C++ pointers are easy and fun to learn. Some C++ tasks are performed more easily with pointers, and other C++ tasks, such as dynamic memory allocation, cannot be performed without them.

As you know every variable is a memory location and every memory location has its address defined which can be accessed using ampersand (&) operator which denotes an address in memory. Consider the following which will print the address of the variables defined:

```cpp
#include <iostream>

using namespace std;

int main ()
{
    int var1;
    char var2[10];

    cout << "Address of var1 variable: ";
    cout << &var1 << endl;

    cout << "Address of var2 variable: ";
    cout << &var2 << endl;

    return 0;
}
```

When the above code is compiled and executed, it produces result something as follows:

```
Address of var1 variable: 0xbfebd5c0
Address of var2 variable: 0xbfebd5b6
```

**What Are 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:

```cpp
type *var-name;
```

Here, `type` is the pointer's base type; it must be a valid C++ type and `var-name` is the name of the pointer variable. The asterisk you used to declare a pointer is the same asterisk that you use for multiplication. However, in this statement the asterisk is being used to designate a variable as a pointer. Following are the valid pointer declaration:

```cpp
int *ip;       // pointer to an integer
double *dp;    // pointer to a double
float *fp;     // pointer to a float
char *ch;      // pointer to a character
```

The actual data type of the value of all pointers, whether integer, float, character, or otherwise, is the same, a long hexadecimal number that represents a memory address. The only difference between pointers of different data types is the data type of the variable or constant that the pointer points to.

**Using Pointers in C++:**

There are few important operations, which we will do with the pointers very frequently. 

- **a)** we define a pointer variables
- **b)** assign the address of a variable to a pointer and
- **c)** finally access the value at the address available in the pointer variable. This is done by using unary operator * that returns the value of the variable located at the address specified by its operand. Following example makes
use of these operations:

```cpp
#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;
}
```

When the above code is compiled and executed, it produces result something as follows:

```
Value of var variable: 20
Address stored in ip variable: 0xbfc601ac
Value of *ip variable: 20
```

**C++ Pointers in Detail:**

Pointers have many but easy concepts and they are very important to C++ programming. There are following few important pointer concepts which should be clear to a C++ programmer:

<table>
<thead>
<tr>
<th>Concept</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>C++ Null Pointers</td>
<td>C++ supports null pointer, which is a constant with a value of zero defined in several standard libraries.</td>
</tr>
<tr>
<td>C++ pointer arithmetic</td>
<td>There are four arithmetic operators that can be used on pointers: ++, --, +, -</td>
</tr>
<tr>
<td>C++ pointers vs arrays</td>
<td>There is a close relationship between pointers and arrays. Let us check how?</td>
</tr>
<tr>
<td>C++ array of pointers</td>
<td>You can define arrays to hold a number of pointers.</td>
</tr>
<tr>
<td>C++ pointer to pointer</td>
<td>C++ allows you to have pointer on a pointer and so on.</td>
</tr>
<tr>
<td>Passing pointers to functions</td>
<td>Passing an argument by reference or by address both enable the passed argument to be changed in the calling function by the called function.</td>
</tr>
<tr>
<td>Return pointer from functions</td>
<td>C++ allows a function to return a pointer to local variable, static variable and dynamically allocated memory as well.</td>
</tr>
</tbody>
</table>