APPLICATIONS IN CPP

Applying Generic Functions

Generic functions are one of C++'s most useful features. They can be applied to all types of situations. As mentioned earlier, whenever you have a function that defines a generalizable algorithm, you can make it into a template function. Once you have done so, you may use it with any type of data without having to recode it. Before moving on to generic classes, two examples of applying generic functions will be given. They illustrate how easy it is to take advantage of this powerful C++ feature.

A Generic Sort

Sorting is exactly the type of operation for which generic functions were designed. Within wide latitude, a sorting algorithm is the same no matter what type of data is being sorted. The following program illustrates this by creating a generic bubble sort. While the bubble sort is a rather poor sorting algorithm, its operation is clear and uncluttered and it makes an easy-to-understand example.

The **bubble( )** function will sort any type of array. It is called with a pointer to the first element in the array and the number of elements in the array.

```
// A Generic bubble sort.
#include <iostream>
using namespace std;
template <class X> void bubble(
X *items, // pointer to array to be sorted
int count) // number of items in array
{
```

```
register int a, b;
X t;
for(a=1; a<count; a++)
for(b=count-1; b>=a; b--)
if(items[b-1] > items[b]) {
// exchange elements
t = items[b-1];
items[b-1] = items[b];
items[b] = t;
}
}
int main()
{
int iarray[7] = {7, 5, 4, 3, 9, 8, 6};
double darray[5] = {4.3, 2.5, -0.9, 100.2, 3.0};
int i;
cout << "Here is unsorted integer array: ";
for(i=0; i<7; i++)
cout << iarray[i] << ' ';
cout << endl;
cout << "Here is unsorted double array: ";
for(i=0; i<5; i++)
cout << darray[i] << ' ';
cout << endl;
bubble(iarray, 7);
bubble(darray, 5);
cout << "Here is sorted integer array: ";
for(i=0; i<7; i++)
cout << iarray[i] << ' ';
cout << endl;
cout << "Here is sorted double array: ";
```

```
for(i=0; i<5; i++)
cout << darray[i] << ' ';
cout << endl;
return 0;
}
```

The output produced by the program is shown here.

```
Here is unsorted integer array: 7 5 4 3 9 8 6
Here is unsorted double array: 4.3 2.5 -0.9 100.2 3
Here is sorted integer array: 3 4 5 6 7 8 9
Here is sorted double array: -0.9 2.5 3 4.3 100.2
```

As you can see, the preceding program creates two arrays: one integer and one **double**. It then sorts each. Because **bubble( )** is a template function, it is automatically overloaded to accommodate the two different types of data. You might want to try using **bubble( )** to sort other types of data, including classes that you create. In each case, the compiler will create the right version of the function for you.

Source : http://elearningatria.files.wordpress.com/2013/10/cse-iii-object-oriented-programming-with-c-10cs36-notes.pdf