**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:

أ‌- 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**

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;

}

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:

**Passing parameters by references**

We have discussed how we implement call by reference concept using pointers. Here is another example of call by reference which makes use of C++ reference:

#include <iostream>

using namespace std;

// function declaration

void swap(int& x, int& y);

int main ()

{

// local variable declaration:

int a = 100;

int b = 200;

cout << "Before swap, value of a :" << a << endl;

cout << "Before swap, value of b :" << b << endl;

/\* calling a function to swap the values.\*/

swap(a, b);

cout << "After swap, value of a :" << a << endl;

cout << "After swap, value of b :" << b << endl;

return 0;

}

// function definition to swap the values.

void swap(int& x, int& y)

{

int temp;

temp = x; /\* save the value at address x \*/

x = y; /\* put y into x \*/

y = temp; /\* put x into y \*/

return;

}

**Returning values by reference**

A C++ program can be made easier to read and maintain by using references rather than pointers. A C++ function can return a reference in a similar way as it returns a pointer.

When a function returns a reference, it returns an implicit pointer to its return value. This way, a function can be used on the left side of an assignment statement. For example, consider this simple program:

#include <iostream>

#include <ctime>

using namespace std;

double vals[] = {10.1, 12.6, 33.1, 24.1, 50.0};

double& setValues( int i )

{

return vals[i]; // return a reference to the ith element

}

// main function to call above defined function.

int main ()

{

cout << "Value before change" << endl;

for ( int i = 0; i < 5; i++ )

{

cout << "vals[" << i << "] = ";

cout << vals[i] << endl;

}

setValues(1) = 20.23; // change 2nd element

setValues(3) = 70.8; // change 4th element

cout << "Value after change" << endl;

for ( int i = 0; i < 5; i++ )

{

cout << "vals[" << i << "] = ";

cout << vals[i] << endl;

}

return 0;

}

When returning a reference, be careful that the object being referred to does not go out of scope. So it is not legal to return a reference to local var. But you can always return a reference on a static variable.

int& func() {

int q;

//! return q; // Compile time error

static int x;

return x; // Safe, x lives outside this scope

}