Arrays: Homogeneous Aggregate Data Types

Savitch, 2007. Chapter 7
Arrays are used to store and process a collection of data of the same type:

- They are data structures implementing the list data type.
  - All elements are of the same type.
  - Duplicates are allowed.
  - Order is relevant.

They are useful for:
- naming large numbers of variables, and
- processing them.
An array, named score, of type `int` can be declared as

```java
int score[5];
```

This is like defining five variables of type `int`:

```java
int score[0], score[1], score[2], score[3], score[4];
```

The value in brackets is called

- A subscript, or
- An index
The Array Variables

- The variables making up the array are referred to as
  - Indexed variables
  - Subscripted variables
  - Elements of the array

- The number of indexed variables in an array is the declared size, or size of the array
  - The largest index is one less than the size
  - The first index is always zero
Array Variable Types

- An array can have indexed variables of any type.
- All indexed variables in an array are of the same type.
  - This is the base type of the array.
- An indexed variable can be used anywhere an ordinary variable of the base type can be used.
Using \( [\ ] \) With arrays

In an array declaration, \( [\ ] \)s enclose the size of the array such as this array of five \texttt{ints}:

\[
\text{int score}[5];
\]

When referring to one of the indexed variables, the \( [\ ] \)s enclose a number identifying one of the indexed variables

\[
\text{score[3]} \text{ is one of the indexed variables}
\]

The value in the \( [\ ] \)s can be any expression that evaluates to one of the integers from 0 to size - 1.
To assign a value to an indexed variable, use the assignment operator:

```java
int n = 2;
score[n + 1] = 99;
```

In the example, the indexed variable `score[3]` is assigned the value 99.
Loops are commonly used to process arrays:

```cpp
for (int i = 0; i < 5; i++)
    cout << score[i] << " off by"
    << (max - score[i]) << endl;
```

This would display the difference between each score and the largest score in the array.
Use constants to declare the size of an array.
This allows your code to be easily altered for use on a smaller or larger set of data:

```cpp
const int NUMBER_OF_STUDENTS = 50;
int score[NUMBER_OF_STUDENTS];

//...

for (i = 0; i < NUMBER_OF_STUDENTS; i++)
    cout << score[i] << " off by " << (max - score[i]) << endl;
```
C/C++ do not allow the use of a variable to declare the size of an array:

```c
int number;
//...
cout << "Enter number of students: ";
cin >> number;
int score[number];
```
Memory

- Memory consists of byte-addressable locations:
  - An eight-bit value stored
  - A 32 bit (usually) address pointing to the value
- A simple variable is stored in consecutive bytes, the number of which depends on the variable's type
- A variable's address is the address of the first byte
Declaring the array

```c
int a[6];
```

- Reserves memory for six variables of type `int`
- The variables are stored contiguously
- The address of `a[0]` is associated with `a`
- The address of `a[3]` is determined by:
  - starting at `a[0]`
  - counting past enough memory for three `ints`
  - to find `a[3]`
An Array in Memory

int a[6];

address of a[0]

On this computer each indexed variable uses 2 bytes, so a[3] begins 2 \times 3 = 6 bytes after the start of a[0].

There is no indexed variable a[6], but if there were one, it would be here.

There is no indexed variable a[7], but if there were one, it would be here.

a[0]
a[1]
a[2]
a[3]
a[4]
a[5]
some variable named stuff
some variable named more_stuff
A common error is using a nonexistent index

- Index values for
  ```
  int a[6];
  ```
  are the values 0 through 5

- An index value not allowed by the array declaration is out of range

- Using an out of range index does not produce an error message!
Out of Range Problems

If an array is declared as `int a[6];`
and an integer is declared as `int i = 7;`

Executing the statement `a[i] = 238;` causes:

- The calculation of the address of the nonexistent `a[7]`
- This address may hold other valid data
- The value 238 is stored at the calculated address, perhaps overwriting valid data
- No warning is issued!
Initializing arrays

To initialize an array when it's declared, the values for the indexed variables are enclosed in braces and separated by commas:

```c
int children[3] = {2, 12, 1};
```

// Is equivalent to:

```c
int children[3];
children[0] = 2;
children[1] = 12;
children[2] = 1;
```
Default values

