

Lecture 19: Introduction to Arrays

PIC 10A
Todd Wittman

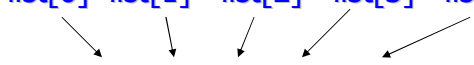


Sec 6.5: Declaring an Array

- An array is a list of multiple values of the same type.
- The general form of an array declaration is:
type array_name [size];
- For example, a list of 5 integers is declared:
`int list[5];`
- Note that without [5] it would be just a single integer.
- We can declare any data type, even classes.
`double double_list[20];`
`string words[100];` *//List of 100 words.*
`Card deck[52];` *//A deck of 52 cards.*
- Just like before, when we declare the array it has no useful values in it.
- We can initialize an array with: `type name[size] = {val1, val2,...valN};`
`int list[5] = {11, -12, 42, 38, -105};` *//Also could omit the [5].*
- This often is not practical, since we usually don't know the values right away and/or the array size is very large.

Using an Array

- An individual element of an array can be accessed by its index: `array_name[index]`
- Array indices always start at 0.
- So the last index is always `size-1`.
- What does the code below print?

`list[0] list[1] list[2] list[3] list[4]`

`int list[5] = {11, -12, 42, 38, -105};`
`cout << list[3];`

- What is the index of the last element?

3 Simple Examples

- We often use for loops to assign array values.
- Ex Store the first 100 even integers.

```
int evens[100];
for (int i = 0; i < 100; i++)
    evens[i] = 2*i+2;
```

- Ex Read in 10 doubles and compute the sum.

```
double list[10];
double sum = 0;
for (int i = 0; i < 10; i++) {
    cin >> list[i];
    sum += list[i];
}
cout << "\nThe sum is: " << sum;
```

- Ex Read in 4 hobbit names.

```
string hobbits[4];
for (int i = 0; i < 4; i++) {
    cout << "Enter the name of hobbit #" << i+1 << ": ";
    cin >> hobbits[i];
}
```

Thing1 to Watch Out For



- **DON'T FORGET**: For an array of size N, the array indices run from 0 to N-1.
- Visual Studio will not give you an error if you run past the length of the array. But you'll get very strange results.

```
int values[3] = {10, 20, 30};  
for (int i=0; i<=3; i++)  
    cout << values[i] << " ";
```

OUTPUT

10 20 30 -858993460

Thing2 to Watch Out For



- The code below will give you an error.

```
int my_size = 3;           //BAD!!  
int values[my_size];      //ERROR!!
```
- Gives the error: "expected constant size"
- The size of the array must be a constant. For example:

```
const int my_size = 3;    //GOOD!!  
int values[my_size];
```

- Once you've set up an array, its size cannot be changed.
- Why does the code below not work?

```
int size;  
cin >> size;  
int my_array[size];
```

Example: Find Max & Min

- Ex Find the maximum and minimum of an a given integer array **list** of length **N**.

```
int min = list[0];
int max = list[0];
for (int i = 1; i < N; i++) {
    if ( list[i] < min )
        min = list[i];
    if ( list[i] > max )
        max = list[i];
}
cout << "The maximum is " << max << ".\n";
cout << "The minimum is " << min << ".\n";
```

Arrays in Functions

- In a function declaration, we denote that a parameter is an array with blank brackets [].
- We usually pass the size of the array as well, unless the size is available as a global constant.

```
void my_fun (double my_array[], int size) {
```

- When we call the function, we just give the variable name.

```
double list[100];
my_fun (list, 100);
```

- Unlike vectors, arrays are always passed by reference even without the &. This is to save memory. So we don't include the & because changes are always recorded.
- Also unlike vectors, arrays cannot be a return type.

Arrays in Functions

- Ex Write the max/min routine as a function.

```
void maxmin (int list[], int N, int& max, int& min)  {  
    min = list[0];  
    max = list[0];  
    for (int i = 1; i < N; i++)  {  
        if ( list[i] < min )  
            min = list[i];  
        if ( list[i] > max )  
            max = list[i];  
    }  
    return;  
}
```

Class Arrays

- We can even create arrays with a user-defined class.

