Outline: Strings and Arrays

In this chapter you will:

- arrays
- how to declare and manipulate data into arrays
- the meaning of “array index out of bounds”
- the restrictions on array processing
- to pass an array as a parameter to a function

Outline: Strings and Arrays

- C-strings
- the use of string functions to process C-strings
- how to input data in—and output data from—a C-string
- parallel arrays
- how to manipulate data in a two-dimensional array
- multidimensional arrays

Data Types

- A data type is called simple if variables of that type can store only one value at a time
- A structured data type is one in which each data item is a collection of other data items
Arrays

- **Array** - a collection of a fixed number of components wherein all of the components are of the same data type
- **One-dimensional array** - an array in which the components are arranged in a list form
- The general form of declaring a one-dimensional array is:
  
  ```
  dataType arrayName[intExp];
  ```
- where intExp is any expression that evaluates to a positive integer

Declaring an array

- The statement `int num[5];` declares an array `num` of 5 components of the type `int`

![Figure 9-1 Array num](image)

Accessing Array Components

- The general form (syntax) of accessing an array component is:
  
  ```
  arrayName[indexExp]
  ```
- where indexExp, called index, is any expression whose value is a nonnegative integer
- Index value specifies the position of the component in the array
- The `[]` operator is called the array subscripting operator
- The array index starts at 0 !!!

Processing One-Dimensional Arrays

- Some of the basic operations performed on a one-dimensional array are:
  - initialize
  - input data
  - output data stored in an array
  - find the largest and/or smallest element
- Each of these operations requires the ability to step through the elements of the array
- Stepping-through the elements of an array is easily accomplished by a loop
Accessing Array Components

Consider the declaration

```cpp
int list[100];  // list is an array of the size 100
int i;
```

This for loop steps-through each element of the array list starting at the first element

```cpp
for(i = 0; i < 100; i++)  // Line 1
    process list[i]  // Line 2
```

If processing list requires inputting data into list, the statement in Line 2 takes the form of an input statement, such as the cin statement

```cpp
for(i = 0; i < 100; i++)  // Line 1
    cin>>list[i];
```

Array Initialization

- Like any other simple variable, arrays can also be initialized while they are being declared
- When initializing arrays while declaring them, it is not necessary to specify the size of the array
- The size of the array is determined by the number of initial values in the braces
- For example:

```cpp
double sales[] = {12.25, 32.50, 16.90, 23, 45.68};
```

Partial Initialization

- The statement

  ```cpp
  int list1[10] = {0};
  ```

  declares list1 to be an array of 10 components and initializes all components to zero
- The statement

  ```cpp
  int list2[10] = {8, 5, 12};
  ```

  declares list2 to be an array of 10 components, initializes list2[0] to 8, list2[1] to 5, list2[2] to 12 and all other components are initialized to 0
- The statement

  ```cpp
  int list[] = {5, 6, 3};
  ```

  declares list to be an array of 3 components and initializes list[0] to 5, list[1] to 6, and list[2] to 3
- The statement

  ```cpp
  int list[25] = {4, 7};
  ```

  declares list to be an array of 25 components. The first two components are initialized to 4 and 7, respectively and all other components are initialized to zero
Array Index Out of Bounds

- If we have the statements:
  
  ```
  double num[10];
  int i;
  ```

  - The component num[i] is a valid index if i = 0, 1, 2, 3, 4, 5, 6, 7, 8, or 9
  - The index of an array is in bounds if the index >= 0 and the index <= arraySize-1
  - If either the index < 0 or the index > arraySize-1, then we say that the index is out of bounds
  - There is no guard against indices that are out of bounds because C++ does not check if the index value is with in range

Restrictions on Array Processing

- If x and y are two arrays of the same type and size then the following statement is illegal:
  
  ```
  y = x; // illegal
  ```

- In order to copy one array into another array we must copy component-wise
  
  ```
  for(j = 0; j < 25; j++)
  y[j] = x[j];
  ```

Comparison of arrays, reading data into an array and printing the contents of an array must be done component-wise

```
 cin >> x;  // illegal
 cout << y; //illegal
```  

- C++ does not allow aggregate operations on an array
- An aggregate operation on an array is any operation that manipulates the entire array as a single unit

Arrays as Parameters to Functions

- Arrays are passed by reference only
- The symbol & is NOT necessary when declaring an array as a formal parameter
- The size of the array is usually omitted
- If the size of the one-dimensional array is specified when it is declared as a formal parameter, it is ignored by the compiler
- The reserved word const in the declaration of the formal parameter can still prevent the function from changing the actual parameter
**Base Address of an Array**

- The base address of an array is the address, or memory location of the first array component.
- If list is a one-dimensional array, then the base address of list is the address of the component list[0].
- When we pass an array as a parameter, the base address of the actual array is passed to the formal parameter.
- The function can not return a value of the type array.