If too few values are listed in an initialization:

- The listed values are used to initialize the first of the indexed variables
- The remaining variables are initialized to a zero of the base type

For example,

```c
int a[10] = {5, 5};
```

initializes `a[0]` and `a[1]` to 5 and `a[2]` through `a[9]` to 0

Uninitialized arrays have undefined values.
#include <iostream>

const int NUMBER_OF_EMPLOYEES = 3;

int adjust_days(int old_days);
//Returns old_days plus 5.

int main() {
    using namespace std;
    int vacation[NUMBER_OF_EMPLOYEES], number;

    cout << "Enter allowed vacation days for employees 1"
    << " through " << NUMBER_OF_EMPLOYEES << ":\n"
    for (number = 1; number <= NUMBER_OF_EMPLOYEES; number++)
        cin >> vacation[number - 1];
    for (number = 0; number < NUMBER_OF_EMPLOYEES; number++)
        vacation[number] = adjust_days(vacation[number]);
    cout << "The revised number of vacation days are:\n"
    for (number = 1; number <= NUMBER_OF_EMPLOYEES; number++)
        cout << "Employee number " << number
        << " vacation days = " << vacation[number - 1] << endl;
    return 0;
}

int adjust_days(int old_days) {
    return (old_days + 5);
}
Arrays as Function Arguments

- A parameter can be an entire array
- Arrays are passed to the receiving function by name only
- Although they are value parameters, they behave like reference parameters
- This is because the array name is actually the address of the first element
- Therefore, the values of the original indexed variables can actually be changed by the function
Array Parameter Considerations

- A function does not know the size of an array argument.
- The programmer should include a parameter that specifies the size of the array.
- The function will, therefore, be able to process arrays of various sizes.
```cpp
#include <iostream>

void fill_up(int a[], int size);
// Precondition: size is the declared size of
// the array a. The user will type in size integers.
// Postcondition: The array a is filled with
// size integers from the keyboard.
// Uses iostream:

void fill_up(int a[], int size) {
    using namespace std;
    cout << "Enter " << size << " numbers:\n";
    for (int i = 0; i < size; i++)
        cin >> a[i];
    size--;
    cout << "The last array index used is "
         << size << endl;
}
```
Case Study: Production Graph
The **const** Modifier

- Array parameters allow a function to change the values stored in the array argument.

- If a function should not change the values of the array argument, use the **const** modifier.

- If **const** is used to modify an array parameter:
  - **const** is used in both the prototype and the function definition.
  - The compiler will issue an error if code is written that changes the values in the array.
Function Calls and \textit{const}

If a function with a constant array parameter calls another function using the \textit{const} array as an argument...

- The called function must use a constant array parameter.
- The compiler will issue an error if a function is called that does not have a \textit{const} array parameter to accept the array argument.
Programming With Arrays

The size needed for an array changes.

- Runtime conditions may dictate requirements.
- The conditions may not be known completely at compile time.

Common solutions:

- Declare the array size to be the largest that could be needed.
- Decide how to deal with partially-filled arrays.
Partially-Filled Arrays

When using arrays that are partially filled:

- Functions dealing with the array may not need to know the declared size of the array, only how many elements are currently stored.

- A parameter, `number_used`, may be sufficient to ensure that referenced index values are legal.

- A function such as `fill_array()` in the next example needs to know the declared size of the array.
#include <iostream>

const int MAX_NUMBER_SCORES = 10;

void fill_array(int a[], int size, int& number_used);
double compute_average(const int a[], int number_used);
void show_difference(const int a[], int number_used);

int main() {
    using namespace std;
    int score[MAX_NUMBER_SCORES], number_used;

