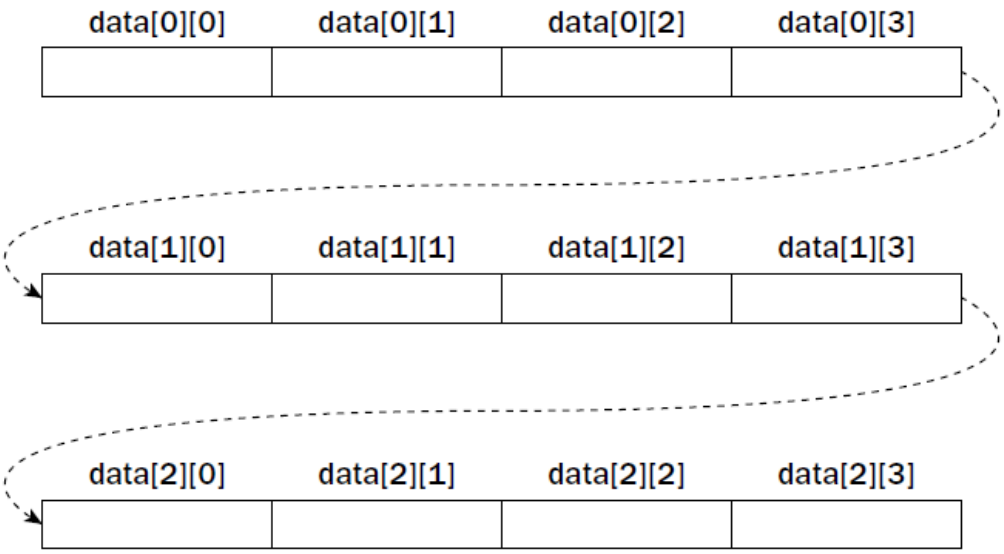
Multidimensional Arrays

The arrays that we have defined so far with one index are referred to as **one-dimensional** arrays. An array can also have more than one index value, in which case it is called a **multidimensional** array.

Arrays are stored in memory such that the rightmost index value varies most rapidly. Thus the array **data[3][4]** is three one-dimensional arrays of four elements each. The arrangement of this array is illustrated in Figure 4.

The elements of the array are stored in a contiguous block of memory, as indicated by the arrows in Figure 4. The first index selects a particular row within the array and the second index selects an element within the row.



The array elements are stored in contiguous locations in memory.

Figure 4

Initializing Multidimensional Arrays

To initialize a multidimensional array, you use an extension of the method used for a one-dimensional array. For example, you can initialize a two-dimensional array, data, with the following declaration:

int data[2][4] = {{ 1, 2, 3, 5 }, { 7, 11, 13, 17 }};

Thus, the initializing values for each row of the array are contained within their own pair of braces. Because there are four elements in each row, there are four initializing values in each group, and because there are two rows, there

are two groups between braces, each group of initializing values being separated from the next by a comma.

You can omit initializing values in any row, in which case the remaining array elements in the row are zero. For example:

```
int data[2][4] = {{ 1, 2, 3 }, { 7, 11 }};
```

We have spaced out the initializing values to show where values have been omitted. The elements **data[0][3]**, **data[1][2]**, and **data[1][3]** have no initializing values and are therefore zero.

If you wanted to initialize the whole array with zeros you could simply write:

```
int data[2][4] = {0};
```

If you are initializing arrays with even more dimensions, remember that you need as many nested braces for groups of initializing values as there are dimensions in the array.

Example 1:

```
int main()
{
        int i=0, j=0;
        int data[2][4] = {{ 1, 2, 3 }, { 7, 11 } };

        for(i=0; i<2; i++)
        {
                for(j=0; j<4; j++)
                        cout<<data[i][j]<<"\t";
                cout<<endl;
        }
        return 0;
}
```

Example 2:

```
// A nicely labeled multiplication table stored in array "table".

int main()
{
        int i=0, j=0, table[10][10];

        for(i=0; i<=9; i++)
```

```
            for(j=0; j<=9; j++)
                table[i][j]=(i+1) * (j+1);

        for(i=0; i<=9; i++)
        {
            for(j=0; j<=9; j++)
                cout<<table[i][j]<<"\t";
            cout<<endl;
        }
        return 0;
}
```

Example 3:

```
// Storing strings in an array.

int main()
{
    char colleges[6][80] = {    "Veterinary Medicine",
                                "Science",
                                "Engineering",
                                "Education",
                                "Agriculture",
                                "Administration & Economics"
                                };
    int dice = 0;
    cout << endl<< " Pick a college!"<< " Enter a number 1 .. 6: ";
    cin >> dice;
    if(dice >= 1 && dice <= 6) // Check input validity
        cout << endl // Output college name
        << "Your favorite college is "<< colleges [dice - 1];
    else
        cout << endl
        << "Sorry, you haven't got a lucky college.";
    cout << endl;
    return 0;
}
```