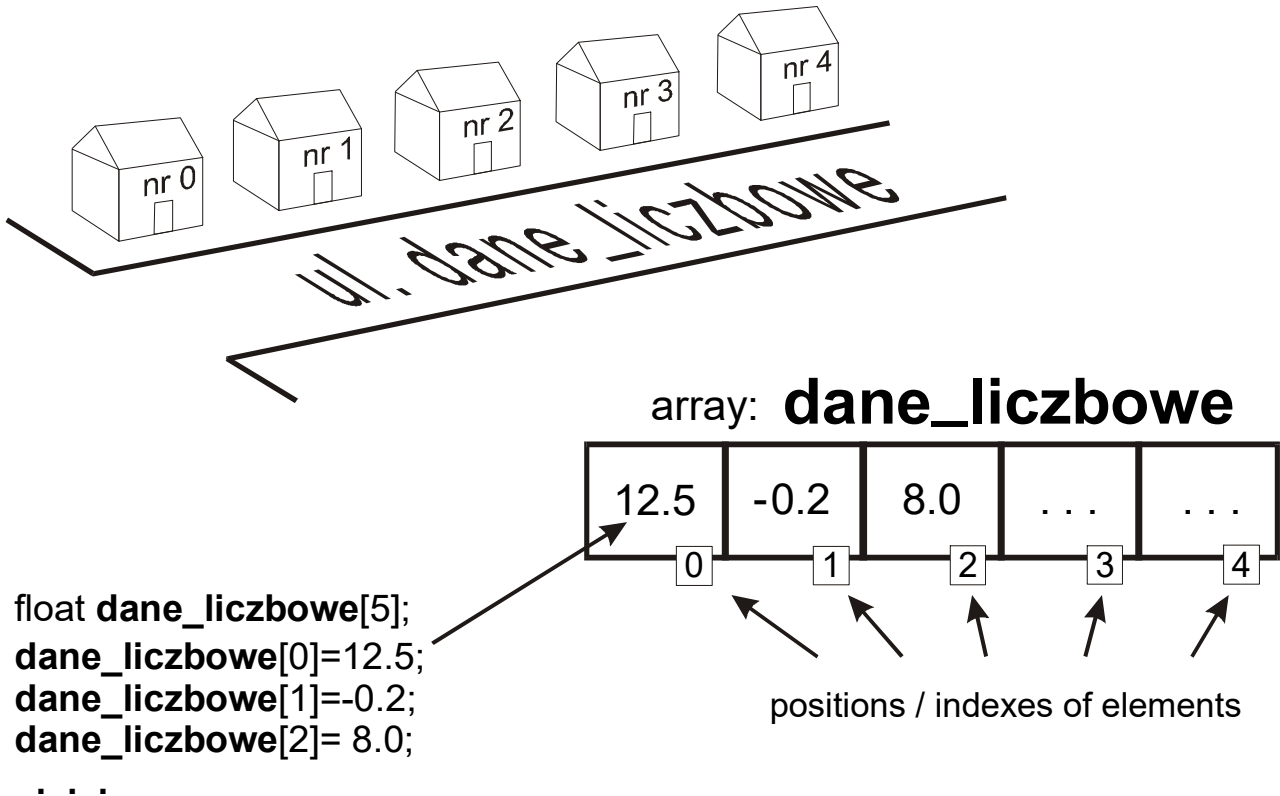**dane_liczbowe**[2]= 8.0;
 . . .

positions / indexes of elements

Array $\rightarrow$ is a representation of data that allows **grouping** of several items of **the same type** and referring to them using common name.
It is one of the most commonly used data types.

The general form of array definition:

> **type_of_element   arrayName [number_of_elements] ;**

examples:
  float numerical_data[5];   *// 5-element array of real numbers*
  int   tab[10];   *// 10-element array of integers*
  char text[255];   *// 255-element array of characters*

  double ($*$functions [20]) (**double**,**double**); *// array of 20 function pointers*

Notes on tables:

➤ An important feature of the table is a representation in the form of **contiguous region in memory** space and uniform distribution of subsequent components in immediate succession (directly one after the other)

      With such representation the computer can **quickly calculate the position** of an item in memory (based on its order number - index) and a significant reduction the code size, through the **use of loops**.

➤ Array elements are indexed starting **from zero !**

➤ C / C++ languages **do not check the range (correctness) of indexes!**

This is often the cause of nasty (difficult to detect) errors. For example|,
    assuming definition: `float numerical_data [5];`
           instruction: `numerical_data [5]=10.5;`
destroys / overwrites the memory **after** the last element of this array.