    cout << "This program reads golf scores and shows\n" << "how much each differs from the average.\n";
    cout << "Enter golf scores: \n";
    fill_array(score, MAX_NUMBER_SCORES, number_used);
    show_difference(score, number_used);
    return 0;
}
```cpp
void fill_array(int a[], int size, int& number_used) {
    using namespace std;
    cout << "Enter up to " << size << " nonnegative whole numbers.\n" << "Mark the end of the list with a negative number.\n";
    int next, index = 0;
    cin >> next;
    while ((next >= 0) && (index < size)) {
        a[index] = next;
        index++;
        cin >> next;
    }
    number_used = index;
}
```


```cpp
double compute_average(const int a[], int number_used) {
    double total = 0;

    for (int index = 0; index < number_used; index++)
        total = total + a[index];
    if (number_used > 0) {
        return (total / number_used);
    } else {
        using namespace std;
        cout << "ERROR: number of elements is 0 in compute_average.\n" << "compute_average returns 0.\n";
        return 0;
    }
}
```
void show_difference(const int a[], int number_used) {
    using namespace std;
    double average = compute_average(a, number_used);
    cout << "Average of the " << number_used << " scores = " << average << endl
         << "The scores are:\n";
    for (int index = 0; index < number_used; index++)
        cout << a[index] << " differs from average by "
             << (a[index] - average) << endl;
}
When function fill_array is called, `MAX_NUMBER_SCORES` is used as an argument. Can't `MAX_NUMBER_SCORES` be used directly without making it an argument?

Using `MAX_NUMBER_SCORES` as an argument makes it clear that `fill_array()` requires the array's declared size.

This makes `fill_array()` easier to be used in other programs.
A sequential search is one way to search an array for a given value. Look at each element from first to last to see if the target value is equal to any of the array elements. The index of the target value can be returned to indicate where the value was found in the array. A value of -1 can be returned if the value was not found.
The `search()` Function

- Uses a while loop to compare array elements to the target value.
- Sets a variable of type bool to true if the target value is found, ending the loop.
- Checks the boolean variable when the loop ends to see if the target value was found.
- Returns the index of the target value if found, otherwise returns -1.
const int DECLARED_SIZE = 20;

void fill_array(int a[], int size, int& number_used);
int search(const int a[], int number_used, int target);

int main( ) {
    using namespace std;
    int arr[DECLARED_SIZE], list_size, target;

    fill_array(arr, DECLARED_SIZE, list_size);
    char ans;
    int result;
    do {
        cout << "Enter a number to search for: ";
        cin >> target;
        result = search(arr, list_size, target);
        if (result == -1)
            cout << target << " is not on the list.\n";
        else
            cout << target << " is stored in array position "
                 << result << endl
                 << "(Remember: The first position is 0.)\n";
            cout << "Search again? (y/n followed by return): ";
            cin >> ans; }
        while ((ans != 'n') && (ans != 'N'));
    cout << "End of program.\n";
    return 0;
}
void fill_array(int a[], int size, int& number_used) {
    using namespace std;
    cout << "Enter up to " << size << " nonnegative whole numbers.\n" << "Mark the end of the list with a negative number.\n";
    int next, index = 0;
    cin >> next;
    while ((next >= 0) && (index < size)) {
        a[index] = next;
        index++;
        cin >> next; }
    number_used = index;
}
```cpp
int search(const int a[], int number_used, int target) {
    int index = 0;
    bool found = false;

    while (!found && (index < number_used)) {
        if (target == a[index])
            found = true;
        else
            index++;
    }

    if (found)
        return index;
    else
        return -1;
}
```
Sorting a list of values is very common task.

- Create an alphabetical listing.
- Create a list of values in ascending order.
- Create a list of values in descending order.

Many sorting algorithms exist.

- Some are very efficient.
- Some are easier to understand.
while (the array is not empty)

find the largest (or smallest) element

swap it with the last (or first) element

make the array one element smaller
The Selection Sort
```cpp
void fill_array(int a[], int size, int& number_used);
void sort(int a[], int number_used);
void swap_values(int& v1, int& v2);
int index_of_smallest(const int a[],
                      int start_index,
                      int number_used);

int main( ) {
    using namespace std;
    cout << "This program sorts numbers from lowest to highest.\n";
    int sample_array[10], number_used;
    fill_array(sample_array, 10, number_used);
    sort(sample_array, number_used);
    cout << "In sorted order the numbers are:\n";
    for (int index = 0; index < number_used; index++)
        cout << sample_array[index] << " ";
    cout << endl;
    return 0;
}
```
```cpp
void fill_array(int a[], int size, int& number_used) {
    using namespace std;
    cout << "Enter up to " << size << " nonnegative whole numbers.\n" << "Mark the end of the list with a negative number.\n";
    int next, index = 0;
    cin >> next;
    while ((next >= 0) && (index < size)) {
        a[index] = next;
        index++;
        cin >> next; }
    number_used = index;
}
```
void sort(int a[], int number_used) {
    int index_of_next_smallest;

