1. `int x[2][3];`

6 elements organized into: 2 horizontal rows
3 vertical columns

<table>
<thead>
<tr>
<th>column:</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>row:</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>10</td>
<td>30</td>
<td>50</td>
</tr>
<tr>
<td>1</td>
<td>20</td>
<td>40</td>
<td>60</td>
</tr>
</tbody>
</table>

2. To reference an element:

```
array name[row subscript][column subscript]
```

Example:
```
cout << x[1][2];  // outputs 60
```

3. Contrast with 1-d array:
   a. image
   b. number of subscripts required to reference an element

4. Access elements in Column Order

```
x[0][0] = 10;  // column 0
x[1][0] = 20;
x[0][1] = 30;  // column 1
x[1][1] = 40;
x[0][2] = 50;  // column 2
x[1][2] = 60;
```

5. Set data in column order

```
int r, c, num = 0;
for (c = 0; c < 3; c++)  // for each column
    for (r = 0; r < 2; r++)  // for each row
        { 
            num += 10;
            x[r][c] = num;
        }
```
6. Output in row order

```cpp
int r, c;
for (r = 0; r < 2; r++) // for each row
{
    for (c = 0; c < 3; c++) // for each column
    {
        cout << setw(4) << x[r][c];
        cout << endl; // newline for next row
    }
}
```

same as:

```
// Pass 1 of outer loop
cout << setw(4) << x[0][0] << setw(4) << x[0][1] << setw(4) << x[0][2] << '
';
// Pass 2 of outer loop
cout << setw(4) << x[1][0] << setw(4) << x[1][1] << setw(4) << x[1][2] << '
';
```

7. Two-dimensional arrays are stored in row order.

```
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>x[0][0]</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>x[0][1]</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>x[0][2]</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>x[1][0]</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>x[1][1]</td>
<td>40</td>
<td></td>
</tr>
<tr>
<td>x[1][2]</td>
<td>60</td>
<td></td>
</tr>
</tbody>
</table>
```
// Produce real estate sales report for 4 agents
// using 1-d and 2-d arrays

#include <iostream> // cout
#include <fstream> // ifstream
#include <iomanip> // setw()
#include <cstdlib> // exit()

using namespace std;

const int AGENTS = 4, MONTHS = 5;

