Chapter 35 – Multidimensional Arrays

1. Arrays of Two Indices

Arrays as presented in this text have been treated as good repositories for lists. That is, they have one dimension, length, and the elements are accessed via one index. This is very common, but is by no means the limit on the complexity of arrays.

If the element that comprises an array is also an array, the original array is said to have two dimensions and will require two indices to access a data-bearing element. In effect, the original array can be treated as consisting of rows and columns, much as a tax table or logarithm table. One index will indicate the row and one index will indicate the column.

The following is an example of the concepts of a one-dimension array and a two-dimension array. Both arrays contain the same data, which is two weeks of daily low temperatures. Array \( a \) stores the data in list fashion, one day after the other. Array \( b \) stores the data for the first week on row 0 and the data for the second week on row 1. Both arrays take up the same number of memory locations in RAM, but the organizations used give different advantages and disadvantages to the programmer.

<table>
<thead>
<tr>
<th>( a ) elements</th>
<th>( b ) elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>45</td>
<td>( 0 )</td>
</tr>
<tr>
<td>56</td>
<td>( 1 )</td>
</tr>
<tr>
<td>66</td>
<td>( 2 )</td>
</tr>
<tr>
<td>58</td>
<td>( 3 )</td>
</tr>
<tr>
<td>45</td>
<td>( 4 )</td>
</tr>
<tr>
<td>39</td>
<td>( 5 )</td>
</tr>
<tr>
<td>43</td>
<td>( 6 )</td>
</tr>
<tr>
<td>55</td>
<td>( 0 )</td>
</tr>
<tr>
<td>59</td>
<td>( 1 )</td>
</tr>
<tr>
<td>63</td>
<td>( 2 )</td>
</tr>
<tr>
<td>57</td>
<td>( 3 )</td>
</tr>
<tr>
<td>55</td>
<td>( 4 )</td>
</tr>
<tr>
<td>53</td>
<td>( 5 )</td>
</tr>
<tr>
<td>54</td>
<td>( 6 )</td>
</tr>
</tbody>
</table>

Array \( a \) can be declared in the usual manner. For example,

```c
int a[14];
```

However, array \( b \) can be declared in a variety of ways. The most obvious is with two indices indicating the number of rows and columns of elements. For example:

```c
class-or-type array[rows][columns];
```

Using this method, array \( b \) would be declared as follows:

```c
int b[2][7];
```
In order to find any element in array \( a \), only one index need be used. For example, \( a[10] \) has the value 57. Array \( b \), on the other hand, requires two indices. For example, the same value (57) in array \( b \) can only be access by specifying both the row and column, i.e. \( b[1][3] \).

In order for the average low over the two weeks of data to be calculated in either array, the arrays must be traversed. This is simpler in array \( a \), which requires only one counting loop to produce the required index. For example:

```c
int sum = 0;
for  (int x=0; x<14; x++) sum += a[x];
cout << "average low is " << sum / 14;
```

Finding the average low in array \( b \) is more complex since two counting loops are required to generate all of the indices needed. For example:

```c
int sum = 0;
for  (int x=0; x<2; x++) // for the row index
    for (int y=0; y<7; y++) // for the column index
        sum += a[x][y];
cout << "average low is " << sum / 14;
```

If the average low for each week is desired, roles will reverse and array \( a \) will require a more complex algorithm than array \( b \). Here is an example of finding the average low for each week using array \( a \).

```c
int week = 1; sum = 0;
for  (int x=0; x<14; x++) {
    sum += a[x];
    if ((x+1) % 7 ==0) {
        cout << "The average low for week " << week << is sum / 7;
        week++;
        sum = 0;
    }
}
```

To find the average temperature in each week of data in array \( b \), no complex logic to divine the end of the week is necessary. Instead, we can take advantage of the two counting loops that generate the indices needed for traversal of an array of two dimensions by adding a compound statement to the outermost loop. For example:

```c
int sum;
for  (int x=0; x<2; x++) {
    sum = 0;
    for (int y=0; y<7; y++) sum += a[x][y];
    cout << "The average low for week " << x << is sum / 7;
}
```
Two-dimensional (or more) arrays can be initialized upon declaration. For two-dimensional arrays, this has the form:

\[ \text{class array-name[][] = \{\{value, value\}, \{value, value\}, ..., \{value, value\}\};} \]

