Features

A Multidimensional Array Class

Ali Rahimi

An array is a simple creature in C, perhaps too simple for the more ambitious needs of object-oriented programming in C++. An array class can provide the missing services.

Introduction

Arrays in C++ are not first-class objects. As a result, they suffer from the following major limitations (to paraphrase the ARM [1]):

- An array size is fixed at compile time.
- An array is one dimensional. A multidimensional array is treated as an array of array.
- An array is not self-describing — given just a pointer to an array there is no way to determine its size.

According to the ARM, the last limitation makes "passing pointers to arrays inherently dangerous."

My goal here is to remove the last limitation and mitigate the second one. I use the template and inheritance features of C++ to create an array class with several desirable properties:

- It is self-describing. Given an array object (or a pointer to one) it's possible to learn the number of dimensions and the lower and upper bounds for each dimension (the shape of the array). This "self-describing" feature make it safe to pass pointers to arrays.
- It is as easy to use as C/C++ arrays. The usual subscript notation is used to access an array element.
- It is almost as efficient as C/C++ arrays.
- It can be easily extended to any dimension
- It supports range checking and has the capability to raise range violation exceptions.
- It can be extended to support variable-size multidimensional arrays.

Implementation

The template array class is defined in Listing 1. The one-dimensional array A1D is the base class for all arrays of higher dimension. All the methods are implemented in the A1D array:

- The methods begin and end return the lower and upper bounds of the array, respectively.
- dim returns the number of dimensions.
- operator[] is overloaded to return a reference to an array element type and throw a range error exception on range violations.

The derived classes don't have any code! Every array of dimension n+1 is inherited from a one-dimensional array of n-dimensional arrays.

Usage

Listing 2 provides an example of usage. The code using the array is wrapped in a try block for catching range errors. The example shows a number of different operations. To declare a three-dimensional array, for instance, write:

```
A3D<int, 10,19, 20,29, 30,39> va3d;
```

This is equivalent to the Pascal declaration:

```
va3d : ARRAY[10..19, 20..29, 30..39] OF INTEGER
```

To get the dimension and bounds (shape) for the first subscript of va3d, you can write:

```
int dim1 = va3d.dim();
int low1 = va3d.begin();
int high1 = va3d.end();
```

Here, we get 3 for dim1, 10 for low1, and 19 for high1.

To get the dimension and bounds (shape) for the first subscript of the A2D subarray, write:

```
int dim2 = va3d[low1].dim();
int low2 = va3d[low1].begin();
int high2 = va3d[low1].end();
```

Since va3d[low1] is an A2D array, we get 2 for dim2, 20 for low2, and 29 for high2.

To access array elements, you can write:

```
for( int i=low1; i<=high1; ++i)
 for( int j=low2; j<=high2; ++j)
    for( int k=low3; k<=high3; ++k)
          va3d[i][j][k] = (i*100+j)*100+k;
```

And to obtain a projection, or subarray, you can write:

```
A1D<int, 30,39> va3d_1Dsubarr;
va3d_1Dsubarr = va3d[10][20];
```

Dynamic allocation of arrays is also easy:

```
A3D<int,10,19,20,29,30,39> *pva3d =
    new A3D<int,10,19,20,29,30,39>;
```

To initialize a reference to an array, write:

```
A3D<int,10,19,20,29,30,39>& rva3d = *pva3d;
```

A range-error exception occurs if you attempt to access an element outside the valid range of subscripts. The assignment:

```
va3d[1][20][30] = 1;
```

raises an exception. The example in Listing 2 prints:

```
3-D (sub)array:
   The subscript 1 is out of range: 10:19
```

Similarly, the assignment:

```
va3d[10][20][3] = 3;
```

Will print:

```
1-D (sub)array:
   The subscript 3 is out of range: 30:39
```

It is thus easy to use the array class to detect array range violations in C++ code. Suppose, for example, that you are using a two-dimensional array such as:

```
int A[100][200];
```

Just change the declaration to:

```
Array1D<int, 0, 99, 0, 199> A;
```

recompile and run. An exception will be raised whenever the subscript range is violated.

References

Margaret A. Ellis and Bjarne Stroustrup. The Annotated C++ Reference Manual (also known as the ARM), Addison-Wesley, 1990.

Ali Rahimi has been developing software for over 15 years. He is currently working for Honeywell designing and developing software using OLE/COM and Windows NT. His interests include object-oriented programming, real-time and distributed systems, and AI. He has an M.S.E in Computer Science and Information Science from the University of Pennsylvania, Philadelphia. He can be reached at Ali.A.Rahimi@ftw1.honeywell.com.