1. \texttt{int x[2][3];}

   6 elements organized into:
   \begin{itemize}
   \item 2 horizontal rows
   \item 3 vertical columns
   \end{itemize}

   \begin{center}
   \begin{tabular}{c|c|c}
   column: & 0 & 1 & 2 \\
   \hline
   row: & 
   0 & 10 & 30 & 50 \\
   1 & 20 & 40 & 60 \\
   \end{tabular}
   \end{center}

2. To reference an element:

   \texttt{array name[row subscript][column subscript]}

   Example:
   \texttt{cout << x[1][2]; // outputs 60}

3. Contrast with 1-d array:
   \begin{itemize}
   \item image
   \item number of subscripts required to reference an element
   \end{itemize}

4. Access elements in Column Order

   \begin{verbatim}
   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;
   \end{verbatim}

5. Set data in column order

   \begin{verbatim}
   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;
       }
   \end{verbatim}
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:

```cpp
// 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.

```
x[0][0]       10
x[0][1]       30
x[0][2]       50
x[1][0]       20
x[1][1]       40
x[1][2]       60
```
// REsalesrpt: 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 iostream on most systems.
#include <cstdlib> // exit()

using namespace std;

const int AGENTS = 4, MONTHS = 5;

int main()
{
    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.
        ifstream in_file("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
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];
}

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];
}

// Output report
cout << " RE Sales Report

Agent Sales by Month: Agent
" id AUG SEP OCT NOV DEC Totals

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</td>
<td>2 4 5 1 2</td>
<td>0 14</td>
</tr>
<tr>
<td>1</td>
<td>6 3 1 2 7</td>
<td>1 19</td>
</tr>
<tr>
<td>2</td>
<td>0 5 4 3 1</td>
<td>2 13</td>
</tr>
<tr>
<td>3</td>
<td>3 1 2 2 4</td>
<td>3 12</td>
</tr>
</tbody>
</table>

Output:

<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>
<tr>
<td>Totals</td>
<td>11</td>
<td>13</td>
<td>12</td>
<td>8</td>
<td>14</td>
<td></td>
</tr>
</tbody>
</table>

System Chart:

Disk → RE data → C++ Program → RE report → Monitor
1. **character**: letter, digit, or special character

2. **field**: one or more characters treated as a unit
   (1 field corresponds to a data item)

3. **record**: one or more fields treated as a unit
   (1 record for each subject)

4. **file**: one or more records treated as a unit

5. **Input File Example**

   Record 1 → LILY^F^18
   Record 2 → JOSE^M^19
   Record 3 → ALICE^F^18
   Record 4 → TIM^M^20

   1 file consisting of 4 input records, each having 3 fields

   **Sequential File**: records are processed sequentially, starting with record 1, then record 2, and so on

   **End Of File (eof)**: end of data

   **eof Record or trailer record**: last record in the file that marks the end of the file

   eof Record → END^X^-1

6. **Output File Example**

   Record 1 → LILY  18
   Record 2 → JOSE  19
   Record 3 → ALICE 18
   Record 4 → TIM   20

   1 file consisting of 4 output records, each having 2 fields
1. **stream**: stream of bytes or stream of characters
   
   **input stream**: bytes going from input device to program
   
   **output stream**: bytes going from program to output device
   
   An I/O stream can be connected to the keyboard (cin), monitor (cout), or a disk file.

2. **Disk File**: unit of stored information on disk; 1 or more records stored on disk
   
   Program File or Data File
   
   Reason for saving data on disk:
   a. Can store data permanently
   b. Can easily store large quantities of data
   c. Data portability
   d. Backups

3. **stream variable**: object variable that deals with I/O
   
   Stream variable is analogous to counter variable or accumulator variable.
   
   cin and cout are streams; object variables declared in iostream
   
   ```c++
   istream cin;
   ostream cout;
   ```

4. To declare a stream variable that is connected to a file:
   
   ```c++
   ifstream in_file;    // Input File stream variable
   ofstream out_file;   // Output File stream variable
   ```
   
   ifstream is a class which handles reading from a disk file.
   ofstream is a class which handles writing to a disk file.
   
   `#include <fstream>`
1. Syntax for call to open():
   `stream variable.open(external file id, file mode);`
   - internal file id: file id used by the operating system
   - DOS: "a:agents.dat"
   - DOS: "c:\ccsf\cs110b\agents.dat"
