STEP computer ACADEMY

PROGRAMMING C

Lesson No. 6

Programming

C

Contents

1. Necessity of data aggregate

Today we will talk about data storage. At one of our lessons we learnt about a variable and defined it as a part of a random memory for information storing. Definitely, a program cannot exist without variables, though occasionally simple variable do not solve data operating problems. The thing is that each variable reviewed in the previous lessons can store only one information element at a time. To store the second one we need to create another variable. But what to do if we need to store many elements of однородных data types. It will be quite inconvenient to create a variable for each element. And what if we need to work with hundreds of elements? The task becomes impossible to fulfill. Agree that it is mad to create several hundreds of elements.

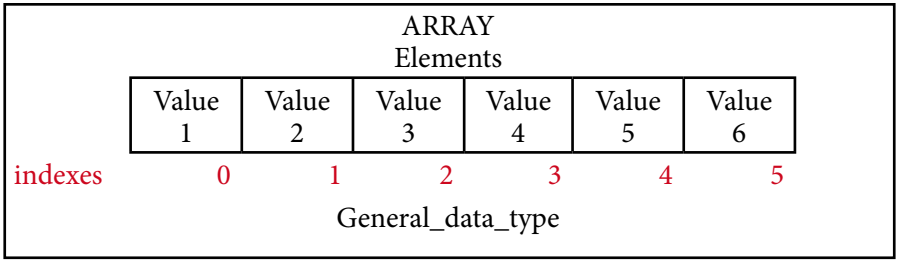
And how to resolve this seemingly difficult problem?! In our case so called arrays will prove helpful. Let's review the definition and peculiarities.

Concept of an array.

1. Array — is a set of variables allowing storing several one-type values.
2. All values of the aggregate are united under one name.
3. Wherein each variable within the array is an independent unit called element.
4. Each element has its own index number — index. We can address a particular array element using its index.
5. Numbering of elements within the array starts from zero.

Scheme:

Proceeding from the above described assertions a general array representation scheme looks like:

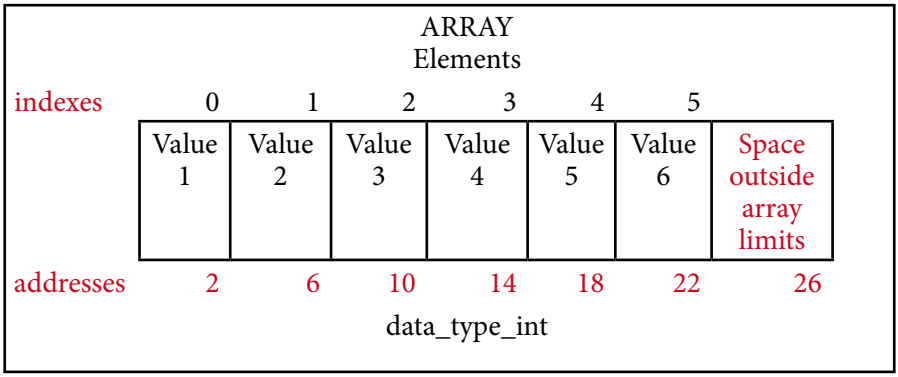


Array position within the memory:

Array is situated within the memory in sequential order, element after element. At first there is a zero one, then first one, etc. elements are ordered in the manner of address ascending: one array element is put right from another one on a distance of equal number of bytes and at the same time equal to basic array type. Formula which is used for positioning within the array:

basic address + basic type size * index;

If we show a wrong address there is exercised a positioning of the basic address for the address obtained by the formulas. Wherein the program obtains a full access to memory cell contents, which actually does not belong to it. As a result there may be an error at the execution stage..



In conclusion we should note that each array element has its own dimension, which directly depends on the type of the whole array. For example, if an array has int data type — a size of each element within it is equal to 4 bytes. Therefore, general size of the whole array is calculated by the formula:

GENERAL_SIZE = DATA_TYPE_SIZES*NUMBER_OF ELEMENTS_WITHIN_ARRAY

Now theoretically we know almost everything about an array. We are just left to get acquainted with the practical part and make sure how it is easy and convenient to create and use this construction. With this purpose we pass to the next lesson unit.

2. Creating an array and filling it with data

Array declaration syntax.

At first we need to learn how to create an array. And with this purpose first of all we should define a general syntax. Second, find out what are the rules and restrictions of the syntax.

```
data_type array_name[number_of elements];
```

1. data_type — any of existing and familiar data types. Each array element will have this type.

2. array_name — any name, which is a subject to the «variable names rule» (we studied these rules in the first lesson).

3. number_of elements — number of elements within an array. There should be an integer constant value at this place. This value can be either in a form of integer literal or integer constant variable.

Note: Pay attention that number of array elements should be defined at the stage of program creation. In other words, it would be impossible to preset dimension of such array depending on a condition or user's decision. This will result in an error at a compile stage.

First method.

Ar array is declared and it consists of 5 elements, each of them is of int data type.

```
int ar[5];
```

Second method.

Size constant is declared and its value is equal to 3, and afterwards a br array consisting of 3 elements, each of them is of double data type.

```
const int size=3;
double br[size];
```

Note: We would recommend you to use second form of record, for it is more precise and convenient.

Addressing array elements.

Let's review how to address a particular array element.

```
value record
array_name[element_index]=value;
obtaining a value
cout<<array_name[element_index];
```

Here a position of element_index can be taken by **ANY** integer, including an expression with the integer in the result.

```
const int size=5;
int ar[size]; // creating an array
ar[2]=25; // record of value 25 into the element with index 2
// displaying on a screen element with index 2 - 25
cout<<ar[2]<<"\n";
```

Note: we remind again that array elements numeration starts from zero! Thus, the last out of five elements within the array will have an index 4. We cannot go outside of an array limits; it will lead to the mistake at the execution stage.

