The arrays we have been looking at until now have been linear, or sequential, lists. One-dimensional. A vector is a one-dimensional array. The word array usually (but not always) refers to a two- or more dimensional array. It is a vector whose elements are vectors. Multi-dimensional arrays are useful for organizing data which is dependent on two or more variables.

If X is a matrix of numbers, it is said to be 2-dimensional. A 2-dimensional array can be thought of as a table of data items, with rows and columns. The 3rd element in the 2nd row is referred to as X(2,3).

Example:

```cpp
#include <iostream>

int main(){
    int x[3][5];
    //read the 2-dimensional array
    cout << "Enter 15 integers, 5 per row:\n";
    for (int i=0; i<3; i++){
        cout << "Row " << i << " : ";
        for (int j=0; j<5; j++)
            cin >> x[i][j];
    }
    //print the 2-dimensional array
    cout << "Your data matrix contains the following:";
    for (int i=0; i<3; i++){
        for (int j=0; j<5; j++)
            cout << "  " << x[i][j];
        cout << endl;
    }
    cout << "Press any key to close console window.";
    char c; cin >> c;
    return 0;
}
```

Output:

```
Enter 15 integers, 5 per row:
Row 0: 44 33 87 88 100
Row 1: 22 33 45 75 66
Row 2: 4 6 66 44 110

Your data matrix contains the following:
  44  33  87  88  100
  22  33  45  75  66
   4  6  66  44  110

Press any key to close console window.
```
A 3-dimensional array would be declared with three dimensions, e.g.:
```java
int x[25][10][5];
```
and three for loops with three index variables (e.g. i, j, k) would be used for processing it. A 3-
dimensional array can be thought of as a collection of tables, like pages in a book.

NOTE: When a multidimensional array is an argument to a function, only the size of the first
dimension can be missing. The others must be specified in the parameter list, e.g.,
```java
int sumMatrix(int x[][100], int rows, int cols);
```

Examples:
Two-level array – Rating Table – Insurance rates by age and job classification:

<table>
<thead>
<tr>
<th>Age</th>
<th>Class 1</th>
<th>Class 2</th>
<th>Class 3</th>
<th>Class 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>18-34</td>
<td>23.50</td>
<td>25.25</td>
<td>27.05</td>
<td>52.90</td>
</tr>
<tr>
<td>35-39</td>
<td>24.00</td>
<td>35.75</td>
<td>27.55</td>
<td>53.40</td>
</tr>
<tr>
<td>40-44</td>
<td>24.60</td>
<td>36.35</td>
<td>28.15</td>
<td>54.00</td>
</tr>
<tr>
<td>45-49</td>
<td>25.30</td>
<td>37.05</td>
<td>28.85</td>
<td>54.70</td>
</tr>
<tr>
<td>50-54</td>
<td>26.30</td>
<td>38.05</td>
<td>29.85</td>
<td>55.70</td>
</tr>
<tr>
<td>55-59</td>
<td>28.00</td>
<td>39.75</td>
<td>31.55</td>
<td>57.40</td>
</tr>
</tbody>
</table>

Three-level array – Rating Table – Insurance rates by age, sex, and job classification:

<table>
<thead>
<tr>
<th>Age</th>
<th>Men</th>
<th>Women</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class 1</td>
<td>Class 2</td>
<td>Class 1</td>
</tr>
<tr>
<td>18-34</td>
<td>23.50</td>
<td>25.25</td>
</tr>
<tr>
<td>35-39</td>
<td>24.00</td>
<td>35.75</td>
</tr>
<tr>
<td>40-44</td>
<td>24.60</td>
<td>36.35</td>
</tr>
<tr>
<td>45-49</td>
<td>25.30</td>
<td>37.05</td>
</tr>
<tr>
<td>50-54</td>
<td>26.30</td>
<td>38.05</td>
</tr>
<tr>
<td>55-59</td>
<td>28.00</td>
<td>39.75</td>
</tr>
</tbody>
</table>
Sparse arrays

e.g.:

```
0 0 0 0 1 0 0 2 0 0
0 1 0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 4 0 0 0 0 0 0 0
0 0 0 0 0 0 2 0 0 0
0 0 0 0 0 0 0 0 0 0
2 1 0 0 0 0 0 0 0 0
```

The problem here is that there is a lot of wasted space. Consider, a 1000 by 1000 array, 1 million elements and, say, 1500 are nonzero. What is a more efficient way to represent this data?

How about an array of triplets, or 3 vectors:

```
Row[i]  Column[i]  Value[i]
1       5          1
1       8          2
2       2          1
3       1          1
5       4          4
6       8          2
8       1          2
8       2          1
```

Or – even better – a record (struct) or object.

Practice Assignment: Read in triplets (as records). Print out original matrix. Print our transpose. Do not at any time store the entire 8 x 10 array in main memory.