**Arrays - Class**

- If a class has constructors and you declare an array of class objects, the class must have the default constructor.
- The default constructor is used to initialize each (array) class object.
- For example:
  ```
  clockType clocks[100];
  ```

**Arrays - exercise**

- Write a function that takes an integer array and its size as the parameters, and return the maximum of the array.
- Write a function that takes an integer array and its size as the parameters, and return the minimum of the array.
- Write a program to read values for an integer array and print its maximum and minimum.

**C Strings (Character Arrays)**

- Character array - an array whose components are of the type char is called a character array.
  ```
  ch = '\0';
  ```
- String - a sequence of zero or more characters enclosed in double quote marks.
- C stings are null terminated.
- This means that the last character in a string is the null character.
C Strings (Character Arrays)

- If we have the text:
  
  "John L. Johnson"
  "Hello there."

- There is a difference between 'A' and "A"
- The first one is character A and the second is string A
- Since strings are null terminated, "A" represents two characters, 'A' and \0

"Hello" represents six characters, 'H', 'e', 'l', 'l', 'o', and \0

- To store 'A' we need only one memory cell of the type char, while to store "A", we need two memory cells of the type char, one for 'A' and the other for \0
- To store the string "HELLO" we need six memory cells of the type char

Consider the statement

```c
char name[16];
```

Since C strings are null terminated and name has sixteen components, the largest string that can be stored in name is 15

- If you store a string of length, say 10 in name, the first 11 components of name are used and the last 5 are left unused
- If we have the statement:
  ```c
  char name[16] = {'J', 'o', 'h', 'n', '\0'};
  ```

During char array variable declaration, the string notation to be used is

```c
char name[16] = "John";
```

- The statement
  ```c
  char name[] = "John";
  ```
- declares a string variable name of length large enough, that is, 5 (here) and stores "John" in it
- Both statements stores "John" in name
- The size of name in the statement in Line A is 16 where it is 5 in Line B
String Comparison

- C-strings are compared character by character using the collating sequence of the system.
- If we are using the ASCII character set:
  1. The string "Air" is smaller than the string "Boat".
  2. The string "Air" is smaller than the string "An".
  3. The string "Bill" is smaller than the string "Billy".
  4. The string "Hello" is smaller than "hello".

Reading and Writing Strings

String Input

- Since aggregate operations are allowed for string input, the statement `cin >> name;` will store the next input string into name.
- Strings that contain blanks cannot be read using the extraction operator `>>`.
- To read strings with blanks, the general form (syntax) of the get function together with an input stream variable, such as `cin` is:
  ```
  cin.get(str, m+1);
  ```
- This statement stores the next `m` characters or until the newline character `\n` is found into `str` but the newline character is not stored in `str`.

String Output

- The statement `cout << name;` will output the content of name on the screen.
- The insertion operator `<<` continues to write the contents of name until it finds the null character.
- If name does not contain the null character, then we will see strange output since `<<` continues to output data from memories adjacent to name until `\0` is found.

Input/Output Files

- Values (strings) of the type `string` are not null terminated.
- Variables of type string can be used to read and store the names of input/output files.
- The argument to the function `open` must be a null terminated string, that is a C-string.
- If we use a variable of the type string to read the name of an input/output file and then use this variable to open a file, the value of the variable must first be converted to a C-string (null-terminated string).
Input/Output Files

- The header file string contains the function \texttt{c\_str}, which converts a value of the type \texttt{string} to a null-terminated character array (C-string).
- The syntax to use the function \texttt{c\_str} is:
  \begin{verbatim}
  strVar.c\_str()
  \end{verbatim}
- where \texttt{strVar} is a variable of the type \texttt{string}

Parallel Arrays

- Two (or more) arrays are called parallel if their corresponding components hold related information.
- For example:
  \begin{verbatim}
  int studentId[50];
  char courseGrade[50];
  \end{verbatim}

Two-Dimensional Arrays

- Two-dimensional Array - a two-dimensional array is a collection of a fixed number of components arranged in two dimensions, and all components are of the same type.
- The syntax for declaring a two-dimensional array is:
  \begin{verbatim}
  dataType arrayName[intexp1][intexp2];
  \end{verbatim}
- where \texttt{intexp1} and \texttt{intexp2} are expressions yielding positive integer values.
- The two expressions \texttt{intexp1} and \texttt{intexp2} specify the number of rows and the number of columns, respectively, in the array.