For example:

\[ \text{int a[][] = \{\{1,100\}, \{2,200\}, \{3,300\}, \{4,400\}\};} \]

**Programming Assignment 35.1**

Create a two-dimensional integer array of 4 rows and 10 columns. Fill the array with random integers. Output the array in rows and columns, evenly spaced. At the end of each row, output the sum of the row. At the bottom of each column, output the sum of the column. Output the sum of the entire array at the intersection of the sums of the rows and the sums of the columns.

**Programming Assignment 35.2**

Using a copy of the program of 35.1, replace the outputs at the end of each row and bottom of each column with the average of the row or column. Output the average of the entire array at the intersection of the averages of the rows and the averages of the columns.

**Programming Assignment 35.3**

Using a copy of the program of 35.1, replace the outputs at the end of each row with the minimum value of the row. Replace the outputs at the bottom of each column with the maximum value of the column.

**Programming Assignment 35.4**

Multiplication of two 2-dimensional matrices produces another matrix of 2 dimensions. 2-dimensional matrices are multiplied with the following formula.

\[ C_{ij} = \sum_{k=0}^{n-1} A_{ik} \cdot B_{kj} \]

In terms of C++, this states that the element \(C[i][j]\) in the target (answer) matrix is the sum of

\[ A[i][0] \cdot B[0][j] + A[i][1] \cdot B[1][j] + ... + A[i][n-1] \cdot B[n-1][j] \]

In this formula, \(n\) is the number of columns in matrix \(A\) or the number of rows in matrix \(B\), whichever is smaller. The index \(i\) has an upper limit of the smaller of the number of rows in matrix \(A\) or \(B\). The index \(j\) has an upper limit of the smaller of the number of columns in matrix \(A\) or \(B\). Thus matrix \(C\) will have the number of rows that are in the matrix \(A\) or \(B\), whichever is smaller. The matrix \(C\) will also have the number of columns that the
matrix $A$ or $B$ has, whichever is smaller. In order to perform matrix multiplication on $A$ and $B$ to produce matrix $C$, three counter loops are required. The outermost produces $i$, the middle loop produces $j$ and the innermost loop is responsible for $k$ (which is limited by $n$).

Write a program to multiply the following two matrices to produce a third matrix. Output the result in rows and columns.

$$
\begin{bmatrix}
3 & 5 & 7 \\
4 & 1 & 4 \\
3 & 2 & 8
\end{bmatrix} \cdot 
\begin{bmatrix}
9 & 4 & 1 \\
7 & 7 & 4 \\
6 & 5 & 6
\end{bmatrix}
$$

**Programming Assignment 35.5**
Rewrite the program of 35.4, but with the order of the matrices to be multiplied reversed, i.e. multiply $B$ times $A$ to produce $C$. Is the answer the same? What does this indicate about the rules of matrix multiplication compared to those of multiplication of integer numbers or real numbers?

**Programming Assignment 35.6**
Write a program to multiply the following two matrices. Output the result in rows and columns.

$$
\begin{bmatrix}
3 & 5 \\
4 & 1 \\
3 & 2
\end{bmatrix} \cdot 
\begin{bmatrix}
9 & 4 & 1 \\
7 & 7 & 4
\end{bmatrix}
$$
2. **Alternative Declarations of 2 Dimensional Arrays**

A problem with arrays or more than one dimension is that they cannot be passed to free functions as parameters in a simple way. A way around this limitation is to declare a type, struct or class of an array to serve as the unit out of which to build final array. Here are some example fragments using two dimension arrays built from typedefs, structs and classes, complete with function calls and traversals:

**Example 1: Typedef with Passing of Entire Array**

```c
typedef int weektype[7];

int twoWeekAvg(weektype a[7]){  
    int sum=0;
    for (int x=0; x<2; x++)
        for (int y=0; y<7; y++) sum += a[x][y];
    return sum / 14;
};

int main( ) {
    weektype b[2];
    :
    :
    cout << "The average low temp is " << twoWeekAvg(b);
    :
    :
```