    for (int index = 0; index < number_used - 1; index++) {
        index_of_next_smallest =
            index_of_smallest(a, index, number_used);
        swap_values(a[index], a[index_of_next_smallest]);
    }
}
void swap_values(int& v1, int& v2) {
    int temp = v1;

    v1 = v2;
    v2 = temp;
}

int index_of_smallest(const int a[], int start_index, int number_used) {
    int min = a[start_index],
       index_of_min = start_index;

    for (int index = start_index + 1; index < number_used; index++)
        if (a[index] < min) {
            min = a[index];
            index_of_min = index; }
    return index_of_min;
}
C++ allows arrays with multiple index values

```cpp
char page [30] [100];
```
declares an array of characters named `page`

`page` has two index values:
The first ranges from 0 to 29
The second ranges from 0 to 99

Each index is enclosed in its own brackets

`page` can be visualized as an array of 30 rows and 100 columns
Index Values of page

The indexed variables for array `page` are

\[
\begin{align*}
\text{page}[0][0], & \quad \text{page}[0][1], \quad \ldots, \quad \text{page}[0][99] \\
\text{page}[1][0], & \quad \text{page}[1][1], \quad \ldots, \quad \text{page}[1][99] \\
\ldots \\
\text{page}[29][0], & \quad \text{page}[29][1], \quad \ldots, \quad \text{page}[29][99]
\end{align*}
\]

`page` is actually an array of size 30

`page`'s base type is an array of 100 characters
Multidimensional Array Parameters

Recall that the size of an array is not needed when declaring a parameter:

```c
void display_line
  (const char a[ ], int size);
```

The base type of a multi-dimensional array, however, must be completely specified in the parameter declaration:

```c
void display_page
  (const char page[ ] [100],
   int size_dimension_1);
```
Grade records for a class can be stored in a two-dimensional array.

For a class with 4 students and 3 quizzes the array could be declared as `int grade[4][3];`

The first array index refers to the number of a student.

The second array index refers to a quiz number.

Since student and quiz numbers start with one, we subtract one to obtain the correct index.
The grading program uses one-dimensional arrays to store...
- Each student's average score
- Each quiz's average score

The functions that calculate these averages use global constants for the size of the arrays.

This was done because the functions seem to be particular to this program.
### Two-Dimensional Array, Grade

<table>
<thead>
<tr>
<th>student 1</th>
<th>quiz 1</th>
<th>grade[0][0]</th>
<th>grade[0][1]</th>
<th>grade[0][2]</th>
</tr>
</thead>
<tbody>
<tr>
<td>student 2</td>
<td>quiz 2</td>
<td>grade[1][0]</td>
<td>grade[1][1]</td>
<td>grade[1][2]</td>
</tr>
<tr>
<td>student 3</td>
<td>quiz 3</td>
<td>grade[2][0]</td>
<td>grade[2][1]</td>
<td>grade[2][2]</td>
</tr>
<tr>
<td>student 4</td>
<td></td>
<td>grade[3][0]</td>
<td>grade[3][1]</td>
<td>grade[3][2]</td>
</tr>
</tbody>
</table>

- `grade[3][0]` is the grade that student 4 received on quiz 1.
- `grade[3][1]` is the grade that student 4 received on quiz 2.
- `grade[3][2]` is the grade that student 4 received on quiz 3.
### Another View

#### Data Table

<table>
<thead>
<tr>
<th>Student</th>
<th>Quiz 1</th>
<th>Quiz 2</th>
<th>Quiz 3</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Student 1</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10.0</td>
</tr>
<tr>
<td>Student 2</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1.0</td>
</tr>
<tr>
<td>Student 3</td>
<td>8</td>
<td>6</td>
<td>9</td>
<td>7.7</td>
</tr>
<tr>
<td>Student 4</td>
<td>8</td>
<td>4</td>
<td>10</td>
<td>7.3</td>
</tr>
</tbody>
</table>

