Programming with Arrays

Larry Caretto
Computer Science 106

Computing in Engineering and Science

Spring 2005

California State University Northridge

Outline

- Why do we need arrays
- Declaring and using arrays
- Writing code with arrays and for loops
- Data processing with arrays
- Passing arrays to functions
- Writing functions with arrays
- Two-dimensional arrays

California State University Northridge 2

Representing Data

Run	Data
1	12.3
2	14.4
3	11.8
4	12.5
5	13.2
6	14.1

- Consider a set of experimental data with several runs
- How do we represent the data in such a way that we can process these data and similar data with more values?

California State University Northridge 3

Representing Data II

Run, i	x data
1	x_1
2	x_2
3	x_3
4	x_4
5	x_5
6	x_6

- x_i is mathematical notation for several cases of similar data
- Use this formula to find the mean of N data items

$$\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i$$

California State University Northridge 4

Representing Data III

- When we have a set of N data items like x_i it will occupy N memory locations
- A variable like x, declared as double x, occupies only one memory location
- For our data on x_i
  - We want to call is by its name, x
  - We want to have N memory locations
  - We want to compute formulas like average x
  - We want to refer to a specific x, say x_3

California State University Northridge 5

Arrays Represent Data

- An array is a way that we can represent the mathematical notation for x_i
- We use the programming notation x[i] to represent the general data element x_i
- When we declare a variable as an array, we reserve the memory locations that we will need for the data
  - Regular variable: double x;
  - Array variable: double x[200];

California State University Northridge 6

How to represent x_i

- An array has a single variable name, like x, augmented by a subscript to identify the particular data item
- Example x[3] or x[k]
- Power of array structure is use of variable subscript as loop index to refer to different elements
- Arrays must be declared with maximum size

One-dimensional C++ Array

Math	C++
x_0	x[0]
x_1	x[1]
x_2	x[2]
x_3	x[3]
x_4	x[4]
x_5	x[5]

- View one-dimensional arrays as a column (or row) of cells
- Start with zero subscript
- Array shown here has 6 elements with subscripts from 0 to 5

Maximum Array Subscript

```
double w[4];  // 4 elements
const int MAX_SIZE = 10;
double x[MAX_SIZE];  // 10 elements
```

- Minimum subscript is zero
- Maximum subscript is one less than the number of elements
- w[0], w[1], w[2], and w[3] are the four elements of the w array
- Note different meanings of w[N]

Maximum Array Subscript

- Array elements are stored in contiguous memory locations
- Program computes memory location from subscript
- C++ does not check to see if an array subscript is in bounds
- An incorrect subscript could affect some other memory location

Subscript out of Range

y
x[0]
x[1]
x[2]
x[3]
x[4]
z

- Cells show memory locations for y, x[] array, and z
- The x array has five elements stored in the locations shown
- x[-1] would give the same location as the variable y
- x[5] would give the same location as the variable z

Using Arrays

- Individual components of arrays, such as x[3] or y[k], are used in the same way as ordinary variables
- Variable subscripts must be assigned a value before use as in examples below

  int k = 3, m = 5;
  double x[5] = { 1, 3, 5, 18, 143 }, z[50], r = 1;
  x[k] = 4; x[3] = 4
  z[2*k+3] = x[k-2] - 5 * r * x[3]; // = ???

z[2*3+3] = x[3-2] – 5 * r * x[3]; or z[9] = x[1] – 5 * r * x[3]
= 3 – 5 * 1 * 4 = -17

Examples of Use

- cin >> x[k];
- cout << “y[“ << k << “] = “ << y[k];
- data[3] = <*expression*>
- result = 3 + voltage * current[m]
- position[m] = position[m+1]
- for (int k = 0; k < N; k++) x[k] = 0;
- r = pow(y[3], 2);
- power[i] = current[i] * voltage[i];

California State University Northridge 13

Array Questions

- Write statements to do the following
  - to declare a double array, x, that can have 20 elements double x[20]
  - to set element 3 of the slide array equal to the value of element 2 slide[3] = slide[2]
  - Assign element k of the power array a value of the product of element k of the current array times element k of the voltage array

    power[k] = current[k] * voltage[k]