   - UNIX: "/users/arule/cs110b/agents"

   Opening a file will
   a. Establish a buffer
   b. Set the file pointer
   c. Connect the internal file id to the external file id

2. Syntax for call to is_open():
   `stream variable.is_open() // returns true if file could be opened`

   Reasons for open() to fail: disk drive failure, no disk and disk full

3. `void exit(int status); // Prototype`

   Immediately exits the C++ program, regardless of the location of exit()
   1 as the argument indicates failure

   `#include <cstdlib> // Contains the prototype`

4. Syntax for call to close():
   `stream variable.close();`

   Closing a file will
   a. Deallocate the buffer
   b. Disconnect the internal file id from the external file id

   You should close a file as soon as the program finishes with it.

   A file is automatically closed when the stream variable goes out of scope.
// REsalesrpt2: Produce real estate sales report for 4 agents
// using 1-d and 2-d arrays
// Modular version
#include <iostream>  // for cout
#include <iomanip>   // for setw()
using namespace std;

const int AGENTS = 4, MONTHS = 5;

void compute_sums(int asum[], int msum[], const int sales[][MONTHS]);
void output_report(const int id[], const int asum[],
                   const int msum[], const int sales[][MONTHS]);

int main()
{
    int id[AGENTS] = {1234, 2531, 8730, 6021}, // Declare arrays
    sales[AGENTS][MONTHS] = {{2, 4, 5, 1, 2},
        {6, 3, 1, 2, 7},
        {0, 5, 4, 3, 1},
        {3, 1, 2, 2, 4}}; // Inner {} are optional
    int agent_sum[AGENTS] = {0}; // Partial initialization
    int month_sum[MONTHS] = {0};

    compute_sums(agent_sum, month_sum, sales);
    output_report(id, agent_sum, month_sum, sales);

    return 0;
}

void compute_sums(int asum[], int msum[], const int sales[][MONTHS])
{
    int r, c;
    for (r = 0; r < AGENTS; r++)
        for (c = 0; c < MONTHS; c++)
          {
            asum[r] += sales[r][c]; // Sum by agents (rows)
            msum[c] += sales[r][c]; // Sum by months (columns)
          }
}

<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></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2531</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>8730</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>6021</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sales</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 2</td>
<td>4</td>
<td>5</td>
<td>1</td>
<td>2</td>
<td></td>
<td>0 0</td>
</tr>
<tr>
<td>1 6</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>7</td>
<td></td>
<td>1 0</td>
</tr>
<tr>
<td>2 0</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>1</td>
<td></td>
<td>2 0</td>
</tr>
<tr>
<td>3 1</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td></td>
<td></td>
<td>3 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>month_sum</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
void output_report(const int id[], const int asum[],
const int msum[], const int sales[][MONTHS])
{
    cout << " RE Sales Report\n\n" // Output headers
    " Agent Sales by Month: Agent\n"
    " Id AUG SEP OCT NOV DEC Totals\n\n";

    int r, c;
    for (r = 0; r < AGENTS; r++) // Output details
    {
        cout << setw(8) << id[r];

        for (c = 0; c < MONTHS; c++)
            cout << setw(6) << sales[r][c];

        cout << setw(8) << asum[r] << endl;
    }

    cout << "\nTotals "; // Output footers
    for (c = 0; c < MONTHS; c++)
        cout << setw(6) << msum[c];

    cout << endl;
}

Output:

RE Sales Report

<table>
<thead>
<tr>
<th>Agent</th>
<th>Sales by Month:</th>
<th>Agent Totals</th>
</tr>
</thead>
<tbody>
<tr>
<td>Id</td>
<td>AUG</td>
<td>SEP</td>
</tr>
<tr>
<td>-------</td>
<td>------</td>
<td>------</td>
</tr>
<tr>
<td>1234</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>2531</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>8730</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>6021</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>

Totals 11 13 12 8 14

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 1 2 2 0 14</td>
<td></td>
</tr>
<tr>
<td>1 2531</td>
<td>1 6 3 1 2 7 1 19</td>
<td></td>
</tr>
<tr>
<td>2 8730</td>
<td>2 0 5 4 3 1 2 13</td>
<td></td>
</tr>
<tr>
<td>3 6021</td>
<td>3 3 1 2 2 4 3 12</td>
<td></td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>month_sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 11 13 12 8 14</td>
</tr>
</tbody>
</table>
```
1. \[ \text{int } a[3][2] = \{1, 2, 3, 4, 5, 6\}; \]

