**Array**

An **array** is a data structure which allows a collective name to be given to a group of elements which all have the same type. An individual element of an **array** is identified by its own unique index (or subscript). An **array** can be thought of as a collection of numbered boxes each containing one data item.

#include <iostream>

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* This example shows use of for loops. \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int main()

{

const int array\_size = 10; //array size

int array[array\_size] = {32,85,41,9,78,35,24,10,1,65};

cout << "Displaying Array..." << endl;

for(int i = 0; i < array\_size; i++)

{

cout << "Array[" << i << "] => " << array[i] << endl;

}

return 0;

}

## Pointers?

A **pointer** is a variable whose value is the address of another variable. Like any variable or constant, you must declare a pointer before you can work with it. The general form of a pointer variable declaration is:

#include <iostream>

using namespace std;

int main ()

{

int var = 20; // actual variable declaration.

int \*ip; // pointer variable

ip = &var; // store address of var in pointer variable

cout << "Value of var variable: ";

cout << var << endl;

// print the address stored in ip pointer variable

cout << "Address stored in ip variable: ";

cout << ip << endl;

// access the value at the address available in pointer

cout << "Value of \*ip variable: ";

cout << \*ip << endl;

return 0;

}

**Reference**

* A reference is an entity that is an alias for another object.
* A reference is not a **variable** as a **variable** is only introduced by the declaration of an object. An object is a region of storage and, in C++, references do not (necessarily) take up any storage.

#include <iostream>

int main ()

{

int i;

double d;

int& r = i;

double& s = d;

i = 5;

cout << "Value of i : " << i << endl;

cout << "Value of i reference : " << r << endl;

d = 11.7;

cout << "Value of d : " << d << endl;

cout << "Value of d reference : " << s << endl;

return 0;

}

## C++ References vs Pointers:

References are often confused with pointers but three major differences between references and pointers are:

* You cannot have NULL references. You must always be able to assume that a reference is connected to a legitimate piece of storage.
* Once a reference is initialized to an object, it cannot be changed to refer to another object. Pointers can be pointed to another object at any time.
* A reference must be initialized when it is created. Pointers can be initialized at any time.

**\_Differences Between C And C++**

C++, as the name suggests, is a superset of C. As a matter of fact, C++ can run most of C code while C cannot run C++ code. Here are the 10 major differences between C++ & C…

1. C follows the procedural programming paradigm while C++ is a [multi-paradigm](http://durofy.com/programming/c-as-a-multi-paradigm-programming-language/) language(procedural as well as object oriented)

In case of C, importance is given to the steps or procedure of the program while C++ focuses on the data rather than the process.  
Also, it is easier to implement/edit the code in case of C++ for the same reason.

2. In case of C, the data is not secured while the data is secured(hidden) in C++

This difference is due to specific [OOP features](http://durofy.com/programming/the-basics-of-object-oriented-programming/) like Data Hiding which are not present in C.

3. C is a low-level language while C++ is a middle-level language

4. C uses the top-down approach while C++ uses the bottom-up approach

5. C is function-driven while C++ is object-driven. Functions are the building blocks of a C program while objects are building blocks of a C++ program.

6. C++ supports function overloading while C does not

Overloading means two functions having the same name in the same program. This can be done only in C++ with the help of [Polymorphism](http://durofy.com/programming/the-basics-of-object-oriented-programming/)(an OOP feature)

7. We can use functions inside structures in C++ but not in C.

In case of C++, functions can be used inside a structure while structures cannot contain functions in C.

8. The NAMESPACE feature in C++ is absent in case of C

C++ uses NAMESPACE which avoid name collisions. For instance, two students enrolled in the same university cannot have the same roll number while two students in different universities might have the same roll number. The universities are two different namespace & hence contain the same roll number(identifier) but the same university(one namespace) cannot have two students with the same roll number(identifier)

9. The standard input & output functions differ in the two languages

C uses scanf & printf while C++ uses cin>> & cout<< as their respective input & output functions

10. C++ allows the use of reference variables while C does not

Reference variables allow two variable names to point to the same memory location. We cannot use these variables in C programming.

11. C++ supports Exception Handling while C does not.

**Break**

The **break** statement has the following two usages in C++:

* When the **break** statement is encountered inside a loop, the loop is immediately terminated and program control resumes at the next statement following the loop.
* It can be used to terminate a case in the **switch** statement (covered in the next chapter).

If you are using nested loops (i.e., one loop inside another loop), the break statement will stop the execution of the innermost loop and start executing the next line of code after the block.

#include <iostream>

using namespace std;

int main ()

{

// Local variable declaration:

int a = 10;

// do loop execution

do

{

cout << "value of a: " << a << endl;

a = a + 1;

if( a > 15)

{

// terminate the loop

break;

}

}while( a < 20 );

return 0;

}

**continue**

The **continue** statement works somewhat like the break statement. Instead of forcing termination, however, continue forces the next iteration of the loop to take place, skipping any code in between.

For the **for** loop, continue causes the conditional test and increment portions of the loop to execute. For the **while** and **do...while** loops, program control passes to the conditional tests.

#include <iostream>

using namespace std;

int main ()

{

int a = 10;

do

{

if( a == 15)

{

// skip the iteration.

a = a + 1;

continue;

}

cout << "value of a: " << a << endl;

a = a + 1;

}while( a < 20 );

return 0;

}