Variants of array initialization:

We can fill in an array by two methods:

First method — initialization upon creation.

data_type array_name[number of elements]=value1, value 2, … value n};

```
const int size=3;
int ar[size]={1,30,2};
```

This form of initialization has a list of peculiarities:

1. All values within the initialization list have the same data type as an array, that is why upon creation a number of elements may be not indicated. Operating system defines an array size proceeding from the number of elements within the initialization list.

Data_type array_name[]={value1, value 2, value 3, … value n};

```
int ar[]={1,30,2};      /*within this string an array will automatically get size 3.*/
```

2. If number of elements within an initialization list is less than number of array elements, the rest of the values will be automatically exercised by zeros:

```
int ar[5]={1,2,3}
```

This record is equal to the record:

```
int ar[5]={1,2,3,0,0};
```

3. If within the initialization list there are more values than number of array elements an error occurs at the compile stage:

```
int array[2]={1,2,3};   // error at the compile stage
```

4. Upon initializing arrays we can use a uniform initialization that is already familiar for us

```
int arr[]{1, 2, 3};
int arr2[4]{11, 21, 31};
```

Second method — initializing an array using a loop.

In this case we can fill an array with the values by means of a user. Project name InitArray.

```
#include<iostream>
using namespace std;
void main()
{
   const int size=3;
   int ar[size];   //creating an array out of three elements
   //loop searching array elements
   for (int i=0;i<size;i++)
   {
      cout<<"Enter element\n";
   /* at each loop iteration a user gets an elements with the index i for filling
in. the secret is that i is a new value each time */
      cin>>ar[i];
   }
}
```

Displaying array contents on a screen.

You might have guessed that the majority of operations with arrays is more convenient to exercise by means of loops, searching the elements one after another. It is true indeed and displaying on a screen is not an exception. Let's give an example of a program that creates, fills in and displays on a screen an array. Project name ShowArray.

```
#include<iostream>
using namespace std;
void main()
{
   const int size=3;
   int ar[size];   //creating an array out of three elements
   //loop searching array elements
   for (int i=0;i<size;i++)
   {
      cout<<"Enter element\n";
   /* on each loop iteration user gets an elements with index i for filling it in.
And the secret is that i has a new value every time */
      cin>>ar[i];
   }
   cout<<"\n\n";
   //loop searching array elements
   for (int i=0;i<size;i++)
   {
      //displaying an element with index i
      cout<<ar[i]<<"\n";
   }
}
```

Now when we have acquainted with the arrays let's pass to the next lesson units and review several use cases on working with them.

3. Program sample that finds a sum of negative array elements

Problem statement:

Write a program that finds a sum of all negative values within an array. Project name AmountOfNegative.

Realization code:

```
#include <iostream>
using namespace std;
void main ()
{
        //defining array size
        const int size=5;

        //creating and initializing an array by data
        int ar[size]={23,-11,9,-18,-25};

        //variable for amount accumulation
        int sum=0;

        //loop searches array elements in sequence
        for (int i=0;i<size;i++)
        {
        //if element value is negative (less than zero)
        if(ar[i]<0)
        sum+=ar[i]; //add its value to the total sum
        }

        //displaying a sum value on a screen
        cout<<"Sum = "<<sum<<"\n\n";
}
```

Comments to the code:

1. . Loop searches the elements from 0 to size. Whereas size is n вт included within the checked range, because index of the last element is size-1.

2. At each loop iteration there is a check of element contents on negative value.

3. If the value is less than zero it is added to the sum.

As you can see working with an array is like analyzing some range. But in our case a minimal range limit is 0, and maximal is defined by a number of elements within an array.

Program sample for finding a minimal and maximal array element.

Problem statement:

Rite a program, which finds the minimal and maximal value within an array and displays it on a screen. Project name MinMaxElement.

Realization code:

```
#include <iostream>
using namespace std;
void main ()
{
        // defining a number of array elements
        const int size=5;

        // creating and initializing an array
        int ar[size]={23,11,9,18,25};

        int max=ar[0];  // assume 0 is a maximum element
        int min=ar[0];  // assume 0 is a maximum element

        //a loop searches the array elements starting from 1
        for (int i=1;i<size;i++)
        {
           //if the current element is less than minimum
```

```
            if(min>ar[i])
              //rewrite the minimum value
              min=ar[i];

            // if the current element is greater than maximum
            if(max<ar[i])
              // rewrite the maximum value
              max=ar[i];
        }

         // displaying the result on a screen
         cout<<"Max = "<<max<<"\n\n";
         cout<<"Min = "<<min<<"\n\n";
}
```

Comments to the code:

1. At first we assume that array element with index 0 is a minimal one.

2. Write an index 0 element value into the min variable.

3. Afterwards, in order to verify or dispose the fact we search through all elements starting from the element with index 1 within a loop.

4. At each loop iteration we compare a supposed minimum with the current array element (index I element).

5. If there is a value less than supposed minimum — min value is rewritten for the smaller found value and analysis continues.

All described about actions are appropriate for maximum too, but we have to search for a bigger value. Now when we are familiar with the arrays and have reviewed a few examples it is time to make something by ourselves. Good luck in passing the test and doing your home assignment.

4. Home assignment

Input data in all assignments described below is an array of 10 elements filled in by a user using a keyboard.

1.Write a program, which displays array content in an inverted way.

```
Example: array 23 11 6 turns into 6 23 11.
```

2. Write a program, which finds a sum of even and uneven array elements.

3. Write a program, which finds the values duplicated two or more times within an array and displays them on a screen.

4. Write a program, which finds the smallest uneven number within an array and displays it on a screen.