California State University Northridge 14

Arrays and for Loops

- Perhaps the most important array code uses a for loop where the loop index becomes the array subscript

```
const int MAX = 10;
double x[MAX], sum = 0;
// code to input x array goes here
for ( int k = 0; k < MAX; k++ )
     sum += x[k];
```

California State University Northridge 15

General Array Processing

- To process each element in an array with N elements, starting with the initial element, use a for loop with index k
  - k starts at zero
  - The continuation condition, k < N, will process elements 0, 1, 2, … , N-1
  - Increment k by 1
- `for ( int k = 0; k < N; k++)`

California State University Northridge 16

General Array Processing II

- On the previous chart N means the number of elements defined, not the total number of elements that can be stored in the array
- Sometimes it is more convenient to refer to the subscripts than to the number of elements
- E. g., array whose first and last defined elements have subscripts F and L
- `for ( k = F, k <= L; k++ )`

California State University Northridge 17

General Array Processing III

- In the examples that follow, we will generally assume that an array has N elements, whose first subscript is zero
- The for loop command to process each element in such an array is
  `for ( k = 0; k < N; k++)`
- We can use different increments (e.g. k += 3) to skip elements

California State University Northridge 18

Array Input and Output

```
const int MAX_SIZE = 100;
double z[MAX_SIZE];
ifstream infile( "array.dat" );
for (int i = 0; i < MAX_SIZE; i++)
{
    infile >> z[i];
    cout << "z[" << i << "] = "
         << z[i];
}
```


Defined Elements

- The number of elements defined may be less then the array size
- You may declare an array to be the maximum size expected but actually specify a value for fewer elements

```
double x[10];
for ( int j = 1; j < 5; j++ )
    x[j] = 1 / double( j );
```


Computing the Mean

$$\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i = \frac{1}{N}\sum_{i=0}^{N-1} x_i$$

```
sum = 0;
for( i = 0; i < N; i++ )
{    sum += x[i];        }
average = sum / N;
```

- N data items to average
- Subscripts starts at 0
- Last data item is element N-1 in array
- {} not needed

Finding the Maximum

- How do you find the maximum or minimum in a set of numbers?
- *E. g.*: 13 74 -3 12 91 0 -17 88 -4
- Now that you found the maximum and minimum, how would you explain what you did so a computer can understand?
- Scan the list and remember the largest (smallest) number you have seen and replace if you find one larger (smaller)

Finding the Maximum

- Store the current maximum in a variable (e.g., `max`) and compare `max` to new array values
- If a new value is greater than `max` replace `max` by that value

```
double max = x[0];// initialize max
for ( int i = 1; i < N; i++ )
{    if ( x[i] > max )
        { max = x[i]; }
}
```

- Can omit both sets of braces

Initializing Arrays

- We can initialize an array by placing all the data values in braces following the array declaration

```
int x[5] = { 12, 17, -22, 4, 12 };
int x[] = { 12, 17, -22, 4, 12 };
```

- Note that the maximum size is not required when we initialize an array

Data Processing with Arrays

- You have taken data from a circuit that gives the current and voltage
- There are N pairs of data
- Current is stored as the amps[k] array and voltage as the volts[k] array
- Write the code to compute the average power if N, volts[] and amps[] are defined

Average Power One

```
double sum = 0
for ( int k = 0; k < N; k++ )
{
     power[k] = amps[k] * volts[k];
     sum += power[k];
}
double averagePower = sum / N;
cout << "Power = " << averagePower
  << " watts";
```


Average Power Two

```
double sum = 0
for ( int k = 0; k < N; k++ )
{
     power = amps[k] * volts[k];
     sum += power;
}
double averagePower = sum / N;
cout << "Power = " << averagePower
  << " watts";
```


Average Power Three

```
double sum = 0
for ( int k = 0; k < N; k++ )
{
     sum += amps[k] * volts[k];
}
double averagePower = sum / N;
cout << "Power = " << averagePower
  << " watts";
```