int main() // Inline code
{
    ifstream in_file; // Declare input file stream variable
    in_file.open("a:agents.dat", ios_base::in);
    // Associate stream with the file
    /* The following statement is equivalent to the above two.
    in_file.open("a:agents.dat", ios_base::in); */
    if (!in_file.is_open())
    {
        cout << "Could NOT open file.\n";
        exit(1);
    }
    int id[AGENTS], sales[AGENTS][MONTHS]; // Declare arrays
    int r, c;
    for (r = 0; r < AGENTS; r++) // Load the arrays
    {
        in_file >> id[r]; // Load agent id
        for (c = 0; c < MONTHS; c++)
            in_file >> sales[r][c]; // Load agent's sales data
    }
    in_file.close(); // Disassociate stream from the file
    // Done automatically when stream variable goes out of scope
```c
int agent_sum[AGENTS];

for (r = 0; r < AGENTS; r++)
    // Sum by agents (rows)
    
    agent_sum[r] = 0;

    for (c = 0; c < MONTHS; c++)
        agent_sum[r] += sales[r][c];


sales

<table>
<thead>
<tr>
<th>id</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>agent_sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1234</td>
<td>2</td>
<td>4</td>
<td>5</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>2531</td>
<td>6</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>2</td>
<td>8730</td>
<td>0</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>6021</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>

month_sum

| 0 | 1 | 2 | 3 | 4 |

int month_sum[MONTHS];

for (c = 0; c < MONTHS; c++)
    // Sum by months (columns)
    
    month_sum[c] = 0;

    for (r = 0; r < AGENTS; r++)
        month_sum[c] += sales[r][c];


/* Alternative to lines 43-64
   NOTE THAT LINES 67 - 78 IS A COMMENT

int agent_sum[AGENTS] = {0};
int month_sum[MONTHS] = {0};

for (r = 0; c < AGENTS; r++)
    for (c = 0; c < MONTHS; c++)
        
        agent_sum[r] += sales[r][c];

        month_sum[c] += sales[r][c];

*/
cout << "RE Sales Report

Agent Sales by Month:

Id AUG SEP OCT NOV DEC Totals

"; // Output header lines

for (r = 0; r < AGENTS; r++) // Output detail lines
{
    cout << setw(8) << id[r];
    for (c = 0; c < MONTHS; c++)
        cout << setw(6) << sales[r][c];
    cout << setw(8) << agent_sum[r] << endl;
}

cout << "\nTotals "; // Output footer lines

for (c = 0; c < MONTHS; c++)
    cout << setw(6) << month_sum[c];
    cout << endl;
return 0;
Input File Listing:

1234 2 4 5 1 2
2531 6 3 1 2 7
8730 0 5 4 3 1
6021 3 1 2 2 4

Sequential file having 4 records, 6 fields per record
Fields are the agent's id number and sales data for a 5-month period.

Array Images at time of output:

<table>
<thead>
<tr>
<th>id</th>
<th>sales</th>
<th>agent_sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1234</td>
<td>0 2 4 5</td>
<td>0 14</td>
</tr>
<tr>
<td>1 2531</td>
<td>1 6 3 1</td>
<td>1 19</td>
</tr>
<tr>
<td>2 8730</td>
<td>2 0 5 4</td>
<td>2 13</td>
</tr>
<tr>
<td>3 6021</td>
<td>3 3 1 2</td>
<td>3 12</td>
</tr>
</tbody>
</table>

month_sum

<table>
<thead>
<tr>
<th></th>
<th>11</th>
<th>13</th>
<th>12</th>
<th>8</th>
<th>14</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>11</td>
<td>13</td>
<td>12</td>
<td>8</td>
<td>14</td>
</tr>
</tbody>
</table>

Output:

RE Sales Report

<table>
<thead>
<tr>
<th>Agent Id</th>
<th>AUG</th>
<th>SEP</th>
<th>OCT</th>
<th>NOV</th>
<th>DEC</th>
<th>Agent Totals</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>2</td>
<td>4</td>
<td>5</td>
<td>1</td>
<td>2</td>
<td>14</td>
</tr>
<tr>
<td>2531</td>
<td>6</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>7</td>
<td>19</td>
</tr>
<tr>
<td>8730</td>
<td>0</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>1</td>
<td>13</td>
</tr>
<tr>
<td>6021</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>12</td>
</tr>
</tbody>
</table>

Totals 11 13 12 8 14

System Chart:

Disk → RE data → C++ Program → Monitor
// Display the most recent sales data for selected real estate agents

// Sequential Search:
// walk through an array looking (searching) for
// a given value (target)

#include <fstream>      // for ifstream
#include <iostream>     // for cin, cout
#include <cstdlib>      // for exit()
#include <iomanip>      // for setw()

using namespace std;

int main()                 // Inline code
{
    ifstream in_file;
    in_file.open("a:agents.dat", ios_base::in);
    if (!in_file.is_open())
    {
        cout << "Could NOT open output file.\n";
        exit(1);
    }

    const int AGENTS = 4, MONTHS = 5;
    int id[AGENTS], sales[AGENTS][MONTHS];       // Declare arrays
    int r, c;

    for (r = 0; r < AGENTS; r++)                  // Load the arrays
    {
        in_file >> id[r];                       // Load agent id
        for (c = 0; c < MONTHS; c++)
            in_file >> sales[r][c];            // Load agent's sales data
    }

    in_file.close();                             // Finished with file

    cout << "This program will search a real estate agent data base\n" << "and output their sales data for the last " << MONTHS << " months.\n\n";

    int agent_target;
    bool found;
while (true) // Loop for each agent id number
{
    cout << "Enter an agent's id (Negative to QUIT): ";
    cin >> agent_target;
    if (agent_target < 0)
        break; // Exit outer loop
    r = 0;
    found = false;

    // Exit Search Loop if (r >= AGENTS || found)
    while (r < AGENTS && !found) // Search loop
    {
        if (agent_target == id[r])
            found = true; // Found target
        else
            r++;
    }

    if (found) // Match or not???
    {
        cout << "Agent number " << agent_target // A match
            << " has the following sales data:");
        for (c = 0; c < MONTHS; c++)
            cout << setw(3) << sales[r][c];
        cout << endl;
    }
    else
        cout << "Agent number " << agent_target << " does not exist.
