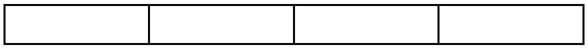
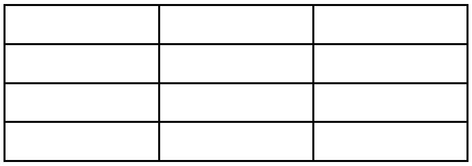
Built in Two-Dimensional Arrays

Arrays in C++ are not limited to a single dimension. Again, an illustration is useful:



One-dimensional array



Two-dimensional array

We have chosen to implement a two-dimensional array using built-in types:

```
int squarearray[5][5];             // Five rows and five columns
int baseballscores[2][9];          // Two rows and nine columns
```

The term *doubly subscripted* refers to the fact that elements in such an array must be referred to by both their row and column number. The entry in the first row and column of `baseballscores` is `baseballscores[0][0]`; the element in the last row and the last column is `baseballscores[1][8]`.

The idea of applying a process to every element of an array, which was previously demonstrated for one-dimension, can easily be extended to two-dimensions:

```
int numbers[10][20];
int m, n;
for (m=0; m<10; m++)
  for (n=0; n<20; n++)
    cin >> numbers[m][n];
```

This fragment's purpose is to read in a number for each of the 200 (10 x 20) elements of the array `numbers`.

Built in arrays may be passed as parameters in functions. They are by default referenced, so you are modifying the original. They have no default value, and cannot be resized, so remember to initialize, and use them carefully. An example of arrays in functions follows.

```
#include <iostream>

using namespace std;

void getData(int numbers[10][10]);

int main()
{
  int numbers[10][10];
  getData(numbers);
  .
  .
  return 0;
}
```

```
void getData(int numbers[10][10])
{
  cout<<"Please enter 100 numbers:";
  for (int i=0; i<10; i++)
  {
    for(int j=0; j<10; j++)
    {
       cin >> names[i][j];

    } // end of inner loop

  }   // end of outer loop

}     // end of function
```

Two-Dimensional Arrays Implemented as `vector` Objects

As with the built-in two-dimensional arrays in C++ two-dimensional vectors allow us to store more information in a single variable. But we get the benefits of being able to resize and find the size, which is not an option in the built-in version. Some sample arrays could be defined as follows:

```
#include <vector>
…
vector<vector<double> >stuff(7);    // A 7-element array of vectors
                                    // that are capable of holding
                                    // doubles
vector<vector<int> >numbers(10);    // A 10-element array of vectors
                                    // that are capable of holding
                                    // integers
```

To use the vector we must now resize each position to give the “matrix” its second dimension. This is accomplished by doing the following:

```
for(int x=0; x<stuff.size(); x++)
    stuff[x].resize(7);            // stuff is now 7 X 7
```

One of the benefits of the vector class, filling a vector on declaration, is lost when creating a two-dimensional vector. The only way to set a default value is manually. See example below.

```
for(int x=0; x<stuff.size(); x++)
   for(int y=0; y<stuff[y].size(); y++)
      stuff[x][y]=0;               // stuff is now a 7 X 7 grid
                                   // full of 0’s
```

Each element of a `matrix` can be referred to individually as represented above. Therefore input and output can written like this:

```
cin >> stuff[0][0];           // Gets input for the first entry
cout << stuff[4][4];          // Displays the value of the fifth row
                              // and fifth column
```

“Matrices” may be passed as parameters in functions. They should always be passed by reference to ensure efficient use of memory. Because you always pass by reference you are always capable of modifying the original. The solution to this problem is the const keyword. By adding the const keyword in front of the vector definition in the parameter list, the memory efficiency is maintained, while the vector is un-modifiable. An example of “matrix” in functions follows on the next page.

```
#include <iostream>
#include <vector>

using namespace std;

void getData(vector<vector<int> >&board);
void display(const vector< vector<int> > &board);

int main()
{
  vector< vector<int> > board(5);
  getData(board);
  display(board);
  .
  .
  return 0;
}

void getData(vector<vector<int> >&board)
{
  for(int x=0; x<5; x++)
   board[x].resize(5);
  for(int col=0; col<5; col++)
   for(int row=0; row<5; row++)
     board[col][row]=0;

   cout<<”Please enter each board position:”;
   for(int col=0; col<5; col++)
     for(int row=0; row<5; row++)
       cin >> board[col][row];

}

void display(const vector<vector<int> >&names)
{
  cout<<”Here is your board:\n”;
  for(int col=0; col<5; col++)
  {
    for(int row=0; row<5; row++)
      cout << board[col][row]<<” “;
    cout<<endl;
  }
}
```