➤ The name of the array is, at the same time, the pointer the first element, ie.
        array_name == & array_name[0]

➤ Standard array **does not** store information about the **number of its elements**. Warning!   function sizeof () does not return the number of elements.

The definition of a multidimensional array:

type <u>array_name</u> [size_1] [size_2] [size_3] . . . ;

examples:
  char Rubik_Cube [3][3][3];

  float matrix [5] [2];   *// ← two-dimensional array of 5 rows, of 2 columns,*

0 , 0	0 , 1	
1 , 0	**1 , 1**	1 , 2
2 , 0	**2 , 1**	
3 , 0	**3 , 1**	
4 , 0	**4 , 1**	

matrix representation ↓ in computer memory

0 , 0	0 , 1	1 , 0	1 , 1	2 , 0	2 , 1	3 , 0	3 , 1	4 , 0	4 , 1

             ↑

matrix[1][2]

The definition can be combined with initialization:

```
int     tab[ 10 ];      // ← definition without initialization
int     initialized_array [ 10 ]  =      { 20, -3, 12, 1, 0, 7, -5, 100, 2, 5 };
char    array_of_chars[ 5 ]  =      { 'a', 'B', '\n', '1', '\0' };
float   matrix_A[ 3 ][ 2 ]  =      { {1,1}, {3.5,7.0}, {-15,100} };
float   matrix_B[ 3 ][ 2 ]  =      { 1, 1, 3.5, 7.0, -15, 100 };
```

⇒ Subsequent "initializers" always are inserted to the subsequent "cells" (therefore inner curly braces can be omitted).

⇒ If the list of initializers is shorter than the number of elements in the array, other elements are complemented with zeros or NULL pointers
eg.:

```
int   tab[ 10 ]  =  { 20, -3, 12, 1 };
```
is equivalent to:
```
int   tab[ 10 ]  =  { 20, -3, 12, 1, 0, 0, 0, 0, 0, 0 };
```

definition:
```
float   matrix [ 3 ][ 2 ]  =  { {1}, {3.5,7.0} };
```
is equivalent to:
```
float   matrix[ 3 ][ 2 ]  =  { {1,0}, {3.5,7.0}, {0,0} };
```
or:
```
float   macierz[ 3 ][ 2 ]  =  { 1, 0, 3.5, 7.0, 0, 0 };
```

⇒ In C language, the initializers must be constant,
whereas in C ++ the initializers can be both constant or variable.

The use of constants to define the amount of array elements:

```
int  array [ 100 ] ;              // the size defined directly

#define  SIZE_C  100             // the definition of a constant using C-style
int  array [ SIZE_C ] ;

const int SIZE_2 = 100 ;         // constant definition using C++ style
int  array_2 [ SIZE_2 ] ;

for( int i=0 ; i < SIZE_2 ; i++ )     // the example of using the size constant
    printf ( "%d" , array_2[ i ] );
```

Basic operations on the array elements

```c
#include <stdio.h>                          // implementation in "C"
int main( )
{
   const  SIZE = 4 ;
   int    Tab [ SIZE ] ;

   // direct assignment of values
   Tab[ 0 ] = 0 ;
   Tab[ 1 ] = 10 ;
   Tab[ 2 ] = - 20 ;
   Tab[ 3 ] = 3 ;

   // entering array content from the keyboard
   scanf( "%d" , &Tab[ 0 ] ) ;
   scanf( "%d %d" , &Tab[ 1 ] , &Tab[ 2 ] ) ;

   printf( " Enter a fourth element = " );
   scanf( "%d" , &Tab[ 3 ] ) ;

   // primitive summation of array elements
   long sum = Tab[0] + Tab[1] + Tab[2] + Tab[3] ;

   // displaying the contents of array elements

   printf( " Tab[1] = %5d " , Tab[0] );
   printf( " Tab[2] = %5d " , Tab[1] );
   printf( " Tab[3] = %5d " , Tab[2] );
   printf( " Tab[4] = %5d " , Tab[3] );

   // indirect indication of the element, using the auxiliary variable
   int  i = 2 ;
   Tab[ i ] = 10;          // is quivalent to:      Tab[ 2 ] = 10;

   // entering the index of element, from the keyboard
   printf( " Enter position of the element, whose value you want to load " );
   scanf( "%d" , &i );

   printf( " Enter a new value of  Tab[ %d ] = " , i );
   scanf( "%d" , &Tab[ i ] );

   printf( " New value of Tab[ %d ] is equal %d " , i , Tab[ i ] );
   return 0;
}
```