**Quiz Averages:**

- quiz_ave[0] = 7.0
- quiz_ave[1] = 5.0
- quiz_ave[2] = 7.5

**Student Averages:**

- st_ave[0] = 10.0
- st_ave[1] = 1.0
- st_ave[3] = 7.3
const int NUMBER_STUDENTS = 4, NUMBER_QUIZZES = 3;

void compute_st_ave(const int grade[][NUMBER_QUIZZES],
                    double st_ave[]);
void compute_quiz_ave(const int grade[][NUMBER_QUIZZES],
                      double quiz_ave[]);
void display(const int grade[][NUMBER_QUIZZES],
             const double st_ave[],
             const double quiz_ave[]);

int main() {
    using namespace std;
    int grade[NUMBER_STUDENTS][NUMBER_QUIZZES];
    double st_ave[NUMBER_STUDENTS];
    double quiz_ave[NUMBER_QUIZZES];

    grade[0][0] = 10;
    grade[0][1] = 10;
    grade[0][2] = 10;
    grade[1][0] = 2;
    grade[1][1] = 0;
    grade[1][2] = 1;
    grade[2][0] = 8;
    grade[2][1] = 6;
    grade[2][2] = 9;
    grade[3][0] = 8;
    grade[3][1] = 4;
    grade[3][2] = 10;
    compute_st_ave(grade, st_ave);
    compute_quiz_ave(grade, quiz_ave);
    display(grade, st_ave, quiz_ave);
    return 0;
}
void compute_st_ave(const int grade[][NUMBER_QUIZZES],
                     double st_ave[])
{
    for (int st_num = 1; st_num <= NUMBER_STUDENTS; st_num++) {
        double sum = 0;
        for (int quiz_num = 1; quiz_num <= NUMBER_QUIZZES; quiz_num++)
            sum = sum + grade[st_num - 1][quiz_num - 1];
        st_ave[st_num - 1] = sum / NUMBER_QUIZZES;
    }
}

void compute_quiz_ave(const int grade[][NUMBER_QUIZZES],
                       double quiz_ave[])
{
    for (int quiz_num = 1; quiz_num <= NUMBER_QUIZZES; quiz_num++) {
        double sum = 0;
        for (int st_num = 1; st_num <= NUMBER_STUDENTS; st_num++)
            sum = sum + grade[st_num - 1][quiz_num - 1];
        quiz_ave[quiz_num - 1] = sum / NUMBER_STUDENTS;
    }
}
```cpp
void display(const int grade[][NUMBER_QUIZZES],
             const double st_ave[],
             const double quiz_ave[]) {

    using namespace std;
    cout.setf(ios::fixed);
    cout.setf(ios::showpoint);
    cout.precision(1);
    cout << setw(10) << "Student"
        << setw(5) << "Ave"
        << setw(15) << "Quizzes\n";
    for (int st_num = 1; st_num <= NUMBER_STUDENTS; st_num++) {
        cout << setw(10) << st_num
            << setw(5) << st_ave[st_num - 1] << " ";
        for (int quiz_num = 1; quiz_num <= NUMBER_QUIZZES; quiz_num++)
            cout << setw(5) << grade[st_num - 1][quiz_num - 1];
        cout << endl; }
    cout << "Quiz averages = ";
    for (int quiz_num = 1; quiz_num <= NUMBER_QUIZZES; quiz_num++)
        cout << setw(5) << quiz_ave[quiz_num - 1];
    cout << endl;
}
Case Study: Production Graph
Problem definition:

- The program will display a bar graph showing the production of each of four plants for a week.
- Each plant has separate records for each department.
- Input is entered plant by plant.
- Output shows one asterisk for each 1,000 units, and production is rounded to the nearest 1,000 units.
Use an array named production to hold production of each plant.

The production for plant \( n \) is stored in \texttt{production[n-1]}.

The program must scale production to the nearest 1,000 units to display asterisks in the bar.
**Production Graph Sub-Tasks**

- **input_data**: Read input for each plant and set `production[plant_number - 1]` to the total production for plant number `n`.

- **scale**: For each plant, change `production[plant_number]` to the correct number of asterisks.

- **graph**: Output the bar graph.
The entire array will be an argument for the functions written to perform the subtasks.

- A parameter for the size of the array will be included.
- The size of the array is equal to the number of plants.
- A constant will be used for the number of plants.
```cpp
#include <iostream>

