**Introducing Reference Types**

 Bookmark this page

C++ includes a type known as a reference.  A reference type is simply an alias for another type.  It "refers to" or "references" another type in your code.  A reference type overloads the use of the & operator.  So far you have seen the & operator used to represent the address-of operator for obtaining the address of a variable.  References also use this operator to denote a reference to another variable.  
  
You declare a reference type using a syntax similar to declaring a pointer variable.  That is, you declare the data type of the C++ variable or object that will be referred to, then you use the & character followed immediately by the reference type name.  Finally, it's important to note that when declaring a reference, you must assign it at that time.  It behaves similar to a constant in this sense.  An example demonstrates the declaration.  
  
    int num = 3;  
    int &refNum = num;  
    int &refNum2;  
    cout << refNum << endl;  
  
In the code sample above we create an integer variable called num and assign it a value of 3.  Next we declare a reference called refNum.  The & character tells us this is a reference value.  Note that we immediately assign it to the num variable.  This binds refNum to num.  This reference cannot   be reassigned later in program code.    
  
The third line in the code sample will cause an error because it has not been initialized.  The last line will output the value 3, which is correct because refNum is an alias or reference to num, which holds the value 3.  
  
To see how this affects the original value of num, let's create another small code sample that displays the value for num, modifies it through refNum, then outputs the memory addresses of num and refNum.  
  
    int num = 3;  
    int &refNum = num;  
  
    cout << "num contains " << num << endl;  
    cout << "refNum contains " << refNum << endl;  
  
    refNum++;    // increment refNum by 1  
  
    cout << "num contains " << num << endl;  
    cout << "refNum contains " << refNum << endl;  
    cout << "refNum is located at " << &refNum << " and num is located at " << &num << endl;  
  
  
On my computer, this code produces the following output.  
  
    \*num contains 3  
    refNum contains 3  
    num contains 4  
    refNum contains 4  
    refNum is located at 0018FE14 and num is located at 0018FE14  
    Press any key to continue . . .\*  
  
Lines 1 and 2 are now familiar to you as we created and assign num and a reference refNum.  The first two cout lines output the value of num and refNum and you can see that both contain the value 3.  On line 5 we increment refNum, which will also increment num due to refNum being an alias for num.  
  
The remaining lines show us that num was indeed incremented to 4 as was refNum.  The last line outputs the memory address of num and refNum to show that they both point to the same memory location.   As a result, any changes made to refNum affect num.  
  
References are commonly used with parameters to functions.  The default mechanism used by C++ when calling a function, is to pass arguments into the parameters of that function, by value.  This means that only the value is passed.  This has one effect in a program as we see here.  
  
    using namespace std;  
  
    void passByValue(int);  
  
    int main()  
    {  
  
        int num = 3;  
        cout << "In main()" << endl;  
        cout << "Value of num is " << num << endl;  
  
        passByValue(num);  
  
        cout << "Back in main and the value of num is  " << num << endl;  
  
  
        return 0;  
    }  
  
    void passByValue(int num1)  
    {  
        cout << "In passByValue()" << endl;  
        cout << "Value of num1 is " << num1 << endl;  
  
        // modify num1, won't impact num  
        num1++;  
  
        cout << "num1 is now " << num1 << endl;  
    }  
  
In this sample code, we declare our function prototype before main().  Inside main(), we declare and assign the variable num.  We output a line indicating which function we are in (main()) and then what the value of num is (3).  
  
Next we call the passByValue(int num1) function and pass num as the argument to the num1 parameter.  We print a line indicating that we are inside the passByValue() function and then output the value of num1 to show that indeed has the same value as num (3).  
  
Inside passByValue, we increment num1 and output its new value, which in this case is 4.  
  
Once passByValue() is finished executing, control returns back to main() where we indicate we are back inside the main function and we output the value for num, which is still 3, not 4.    
  
The pass by value behavior shows that we only passed in a copy of the value that is held in num and not a reference to num itself.  Therefore any changes inside the passByValue() function only impact the local variable num1 and not the original variable num.   If we wanted to modify num inside our passByValue function, we would need to pass in a reference, not a value.  The following code sample changes the function name, only to make it logical what the function is doing, and the way the parameter is passed in.  
  
    using namespace std;  
  
    void passByRef(int &num1);  
  
    int main()  
    {  
  
        int num = 3;  
        cout << "In main()" << endl;  
        cout << "Value of num is " << num << endl;  
  
        passByRef(num);  
  
        cout << "Back in main and the value of num is  " << num << endl;  
  
  
        return 0;  
    }  
  
    void passByRef(int &num1)  
    {  
        cout << "In passByRef()" << endl;  
        cout << "Value of num1 is " << num1 << endl;  
  
        // modify num1 which will now change num  
        num1++;  
  
        cout << "num1 is now " << num1 << endl;  
    }  
  
This produces the following output:  
  
    In main()  
    Value of num is 3  
    In passByRef()  
    Value of num1 is 3  
    num1 is now 4  
    Back in main and the value of num is  4  
    Press any key to continue . . .  
  
  
Because we passed num as a reference, C++ was able to access the contents of num directly in memory rather than a copy of the value held by num as in the passByValue() example.

For more information , you can see:   
C and C++ with Visual Studio: <https://aka.ms/edx-dev210.2x-vs02>