`Triangle my_triangles[100];`

`Card deck[52];`

- Each element is an object. So we can call a member function for each element.

`my_triangles[5].draw();`

`string suitCard25 = deck[25].get_suit();`

`if (deck[11] < deck[12])`

Debug This!

- Ex Write a function which rewrites the elements of a double array in reverse order.

list =

1	2	3	4	5.1
---	---	---	---	-----

 list =

5.1	4	3	2	1
-----	---	---	---	---

- The code below does not work properly. Why?

```
void reverseArray (double list[ ], int size) {  
    for (int i = 0; i < size ; i++) {  
  
        list[i] = list[size-i];  
        list[size-i] = list[i];  
    }  
    return;  
}
```



Debugged!

- The last example had several problems.
 1. Index size-i runs one past the array size.
 2. The element swapping does not work.
 3. We should only proceed through the first half of the array.

```
void reverseArray (double list[ ], int size) {  
    double temp;  
    for (int i = 0; i < size/2; i++) {  
        temp = list[i];  
        list[i] = list[size-i-1];  
        list[size-i-1] = temp;  
    }  
    return;  
}
```



Sec 9.5.4: 2D Arrays

- Often data appears in two dimensions, like in a spreadsheet, data table, or an image.

12.2	3.1	4.27	2.1	8.8
6.9	3.2	-12.8	6.1	2.6
1.0	14.42	42.14	12.12	92.87

- To declare a 2D array, use
type variable_name [num_rows] [num_columns];
- For example, to create the data table above use:
double table [3] [5];
- To assign individual elements, give it a row # and column #.
table [2] [4] = 92.87;

2D Arrays

- When you pass a 2D array to a function, tell the function the # rows and columns.
void my_function (double table [rows] [cols]) { ...
- *(Actually, just the # columns is required. But specifying the # rows too doesn't hurt.)*
- So to pass these numbers, probably best to declare the array size as global constants.
- At the top of your program, right under the #includes state the size of your array.
- Good style to attach the variable name to these values.
const int table_rows = 10;
const in table_cols = 20;

Printing A Table.

- Ex Write a function which prints the values of a 3x5 table.

```
#include <iomanip>
```

```
const int table_rows = 3;
```

```
const int table_cols = 5;
```

```
void printTable (double table [table_rows] [table_cols]) {  
    for (int i = 0; i < table_rows; i++) {  
        for (int j=0; j < table_cols; j++) {  
            cout << setw(10) << table [i] [j];  
        }  
        cout << "\n";  
    }  
    return;  
}
```

Matrix Transpose

- Here's a little something for students who are sweating through Linear Algebra this semester.
- The transpose of a matrix changes every row into a column.
- An MxN matrix becomes a NxM matrix.

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \quad A^T = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}$$

- Write a function which computes the matrix transpose of A.
- Key observation: $A[i][j] = A^T[j][i]$

Matrix Transpose

```
const int A_rows = 2;    //Or whatever sizes you want.
const int A_cols = 3;

void transpose (int A[A_rows][A_cols],
               int Atrans [A_cols][A_rows]) {
    for (int i = 0; i < A_rows; i++)
        for (int j=0; j < A_cols; j++)
            Atrans [j][i] = A[i][j];
    return;
}
```

Vectors vs. Arrays

- We recommend using vectors, but there are situations in which you will use arrays.

	Arrays	Vectors
Dynamic?	Arrays are static, they cannot change size.	Vectors are dynamic, they can change size using <code>resize</code> , <code>push_back</code> , and <code>pop_back</code> .
Type	Arrays are a primitive (built-in) data type.	Vectors are a class defined in the <code><vector></code> library.
Size	The array size is generally kept as a global constant.	Vectors can look up their own size with the <code>size()</code> member function.
2D	Arrays extend naturally to 2D.	Vectors are a little awkward at 2D tables.
Speed	Arrays are much faster on older compilers.	Vectors are almost as fast as arrays on newer compilers.
Usage	Arrays are more common in business.	Vectors are gaining popularity.