const int NUMBER_OF_PLANTS = 4;

void input_data(int a[], int last_plant_number);
void scale(int a[], int size);
void graph(const int asterisk_count[],
           int last_plant_number);

int main( ) {
    using namespace std;
    int production[NUMBER_OF_PLANTS];

    cout << "This program displays a graph showing\n";
    cout << "production for each plant in the company.\n";
    input_data(production, NUMBER_OF_PLANTS);
    scale(production, NUMBER_OF_PLANTS);
    graph(production, NUMBER_OF_PLANTS);
    return 0;
}
```
Design: input_data

All departments' data for each plant are read and added to produce a plant's total.

for each plant number

read all the data for the current plant

sum the numbers

set production[plant_number -1] to the total
Testing `input_data`

Each function should be tested in a program in which it is the only untested function.

Because `input_data` calls `get_total`, `get_total` is tested first.

Once tested, `get_total` can be used to test `input_data`. 
#include <iostream>

const int NUMBER_OF_PLANTS = 4;

void input_data(int a[], int last_plant_number);
void get_total(int& sum);

int main() {
    using namespace std;
    int production[NUMBER_OF_PLANTS];
    char ans;

    do {
        input_data(production, NUMBER_OF_PLANTS);
        cout << endl
          << "Total production for each"
          << " of plants 1 through 4:
          " << endl;
        for (int number = 1; number <= NUMBER_OF_PLANTS; number++)
            cout << production[number - 1] << " ";
        cout << endl
          << "Test Again?(Type y or n and return): ";
        cin >> ans; } 
while ( (ans != 'N') && (ans != 'n') );
    cout << endl;
    return 0;
}
void input_data(int a[], int last_plant_number) {
    using namespace std;
    for (int plant_number = 1;
         plant_number <= last_plant_number; plant_number++) {
        cout << endl
             << "Enter production data for plant number "
             << plant_number << endl;
        get_total(a[plant_number - 1]);
    }
}
void get_total(int& sum) {
    using namespace std;
    cout << "Enter number of units produced by each department.\n"
         << "Append a negative number to the end of the list.\n";
    sum = 0;
    int next;
    cin >> next;
    while (next >= 0) {
        sum = sum + next;
        cin >> next; }
    cout << "Total = " << sum << endl;
}
Test Data for `input_data`

`input_data` should be tested with:

- a plant that contains no production figures,
- a plant having only one production figure, and
- a plant having more than one figure.

In general, plants with zero and non-zero production figures.
scale changes the value of the indexed variable to show the whole number of asterisks to print.

scale is called using
scale(production, NUMBER_OF_PLANTS);

Its algorithm divides the value of each production number by 1,000 and rounds the result to the nearest integer.
The floor function

- Function \texttt{round}, called by \texttt{scale} uses the \texttt{floor} function from the \texttt{cmath} library.

- The \texttt{floor} function returns the nearest integer to the left on the number line.

- Adding 0.5 to the argument for floor accomplishes the rounding.
```cpp
#include <iostream>
#include <cmath>

void scale(int a[], int size);
int round(double number);

int main() {
    using namespace std;
    int some_array[4], index;

    cout << "Enter 4 numbers to scale: ";
    for (index = 0; index < 4; index++)
        cin >> some_array[index];
    scale(some_array, 4);
    cout << "Values scaled to the number of 1000s are: ";
    for (index = 0; index < 4; index++)
        cout << some_array[index] << " " ;
    cout << endl;
    return 0;
}
```
void scale(int a[], int size) {
    for (int index = 0; index < size; index++)
        a[index] = round(a[index] / 1000.0);
}

int round(double number) {
    using namespace std;
    return static_cast<int>(floor(number + 0.5));
}
```cpp
#include <iostream>
#include <cmath>

const int NUMBER_OF_PLANTS = 4;

void input_data(int a[], int last_plant_number);
void scale(int a[], int size);
void graph(const int asterisk_count[], int last_plant_number);
void get_total(int& sum);
int round(double number);
void print_asterisks(int n);