"; // End of outer loop

    cout << "\nEnd of RE Agent Search Program\n";
return 0;
}

Array Images for search logic:

\begin{tabular}{|c|c|c|c|c|c|}
\hline
agent_target & id & sales & 0 & 1 & 2 & 3 & 4 \\
\hline
0 & 1234 & 0 & 2 & 4 & 5 & 1 & 2 \\
1 & 2531 & 1 & 6 & 3 & 1 & 2 & 7 \\
2 & 8730 & 2 & 0 & 5 & 4 & 3 & 1 \\
3 & 6021 & 3 & 3 & 1 & 2 & 2 & 4 \\
\hline
\end{tabular}

while (r < AGENTS && !found)
Sample Run:
This program will search a real estate agent data base
and output their sales data for the last 5 months.

Enter an agent's id (Negative to QUIT): 8730
Agent number 8730 has the following sales data: 0 5 4 3 1
Enter an agent's id (Negative to QUIT): 9
Agent number 9 does not exist.
Enter an agent's id (Negative to QUIT): 1234
Agent number 1234 has the following sales data: 2 4 5 1 2
Enter an agent's id (Negative to QUIT): 6021
Agent number 6021 has the following sales data: 3 1 2 2 4
Enter an agent's id (Negative to QUIT): 2531
Agent number 2531 has the following sales data: 6 3 1 2 7
Enter an agent's id (Negative to QUIT): 1234
Agent number 1234 has the following sales data: 2 4 5 1 2
Enter an agent's id (Negative to QUIT): -1

End of RE Agent Search Program

Listing of agents.dat:
1234 2 4 5 1 2
2531 6 3 1 2 7
8730 0 5 4 3 1
6021 3 1 2 2 4

Array Images:

<table>
<thead>
<tr>
<th>id</th>
<th>sales</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0 1 2 3 4</td>
</tr>
<tr>
<td>0</td>
<td>1234</td>
</tr>
<tr>
<td>1</td>
<td>2531</td>
</tr>
<tr>
<td>2</td>
<td>8730</td>
</tr>
<tr>
<td>3</td>
<td>6021</td>
</tr>
<tr>
<td>0</td>
<td>2 4 5 1 2</td>
</tr>
<tr>
<td>1</td>
<td>6 3 1 2 7</td>
</tr>
<tr>
<td>2</td>
<td>0 5 4 3 1</td>
</tr>
<tr>
<td>3</td>
<td>3 1 2 2 4</td>
</tr>
</tbody>
</table>
// Process grades for multiple students using inline code

#include <iostream> // for cin, cout
#include <iomanip> // for setw()

using namespace std;

int main() // Inline code
{
    const int SIZE = 10; // Maximum number of students
    int id[SIZE], // Declare id and grade arrays
        grade[SIZE], i = 0;

    cout << "You can process a maximum of " << SIZE << " students.\n";
    cout << "Enter first id (q to QUIT): ";

    while (i < SIZE && cin >> id[i]) // Short Circuit Principle
    {
        while (id[i] <= 0) // Edit id numbers
            { // Must be positive
                cout << "Id must be positive; re-enter: ";
                cin >> id[i];
            }

        do // Edit grades
            { // Must be in range of 0 to 100, inclusive
                cout << "Enter grade in range of 0 to 100, inclusive: ";
                cin >> grade[i];
            }
        while (grade[i] < 0 || grade[i] > 100);

        i++;
        if (i < SIZE)
            cout << "Enter next id (q to QUIT): ";
    } // End of outer loop

    int student_count = i; // Set actual student count
    int grade_sum = 0;

    for (i = 0; i < student_count; i++)
        grade_sum += grade[i]; // Accumulate sum of grades

    int grade_average = grade_sum / student_count; // Compute average grade

    cout << "Grade Report\n";
    cout << fixed << setprecision(3);

    for (i = 0; i < student_count; i++)
        cout << setw(9) << id[i] << setw(8) << grade[i] // Output details
            << setw(10) << (float (grade[i]) / grade_average) << endl;

    cout << "Average grade: " << grade_average << endl; // Output footers
    return 0;
}
Run 1:
You can process a maximum of 10 students.
Enter first id (q to QUIT): 332450
Enter grade in range of 0 to 100, inclusive: 93
Enter next id (q to QUIT): 521444
Enter grade in range of 0 to 100, inclusive: 100
Enter next id (q to QUIT): 704263
Enter grade in range of 0 to 100, inclusive: 61
Enter next id (q to QUIT): 881233
Enter grade in range of 0 to 100, inclusive: 79
Enter next id (q to QUIT): 902361
Enter grade in range of 0 to 100, inclusive: 89
Enter next id (q to QUIT): 998876
Enter grade in range of 0 to 100, inclusive: 43
Enter next id (q to QUIT): 878780
Enter grade in range of 0 to 100, inclusive: 85
Enter next id (q to QUIT): 111222
Enter grade in range of 0 to 100, inclusive: 96
Enter next id (q to QUIT): 654321
Enter grade in range of 0 to 100, inclusive: 75
Enter next id (q to QUIT): 101020
Enter grade in range of 0 to 100, inclusive: 92

<table>
<thead>
<tr>
<th>Id</th>
<th>Grade</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>332450</td>
<td>93</td>
<td>1.148</td>
</tr>
<tr>
<td>521444</td>
<td>100</td>
<td>1.235</td>
</tr>
<tr>
<td>704263</td>
<td>61</td>
<td>0.753</td>
</tr>
<tr>
<td>881233</td>
<td>79</td>
<td>0.975</td>
</tr>
<tr>
<td>902361</td>
<td>89</td>
<td>1.099</td>
</tr>
<tr>
<td>998876</td>
<td>43</td>
<td>0.531</td>
</tr>
<tr>
<td>878780</td>
<td>85</td>
<td>1.049</td>
</tr>
<tr>
<td>111222</td>
<td>96</td>
<td>1.185</td>
</tr>
<tr>
<td>654321</td>
<td>75</td>
<td>0.926</td>
</tr>
<tr>
<td>101020</td>
<td>92</td>
<td>1.136</td>
</tr>
</tbody>
</table>

Average grade: 81
Run 2:
You can process a maximum of 10 students.
Enter first id (q to QUIT): 0
Id must be positive; re-enter: -1
Id must be positive; re-enter: 123456
Enter grade in range of 0 to 100, inclusive: 101
Enter grade in range of 0 to 100, inclusive: 102
Enter grade in range of 0 to 100, inclusive: -1
Enter grade in range of 0 to 100, inclusive: 99
Enter next id (q to QUIT): 654321
Enter grade in range of 0 to 100, inclusive: 90
Enter next id (q to QUIT): q

Grade Report

<table>
<thead>
<tr>
<th>Id</th>
<th>Grade</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>123456</td>
<td>99</td>
<td>1.053</td>
</tr>
<tr>
<td>654321</td>
<td>90</td>
<td>0.957</td>
</tr>
</tbody>
</table>

Average grade: 94
The greatest difficulty in programming is expressing the logic clearly. Even the simplest program can be a confusing mess.

Example: Given nonzero coordinates, \( x \) and \( y \), determine the quadrant.

