Observe that list[10] does not refer to a memory location in array list.

Warning: it is possible however to read or write past the end of an array by referencing say list[10], list[11], etc. This is not a syntax error in C/C++ but can cause run time errors which are very difficult to track down.

An array of chars (like word above) is called a string in C/C++. There are certain special operations for processing strings in the library string.h, which we will not cover.

C/C++ allows multidimensional arrays

ex. double table[5][6]

declares an array of 30 doubles which we may think of as being arranged in 5 rows and 6 columns:

\[
\begin{array}{cccccc}
\text{table[0][0]}, & \ldots & , & \text{table[0][5]} \\
\vdots & & & \vdots \\
\text{table[4][0]}, & \ldots & , & \text{table[4][5]} \\
\end{array}
\]
The elements of an array can be manipulated like ordinary variables.

Ex:
```cpp
int list[10];
list[0] = 5;
cin >> list[1] >> list[2];
cout << list[0];

list[7] = 2 * list[1] - list[9];
```

An array index can be any expression which evaluates as an int.

Ex:
```cpp
int i, j, k; int list[10];

list[i + i + k] = -4;
```

Array elements can be initialized when declared.

Ex. double weights[3] = {2.7, 3.4, 5.1};

Data Area

```
weights
[0] [1] [2]
2.7 3.4 5.1
```

When initializing an array in this way, the dimension is optional.
Ex. double weights[] = {2.7, 3.4, 5.1};

Does the same thing.

You can also initialize part of an array this way.

Ex. double weights[5] = {2.7, 3.4, 5.1};

**DATA AREA:**

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.7</td>
<td>3.4</td>
<td>5.0</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>

**WEIGHTS**

Character arrays can also be initialized in this way.

Ex. char word[] = {'i', 'o', 'y'};

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>o</td>
<td>y</td>
</tr>
</tbody>
</table>

OR EVEN SHORTER

Ex. char word[] = "joy";

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>o</td>
<td>y</td>
<td>\0</td>
</tr>
</tbody>
</table>

\0 is a special character called the **null character**, which is used as
A sentinel by the string handling functions in string.h.

Control statements
Recall that there are three types of control structures:

1.) **Sequential** (default)
2.) **Conditional**: Decide which instructions to execute based on the value of some boolean expression.
3.) **Looping**: Execute a group of instructions many times.

Recall a boolean expression (or logical expression) is one which can be assigned one of the boolean values true or false.

Ex: \[ a == 0 \]
    \[ b > (c+d) \]

Boolean expressions generally contain one of C/C++ comparison operators.
<table>
<thead>
<tr>
<th><strong>Symbol</strong></th>
<th><strong>Example</strong></th>
<th><strong>Value</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>==</code></td>
<td><code>1 == 2</code></td>
<td><code>false</code></td>
</tr>
<tr>
<td><code>&lt;</code></td>
<td><code>1 &lt; 2</code></td>
<td><code>true</code></td>
</tr>
<tr>
<td><code>&lt;=</code></td>
<td><code>1 &lt;= 2</code></td>
<td><code>true</code></td>
</tr>
<tr>
<td><code>&gt;</code></td>
<td><code>1 &gt; 2</code></td>
<td><code>false</code></td>
</tr>
<tr>
<td><code>&gt;=</code></td>
<td><code>1 &gt;= 2</code></td>
<td><code>false</code></td>
</tr>
<tr>
<td><code>!=</code></td>
<td><code>1 != 2</code></td>
<td><code>true</code></td>
</tr>
</tbody>
</table>

Example: Int `a=5`, `b=6`, `c=7`, `d=8`;

The expression

\[(a+b) \neq (c+d)\]

Evaluates to `true`.

We can build up logical expressions by using the C/C++ logical operators.

<table>
<thead>
<tr>
<th><strong>Operator</strong></th>
<th><strong>Symbol</strong></th>
<th><strong>Example</strong></th>
<th><strong>Value</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>AND</td>
<td><code>&amp;</code></td>
<td><code>(1&lt;2) &amp; (3&gt;4)</code></td>
<td><code>false</code></td>
</tr>
<tr>
<td>OR</td>
<td>`</td>
<td></td>
<td>`</td>
</tr>
<tr>
<td>NOT</td>
<td><code>!</code></td>
<td><code>!(1==2)</code></td>
<td><code>true</code></td>
</tr>
</tbody>
</table>

Note that `!(1==2)` is equivalent to `1 != 2`.
**Conditional Execution**: if \( \frac{1}{2} \) it-else

\[
\begin{align*}
\text{if (condition)} & \quad \text{stmt;}
\end{align*}
\]

or

\[
\begin{align*}
\text{if (condition)} & \quad \text{if-else statemnent}
\end{align*}
\]

\[
\begin{align*}
\text{if (condition)} & \quad \text{stmt1;}
\end{align*}
\]

\[
\begin{align*}
\text{stmt2;}
\end{align*}
\]

\[
\begin{align*}
\text{stmt k;}
\end{align*}
\]

**Example**: if \( a = b \)

\[
\begin{align*}
c = d;
\end{align*}
\]

**Example**: if \( n != 0 \)

\[
\begin{align*}
\text{if (n != 0) }
\end{align*}
\]

\[
\begin{align*}
\text{cout << "n is not zero
"} & \quad c = d;
\end{align*}
\]

An if statement can also have an else clause.
if (condition)
    stmt1;
else
    stmt2;

or

if (condition)
    stmt1;
    stmt2;
    stmt3;
...
    stmtK;
}
else
    stmt(K+1);
    ...
    stmtn;
}

A Sequence of Statements enclosed in braces \{ ... \} is called a Compound Statement or just a Block.

Ex. if (a == b)
    c = a+1;
else
    c = b+1;
A classic error is to use `a = b` instead of `a == b` as the condition in an if or if-else statement. What happens in this case?

The assignment statement

```
Variable = expression;
```

is itself an expression which can be evaluated. Its value is the value assigned to variable.

**Ex**
```
a = (b = c);
```

Is equivalent to
```
b = c;
a = b;
```

Furthermore, if a numeric value is supplied where a bool is expected (such as the condition in an if statement), then non-zero values are converted to true, and zero is converted to false.

Consider the following examples and assume the declarations

```
int a = 1, b = 2, c;
```