int main() {
    using namespace std;
    int production[NUMBER_OF_PLANTS];

    cout << "This program displays a graph showing\n" << "production for each plant in the company.\n";
    input_data(production, NUMBER_OF_PLANTS);
    scale(production, NUMBER_OF_PLANTS);
    graph(production, NUMBER_OF_PLANTS);
    return 0;
}
```
```cpp
void input_data(int a[], int last_plant_number) {
    using namespace std;
    for (int plant_number = 1; plant_number <= last_plant_number; plant_number++) {
        cout << endl
             << "Enter production data for plant number "
             << plant_number << endl;
        get_total(a[plant_number - 1]);
    }
}
```
```cpp
void get_total(int& sum) {
    using namespace std;
    cout << "Enter number of units produced by each department.\n" << "Append a negative number to the end of the list.\n";
    sum = 0;
    int next;
    cin >> next;
    while (next >= 0) {
        sum = sum + next;
        cin >> next;
    }
    cout << "Total = " << sum << endl;
}
```
```cpp
void scale(int a[], int size) {
    for (int index = 0; index < size; index++)
        a[index] = round(a[index] / 1000.0);
}

int round(double number) {
    using namespace std;
    return static_cast<int>(floor(number + 0.5));
}
```
```cpp
void graph(const int asterisk_count[], int last_plant_number) {
    using namespace std;
    cout << "Units produced in thousands of units:\n";
    for (int plant_number = 1;
        plant_number <= last_plant_number; plant_number++) {
        cout << "Plant #" << plant_number << " ";
        print_asterisks(asterisk_count[plant_number - 1]);
        cout << endl; }
}

void print_asterisks(int n) {
    using namespace std;
    for (int count = 1; count <= n; count++)
        cout << "*";
}
```
Arrays can use structures or classes as their base types.

```c
struct WindInfo {
    double velocity;
    char direction;
};

WindInfo data_point[10];
```
Accessing Members

Use the dot operator to access the members of an indexed variable.

```cpp
for (i = 0; i < 10; i++) {
    cout << "Enter velocity: ";
    cin >> data_point[i].velocity;
    //...
}
```
The **Money** class of Chapter 8 can be the base type for an array.

When an array of classes is declared the default constructor is called to initialize the indexed variables.

An array of class **Money** is demonstrated in the following...
#ifndef MONEY_H
#define MONEY_H

#include <iostream>
using namespace std;

namespace moneysavitch {

    class Money {
        public:
            friend Money operator+(const Money& amount1, const Money& amount2);
            friend Money operator-(const Money& amount1, const Money& amount2);
            friend Money operator-(const Money& amount);
            friend bool operator==(const Money& amount1, const Money& amount2);
            friend bool operator<(const Money& amount1, const Money& amount2);
            Money(long dollars, int cents);
            Money(long dollars);
            Money();
            double get_value() const;
            friend istream& operator>>(istream& ins, Money& amount);
            friend ostream& operator<<(ostream& outs, const Money& amount);
        private:
            long all_cents; }
    }
}
#endif
```cpp
#include <iostream>
#include "money.h"

int main( ) {
    using namespace std;
    using namespace moneysavitch;
    Money amount[5], max;
    int i;

    cout << "Enter 5 amounts of money:\n";
    cin >> amount[0];
    max = amount[0];
    for (i = 1; i < 5; i++) {
        cin >> amount[i];
        if (max < amount[i])
            max = amount[i];
    }
    Money difference[5];
    for (i = 0; i < 5; i++)
        difference[i] = max - amount[i];
    cout << "The highest amount is " << max << endl;
    cout << "The amounts and their\n" << "differences from the largest are:\n";
    for (i = 0; i < 5; i++) {
        cout << amount[i] << " off by "
             << difference[i] << endl;
    }
    return 0;
}
```
#include <iostream>
#include <cctype>
#include <cstdlib>
#include "money.h"

namespace moneysavitch {

bool operator < (const Money& amount1,
                 const Money& amount2) {
    return (amount1.all_cents < amount2.all_cents);
}

int digit_to_int(char c) {
    return (int(c) - int('0') );
}
}
# Money.cpp (b)