**Array of arrays:** Group of arrays of the same type.

- \( a \) is an array of arrays.
- \( a \) is an array of 3 elements: \( a[0], \ a[1], \) and \( a[2] \) each of which is an array of 2 elements

\[
\begin{array}{c|c|c}
0 & 1 & 2 \\
\hline
a[0] & 1 & 2 \\
\hline
a[1] & 3 & 4 \\
\hline
a[2] & 5 & 6 \\
\end{array}
\]

**Master array:** array of arrays

Example: \( a \)

**Subarray:** element of a master array

Examples: \( a[0], \ a[1], \) and \( a[2] \)

4 arrays have been declared: \( a, \ a[0], \ a[1], \) and \( a[2] \)

2. Can also think of \( a \) as a 2-d array having 3 rows and 2 columns.

\[
\begin{array}{c|c|c}
0 & 1 & 2 \\
\hline
1 & 3 & 4 \\
\hline
2 & 5 & 6 \\
\end{array}
\]
// Initial2dArr: Initialize 2-d arrays

#include <iostream> // cout
#include <iomanip> // setw()

using namespace std;

int main()
{
    cout << "a array:\n";
    int a[3][2] = {{1, 2}, {3, 4}, {5, 6}}; // a array consists of 3 elements, each of which is an array of 2 elements
    int i, j;
    for (i = 0; i < 3; i++) // for each row or subarray
    {
        for (j = 0; j < 2; j++) // for each element of subarray
        {
            cout << setw(4) << a[i][j]; // Uninitialized elements contain 0
            cout << "\n";
        }
    }

    cout << "\nb array:\n";
    int b[2][3] = {{1, 2}, {3, 4}}; // b array consists of 2 elements, each of which is an array of 3 elements
    for (i = 0; i < 2; i++) // for each row or subarray
    {
        for (j = 0; j < 3; j++) // for each element of subarray
        {
            cout << setw(4) << b[i][j]; // Uninitialized elements contain 0
            cout << "\n";
        }
    }

    cout << "\nc array:\n";
    int c[2][3] = {1, 2, 3, 4};
    for (i = 0; i < 2; i++) // for each row or subarray
    {
        for (j = 0; j < 3; j++) // for each element of subarray
        {
            cout << setw(4) << c[i][j]; // Uninitialized elements contain 0
            cout << "\n";
        }
    }

    cout << "\nd array:\n";
    int d[][2] = {1, 2, 3, 4, 5, 6};
    for (i = 0; i < sizeof d / sizeof d[0]; i++) // for each row or subarray
    {
        for (j = 0; j < 2; j++) // for each element of subarray
        {
            cout << setw(4) << d[i][j]; // Uninitialized elements contain 0
            cout << "\n";
        }
    }
    return 0;
}
Output:

a array:
1 2
3 4
5 6

b array:
1 2 0
3 4 0

c array:
1 2 3
4 0 0

d array:
1 2
3 4
5 6

Array Images:

<table>
<thead>
<tr>
<th>a:</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>b:</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>c:</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>d:</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>

Note:

for (int i = 0; i < 3; i++)
{
    for (int j = 0; j < 2; j++)
        ...
}

// i is still in scope with MS, but not ANSI. j is out of scope.