```
    quadrant 2          y            quadrant 1
                        x is negative   x is positive
                        y is positive   y is positive

                        x is negative
                        y is negative
                        x is positive
                        y is negative

    quadrant 3          quadrant 4
```

```c
double x, y;
int quad;

if (x > 0.0)
goto label10;
goto label30;

label10: if (y > 0.0)
goto label20;
quad = 4;
goto label50;

label20: quad = 1;
goto label50;

label30: if (y > 0.0)
goto label40;
quad = 3;
goto label50;

label40: quad = 2;
label50: ...
```

In the above code, each statement is easy to understand, but the overall code is confusing due to the excessive use of the `goto`. Although the `goto` is available in C++, it is not needed.

The syntax is: `goto label;`

The above statement will unconditional transfer control to the statement having the label. In fact, the `goto` is sometimes known as the "unconditional goto".
Using the following structured approach, the logic is straight forward.

```c
if ( x > 0.0)
    if (y > 0.0)
        quad = 1;
    else
        quad = 4;
else
    if (y > 0.0)
        quad = 2;
    else
        quad = 3;
```

**Structured Programming:** method of programming which can result in

1. improved programmer productivity
2. improved reliability
3. programs that are easy to maintain

This method was started by a paper presented by two mathematicians, Corrado Bohm and Giuseppe Jacopini, at a 1964 International Colloquium on Algebraic Linguistics and Automata Theory in Israel.

Their paper proved a theorem which states:

A proper program, one which has:

- a. one entry
- b. one exit
- c. no infinite loops
- d. no unreachable code

can be coded using a maximum of 3 basic control structures:

1. sequence
2. selection
3. iteration