```cpp
istream& operator>>(istream& ins, Money& amount) {
    char one_char, decimal_point,
    digit1, digit2;
    long dollars;
    int cents;
    bool negative;

    ins >> one_char;
    if (one_char == '-') {
        negative = true;
        ins >> one_char; } else
        negative = false;
    ins >> dollars >> decimal_point >> digit1 >> digit2;
    if ( one_char != '$' || decimal_point != '.'
         || !isdigit(digit1) || !isdigit(digit2) ) {
        cout << "Error illegal form for money input\n";
        exit(1); } 
    cents = digit_to_int(digit1) * 10 + digit_to_int(digit2);
    amount.all_cents = dollars * 100 + cents;
    if (negative)
        amount.all_cents = -amount.all_cents;
    return ins;
}
```
`operator <<`(ostream& outs, const Money& amount) {
  long positive_cents, dollars, cents;
  positive_cents = labs(amount.all_cents);
  dollars = positive_cents / 100;
  cents = positive_cents % 100;
  if (amount.all_cents < 0)
    outs << "-$" << dollars << ".";
  else
    outs << "$" << dollars << ".";
  if (cents < 10)
    outs << '0';
  outs << cents;
  return outs;
}
Money operator -(const Money& amount1, const Money& amount2) {
    Money temp;
    temp.all_cents = amount1.all_cents - amount2.all_cents;
    return temp;
}

Money operator -(const Money& amount) {
    Money temp;
    temp.all_cents = -amount.all_cents;
    return temp;
}

Money operator +(const Money& amount1, const Money& amount2) {
    Money temp;
    temp.all_cents = amount1.all_cents + amount2.all_cents;
    return temp;
}

bool operator ===(const Money& amount1, const Money& amount2) {
    return (amount1.all_cents == amount2.all_cents);
}
Money::Money(long dollars, int cents) {
  if (dollars*cents < 0) {
    cout << "Illegal values for dollars and cents.\n";
    exit(1); }
  all_cents = dollars * 100 + cents;
}

Money::Money(long dollars) : all_cents(dollars*100) { }

Money::Money() : all_cents(0) { }

double Money::get_value( ) const {
  return (all_cents * 0.01);
}
A structure or class can contain an array as a member.

```c
struct Data {
    double time[10];
    int distance;
};

Data my_best;
```
Use the dot operator to identify the array within the structure.

Use the `[ ]`'s to identify the indexed variable desired.

Example, `my_best.time[i]` references the ith indexed variable of the variable `time` in the structure `my_best`
Arrays as Class Members

- Class **TemperatureList** includes an array.
- The array, named **list**, contains temperatures.
- Member variable **size** is the number of items stored.

```cpp
class TemperatureList {
    public:
        TemperatureList( );
    private:
        double list [MAX_LIST_SIZE];
        int size;
}
```
Overview of TemperatureList

To create an object of type TemperatureList:

```cpp
TemperatureList  my_data;
```

To add a temperature to the list:

```cpp
My_data.add_temperature(77);
```

A check is made to see if the array is full.

```cpp
<< is overloaded so output of the list is
cout << my_data;
```
#ifndef TEMPLIST_H
#define TEMPLIST_H

#include <iostream>
using namespace std;

namespace tlistsavitch {
    const int MAX_LIST_SIZE = 50;

class TemperatureList { 
    public:
        TemperatureList( );
        void add_temperature(double temperature);
        bool full( ) const;
    friend ostream& operator <<(ostream& outs, const TemperatureList& the_object);

    private:
        double list[MAX_LIST_SIZE];
        int size;
    }
};

} #endif
#include <iostream>
#include <cstdlib>
#include "templist.h"

using namespace std;

namespace tlistsavitch {

TemperatureList::TemperatureList( ) : size(0) { }

void TemperatureList::add_temperature(double temperature) {
    if ( full( ) ) {
        cout << "Error: adding to a full list.\n";
        exit(1);
    } else {
        list[size] = temperature;
        size = size + 1;
    }
}

bool TemperatureList::full( ) const {
    return (size == MAX_LIST_SIZE);
}

ostream& operator <<(ostream& outs,
    const TemperatureList& the_object) {
    for (int i = 0; i < the_object.size; i++)
        outs << the_object.list[i] << " F\n";
    return outs;
}
}

}