Example

\begin{verbatim}
int table[3][4];
\end{verbatim}

\begin{verbatim}
<table>
<thead>
<tr>
<th></th>
<th>column 0</th>
<th>column 1</th>
<th>column 2</th>
<th>column 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>row 0</td>
<td>table[0][0]</td>
<td>table[0][1]</td>
<td>table[0][2]</td>
<td>table[0][3]</td>
</tr>
<tr>
<td>row 1</td>
<td>table[1][0]</td>
<td>table[1][1]</td>
<td>table[1][2]</td>
<td>table[1][3]</td>
</tr>
</tbody>
</table>
\end{verbatim}
Accessing Array Components

- The syntax to access a component of a two-dimensional array is:
  \[ \text{arrayName}[\text{indexexp1}][\text{indexexp2}] \]
- where indexexp1 and indexexp2 are expressions yielding nonnegative integer values
- indexexp1 specifies the row position and indexexp2 specifies the column position

Initialization During Declaration

- Like one-dimensional arrays, two-dimensional arrays can be initialized when they are declared
- To initialize a two-dimensional array when it is declared
  1. The elements of each row are enclosed within braces and separated by commas.
  2. All rows are enclosed within braces.
  3. For number arrays, if all components of a row are not specified, the unspecified components are initialized to zero.

Initialization During Declaration

Example

```c
int table[3][4] = { {1, 2, 3, 4},
                   {10, 20, 30, 40},
                   {100, 200, 300, 400}};
```

Processing Two-dimensional Arrays

- A two-dimensional array can be processed in three different ways:
  1. Process the entire array
  2. Process a particular row of the array, called row processing
  3. Process a particular column of the array, called column processing
- Each row and each column of a two-dimensional array is a one-dimensional array
- When processing a particular row or column of a two-dimensional array we use algorithms similar to process one-dimensional arrays
Initialization

- This for loop initializes row two to zero:
  ```cpp
  row = 2;
  for (col = 0; col < 4; col++)
      table[row][col] = 0;
  ```
- To initialize the entire table, we can also put the first index (row position) in a loop
- These nested for loops initialize each component of table to 0:
  ```cpp
  for (row = 0; row < 3; row++)
      for (col = 0; col < 4; col++)
          table[row][col] = 0;
  ```

Print

- The following nested for loops print the components of matrix, one row per line:
  ```cpp
  for (row = 0; row < 3; row++)
  {
      for (col = 0; col < 4; col++)
          cout << setw(5) << table[row][col] << " ";
      cout << endl;
  }
  ```

Input

- This for loop inputs data in row number 4 of table:
  ```cpp
  row = 2;
  for (col = 0; col < 4; col++)
      cin >> table[row][col];
  ```
- By putting the row number in a loop we can input data in each component of the table
  ```cpp
  for (row = 0; row < 3; row++)
      for (col = 0; col < 4; col++)
          cin >> table[row][col];
  ```

Two-dimensional Arrays

- Two-dimensional arrays can be passed as parameters to a function and they are passed by reference
- The base address, that is, the address of the first component of the actual parameter is passed to the formal parameter
- If matrix is the name of a two-dimensional array, then matrix[0][0] is the first component of matrix
Two-dimensional Arrays

- When storing a two-dimensional array in the computer's memory, the row order form is used.
- This means that the first row is stored first, followed by the second row, which is followed by the third row and so on.
- When declaring a two-dimensional array as a formal parameter, we can omit the size of the first dimension, but not the second.
- The number of columns must be specified.

Multi-dimensional Arrays

- Array - a collection of a fixed number of elements (called components) arranged in n dimensions \((n \geq 1)\), called an n-dimensional array.
- The general syntax of declaring an n-dimensional array is:
  \[
  \text{dataType arrayName}[\text{intExp1}][\text{intExp2}]...[\text{intExpn}];
  \]
- where \(\text{intExp1}, \text{intExp2}, \ldots, \) and \(\text{intExpn}\) are constant expressions yielding positive integer values.
- The syntax of accessing a component of an n-dimensional array is:
  \[
  \text{arrayName}[\text{indexExp1}][\text{indexExp2}]...[\text{indexExpn}]
  \]
- where \(\text{indexExp1}, \text{indexExp2}, \ldots, \) and \(\text{indexExpn}\) are expressions yielding nonnegative integer values. \(\text{indexExp}\) gives the position of the array component in the \(i\)th dimension.
Multi-dimensional Arrays

- When declaring a multi-dimensional array as a formal parameter in a function, we can omit the size of the first dimension but not the other dimensions.
- As parameters, multi-dimensional arrays are passed by reference only and a function cannot return a value of the type array.
- There is no check if the array indices are within bounds.

Two-dimensional Arrays

- Assume each student studies 4 subjects each semester. Consider a group of 3 students.
- Declare the array for storing the scores of 3 students in a group.
- Calculate the average of each student.
- Print the best average score.

```
studentScoreArray.cpp
```

```
80.5  90  77.5  88
60   75  76.5  77.5
78.5 89.5 60.5  84
```