POINTERS AND REFERENCES

C++ Pointers

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 –

```
#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 the following result –

```
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 –

```
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 –

```
int    *ip;    // pointer to an integer
double *dp;    // pointer to a double
float  *fp;    // pointer to a float
char   *ch     // pointer to 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 variable. **(b)** Assign the address of a variable to a pointer. **(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 –

```
#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
```

Pointers in C++

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 –

Sr.No	Concept & Description
1	**Null Pointers** C++ supports null pointer, which is a constant with a value of zero defined in several standard libraries.
2	**Pointer Arithmetic** There are four arithmetic operators that can be used on pointers: ++, --,

	+, -
3	**Pointers vs Arrays** There is a close relationship between pointers and arrays.
4	**Array of Pointers** You can define arrays to hold a number of pointers.
5	**Pointer to Pointer** C++ allows you to have pointer on a pointer and so on.
6	**Passing Pointers to Functions** Passing an argument by reference or by address both enable the passed argument to be changed in the calling function by the called function.
7	**Return Pointer from Functions** C++ allows a function to return a pointer to local variable, static variable and dynamically allocated memory as well.

C++ References

A reference variable is an alias, that is, another name for an already existing variable. Once a reference is initialized with a variable, either the variable name or the reference name may be used to refer to the variable.

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.

Creating References in C++

Think of a variable name as a label attached to the variable's location in memory. You can then think of a reference as a second label attached to that memory location. Therefore, you can access the contents of the variable through either the original variable name or the reference. For example, suppose we have the following example –

```
int i = 17;
```

We can declare reference variables for i as follows.

```
int& r = i;
```

Read the & in these declarations as **reference**. Thus, read the first declaration as "r is an integer reference initialized to i" and read the second declaration as "s is a double reference initialized to d.". Following example makes use of references on int and double –

```
#include <iostream>


using namespace std;


int main () {
   // declare simple variables
   int    i;
   double d;


   // declare reference variables
   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;

}
```

When the above code is compiled together and executed, it produces the following result –

```
Value of i : 5
Value of i reference : 5
Value of d : 11.7
Value of d reference : 11.7
```

References are usually used for function argument lists and function return values. So following are two important subjects related to C++ references which should be clear to a C++ programmer –

Sr.No	Concept & Description
1	**References as Parameters** C++ supports passing references as function parameter more safely than parameters.
2	**Reference as Return Value** You can return reference from a C++ function like any other data type.