ARRAY

C++ provides a data structure, **the array**, which stores a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.

Instead of declaring individual variables, such as number0, number1, ..., and number99, you declare one array variable such as numbers and use numbers[0], numbers[1], and ..., numbers[99] to represent individual variables. A specific element in an array is accessed by an index.

All arrays consist of contiguous memory locations. The lowest address corresponds to the first element and the highest address to the last element.

Declaring Arrays:

To declare an array in C++, the programmer specifies the type of the elements and the number of elements required by an array as follows:

```
type arrayName [ arraySize ];
```

This is called a single-dimension array. The **arraySize** must be an integer constant greater than zero and **type** can be any valid C++ data type. For example, to declare a 10-element array called balance of type double, use this statement:

```
double balance[10];
```

Initializing Arrays:

You can initialize C++ array elements either one by one or using a single statement as follows:

```
double balance[5] = {1000.0, 2.0, 3.4, 17.0, 50.0};
```

The number of values between braces { } can not be larger than the number of elements that we declare for the array between square brackets [ ]. Following is an example to assign a single element of the array:

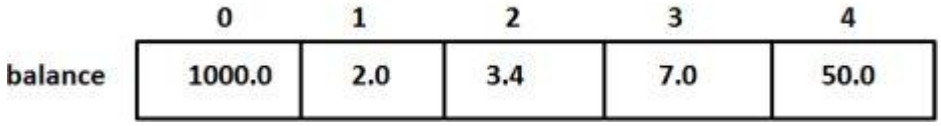
If you omit the size of the array, an array just big enough to hold the initialization is created. Therefore, if you write:

```
double balance[] = {1000.0, 2.0, 3.4, 17.0, 50.0};
```

You will create exactly the same array as you did in the previous example.

```
balance[4] = 50.0;
```

The above statement assigns element number 5th in the array a value of 50.0. Array with 4th index will be 5th ie. last element because all arrays have 0 as the index of their first element which is also called base index. Following is the pictorial representaion of the same array we discussed above:



Accessing Array Elements:

An element is accessed by indexing the array name. This is done by placing the index of the element within square brackets after the name of the array. For example:

```
double salary = balance[9];
```

The above statement will take 10th element from the array and assign the value to salary variable. Following is an example which will use all the above mentioned three concepts viz. declaration, assignment and accessing arrays:

```
#include <iostream>
using namespace std;

#include <iomanip>
using std::setw;

int main ()
{
   int n[ 10 ]; // n is an array of 10 integers

   // initialize elements of array n to 0
   for ( int i = 0; i < 10; i++ )
   {
      n[ i ] = i + 100; // set element at location i to i + 100
   }
   cout << "Element" << setw( 13 ) << "Value" << endl;

   // output each array element's value
   for ( int j = 0; j < 10; j++ )
   {
      cout << setw( 7 )<< j << setw( 13 ) << n[ j ] << endl;
   }

   return 0;
}
```

This program makes use **setw()** function to format the output. When the above code is compiled and executed, it produces following result:

```
Element        Value
      0          100
      1          101
      2          102
      3          103
      4          104
      5          105
      6          106
      7          107
      8          108
      9          109
```

Pointers

Full info

http://www.cplusplus.com/doc/tutorial/pointers/http://www.cplusplus.com/doc/tutorial/pointers/

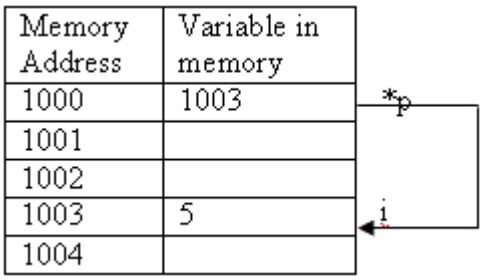
Short info

What is pointer? Explain with examples

A pointer is a variable that holds a memory address. This address is the location of another object (typically, a variable) in memory. That is, if one variable contains the address of another variable, the first variable is said to point to the second.

A pointer declaration consists of a base type, an *, and the variable name. The general form of declaring a pointer variable is:

type *name;
type is the base type of the pointer and may be any valid type.
name is the name of pointer variable.
The base type of the pointer defines what type of variables the pointer can point to.



Two special pointer operators are: * and &.
The & is unary operator that returns the memory address of its operand. It is “the address of” operand.
The * is complement of &. It is also a unary operator and returns the value located at the address that follows.

int i, *p;
i = 5;
p = &i; //places the memory address of i into p

The expression *p will return the value of variable pointed to by p.