Application of command "for" to automate operations on arrays

```c
#include <stdio.h>                                    // implementation in "C"
int main( )
  {
      #define  LENGTH  10
      float  numbers[ LENGTH ];         // the definition of an array of real numbers
      int  i ;


      // --- initialization of an array with successive  even numbers: 0,2,4,6 ,...

      for( i = 0 ; i < LENGTH ; i++ )
        numbers[ i ] = 2*i ;


      // --- loading the contents of array elements from the keyboard
      for( i = 0 ; i < LENGTH ; i++ )
        {
            printf( " Enter  Number [%2d] = ", i+1 );
            scanf( " %f " , &numbers[ i ] );
        }


      // --- control display of the table contents
      for( i = 0 ; i < LENGTH ; i++ )
        printf( " Number[%2d] = %10.3f ", i+1 , numbers[ i ] );


      // --- summing the elements of an array
      float sum = 0 ;
      for( i = 0 ; i < LENGTH ; i++ )
        sum = sum + numbers[ i ];              //  sum += numbers[ i ];
      printf( " The sum of the array elements is: %.2f " , sum );


      // --- counting the number of elements with positive values
      int counter = 0 ;
      for( i = 0 ; i < LENGTH ; i++ )
        if(numbers[ i ] > 0 )
            counter = counter + 1 ;      // counter += 1;   or   counter ++;

      if( counter > 0 )
        printf( " Number of positive elements = %d " , counter );
      else
        printf( " No positive values in the array " );

      return 0;
  }
```

continued examples of array processing with loops „for”

```cpp
#include <iostream>                                    // implementation in "C++"
using namespace std;
int main( )
  {
     const int LENGTH = 10;
     int  array[ LENGTH ];

     for( int i = 0 ; i < LENGTH ; i++ )     //------- entering values from keyboard
       {
          cout << "Array[" << (i+1) << "] = ";
          cin >> array[ i ];
       }

     int counter=0;               //------------------------- counting of nonzero elements
     for( int i = 0 ; i < LENGTH ; i++ )
        if( array[ i ] )
           counter ++;

     int sum = 0;            //---------------------------- an average of negative values
     counter =0;
     for( int i = 0 ; i < LENGTH ; i++ )
        if( array[ i ] < 0 )
           {
              sum += array[ i ];
              counter++;
           }
     if( counter )
        {
           double average = (double)sum / counter;
           cout << endl << "Average of negatives = " << average ;
        }
     else
        cout << endl << " There are no elements with negative values " ;

     int max = array[0];          // determine the maximum (value & position)
     int position = 0;
     for( int i=1; i< LENGTH ; i++ )
        if( max < array[ i ] )
           {
              max = array[ i ];
              position = i ;
           }
     cout << endl << "The biggest value is equal to: " << max;
     cout << endl << "Its position is: " << (position+1);

     return 0;
  }
```

Examples of functions that operate on arrays

```c
#include <stdio.h>                                  // implementation in "C"
#define  LENGTH  10

void readArray ( double array[ ] )
{
   int i;
   printf( " Enter values of the array elements \n" );
   for( i = 0 ; i < LENGTH ; i++ )
     {
        printf( "Array[%2d] = ", i+1  );
        scanf( "%lf" , &array[ i ] );
     }
} //-------------------------------------------------------------------------- readArray


void displayArray( double array[ ] )
{
   int i;
   printf( " The values of array elements are equal: \n" );
   for( i = 0 ; i < LENGTH ; i++ )
      printf( "Array[%2d] = %f", i+1 , array [ i ] );
   printf( " Press any key " );
   fflush(stdin);
   getchar();
} //-------------------------------------------------------------------------- displayArray


void addArrays(double input_1[ ], double input_2[ ], double result[ ])
{
   int i;
   for( i = 0 ; i < LENGTH ; i++ )
      result[ i ] = input_1[ i ] + input_2[ i ] ;
} //-------------------------------------------------------------------------- addArrays


int main( void )
  {
     double  A[ LENGTH ] ;
     double  B[ LENGTH ],  C[ LENGTH ] ;

     readArray( A );
     displayArray( A );

     readArray ( B );
     addArrays( A, B, C );
     displayArray ( C );

     return 0;
}
```