**Example 2: Typedef with Passing of One Row**

```c
typedef int weektype[7];

int WeekAvg(int a[7]){
    int sum=0;
    for (int x=0; x<7; x++) sum += a[x];
    return sum / 7;
};

int main( ) {
    weektype b[2];
    :
    :
    cout << "The average low temp is "
    << (WeekAvg(b[0]) + WeekAvg(b[1]) / 2);
    :
    :
```
Example 3: Passing an array of Structs

```cpp
struct weektype {
    int array[7];
};

int twoWeekAvg(weektype a[]) {
    int sum = 0;
    for (int x = 0; x < 2; x++)
        for (int y = 0; y < 7; y++) sum += a[x].array[y];
    return sum / 14;
};

int main() {
    weektype b[2];
    cout << "The average low temp is " << twoWeekAvg(b);
};
```
Example 4: Class

```cpp
class weektype {
public:
    weektype( );
    ...;
    int returnAvg( );
    ...;
private:
    int array[7];
    ...;
};

int weektype::returnAvg( ) {
    int sum=0;
    for (int x=0; x<7; x++) sum += array[x];
    return sum / 7;
}

int main() {
    weektype b[2];
    ...;
    cout << "The average low temp is " << (b[0].returnAvg() + b[1].returnAvg()) / 2;
}
```

Arrays of simple types of more than one dimension can be passed as a parameter if the second dimension length can be specified in the formal parameter list of the function, which may or may not be possible. For example:

```cpp
void example_function(int a[][100]) { ... }
```
```cpp
int array[35][100];
```
```cpp
example_function(array);
```
Programming Assignment 35.7

Complete the member functions of the following class definition. Use the resulting class to implement a grade book for a class (school class, not C++ class) in a C++ program. The grade book should be loaded from a file when the program starts and returned to the file when the program finishes. Assume that all grades carry the same weight.

```cpp
class student {

public:
    student( );    // default constructor
    student(string initFName,
             string initLName); // student constructor
    string getFName( );  // returns first name of student
    string getLName( );  // returns last name of student
    int getNumberOfGrades(); // returns the number of grades recorded
    int getGrade(int gradeIndex); // returns grade indicated by index
    void insertGrade(int initGrade); // inserts a new numeric grade
    void deleteGrade(int gradeIndex); // deletes indicated grade
    int getAverage( );   // returns average of student

private:
    string firstName, lastName;
    int numberOfGrades;
    int grades[20];
};
```

Programming Assignment 35.8

Design a class called gradebook that depends on the above class to maintain a gradebook for a class. The number of the class, the number of students in the class and the file in which to store the class should all be private data members that the can be set by calls to member functions.
3. Arrays of More than Two Dimensions

Each dimension of an array requires its own index. Just as arrays of two dimensions name their indices row and column, arrays of three dimensions, often have their indices referred to as page, row and column. Arrays of four dimensions often have their indices referred to as volume, page, row and column. Arrays of more than four dimensions are not often used, but arrays can be declared of any number of dimensions. Here are some examples of declarations of arrays and their traversals of more than two dimensions:

**Example of a 3-dimension Array**

```c
float a[45][17][100];  // array of three dimensions

for (int page=0; page<45; page++)
    for (int row=0; row<17; row++)
        for (int col=0; col<100; col++)
            cout << a[page][row][col] << "\t";
```

**Example of a 5-dimension Array**

```c
int c[2][3][4][5][6];  // array of five dimensions

for (int library=0; library<2; library++)
    for (int vol=0; vol<3; vol++)
        for (int page=0; page<4; page++)
            for (int row=0; row<5; row++)
                for (int col=0; col<6; col++)
                    cout << a[library][vol][page][row][col] << "\t";
```

**Programming assignment 35.9**

Create an array of bools of 3 dimensions consisting of 3 pages, two rows and 5 columns. Fill the array with random Boolean values. Output values in the array by page such that each page appears as a table of evenly spaced columns. The end of each page should indicate the number of true values on the page.