Differences in Power Codes

- Used three ways to compute power
- Only one used a power[k] array
- Code works with power not an array or not even a variable
- Usually define arrays when we want to save results of a computation for use in subsequent computations

Passing Arrays to Functions

- We can pass an array element to a function as we pass any variable
- y = pow(x[k], 3);
- Here the pow function returns the cube of element k of the x array
- This is no different from passing a single variable to a function
- We can also pass whole arrays, like x, to functions: getAverage(x, first, last)

getAverage

- Computes the average of elements of the x array from x[first] to x[last] (inclusive)
- Header: double getAverage (double x[], int first, int last)
- Prototypes:
  - double getAverage (double x[], int first, int last);
  - double getAverage (double [], int, int);
- Note use of [] to specify an array as a function argument

getAverage

```
double getAverage ( double x[],
                    int first, int last )
{
    double sum = 0;
    for ( int i = first; i <= last; i++ )
         sum += x[i];

    return sum / ( last - first + 1 );
}
```


Use of getAverage

- double x[22], power[50], density[30];
- // code to get input data on x and power
- double mean = getAverage(x, 0, 10)
- double average = getAverage(power, 12, 24)
- How would you compute the average of all elements of the density array?

```
getAverage( density, 0, 29)
```


Standard Deviation

- Measure of spread around mean

$$s = \sqrt{\frac{\sum_{i=0}^{N-1}(x_i - \bar{x})^2}{N-1}} = \sqrt{\frac{\left(\sum_{i=0}^{N-1} x_i^2\right) - N(\bar{x})^2}{N-1}} = \sqrt{\frac{\left(\sum_{i=0}^{N-1} x_i^2\right) - \frac{1}{N}\left(\sum_{i=0}^{N-1} x_i\right)^2}{N-1}}$$

- First term is definition; others are computational forms
- How would we write a function to com-pute s for all the elements in an N-element array?

getStdDev

```
double getStdDev(double x[],int N)
{
    double sum = 0, sum2 = 0, sumxy =
  0;
    for ( int k = 0; k < N; k++ )
    {
        sum += x[k];
        sum2 += x[k] * x[k];
        sumxy += x[k] * y[k];
    }
    return sqrt( ( sum2 - sum *
       sum / N ) / ( N - 1 ) );
}
```


Arrays Passed by Reference

```
double mystery( double x[], int N )
{
    for ( int k = 0, k < N; k++ )
    {    x[k] = 0;    }
    return 0;
}
```

- A call, double y = mystery(c, M) would zero the first M elements of the c array
- Pass by reference occurs by default without the need for an &

Two-dimensional Arrays

- One-dimensional arrays refer to a variable that has multiple entries with a single classification
- Two-dimensional arrays are used to represent data with two classifications
  – Example: an experiment on manufacturing productivity measures daily output of four machines with six operators

Two-dimensional Arrays

- One-dimensional variable
  – mathematical notation x_i
  – C++ array notation x[i]
- Two-dimensional
  – mathematical notation x_{ik}
  – C++ array notation x[i][k]
- One-way versus two-way classification

One-dimensional Array

[0]
[1]
[2]
[3]
[4]
[etc.]

- View one-dimensional arrays as a column (or row) of cells
- Start will subscript [0] and increase by 1 for each new cell

Two-Dimensional Array

[0][0]	[0][1]	[0][2]	[0][3]	[0][4]
[1][0]	[1][1]	[1][2]	[1][3]	[1][4]
[2][0]	[2][1]	[2][2]	[2][3]	[2][4]
[3][0]	[3][1]	[3][2]	[3][3]	[3][4]
[4][0]	[4][1]	[4][2]	[4][3]	[4][4]
[5][0]	[5][1]	[5][2]	[5][3]	[5][4]
[6][0]	[6][1]	[6][2]	[6][3]	[6][4]

- View two-dimensional arrays as a table with rows and columns of cells

Two-dimensional Example

- In the example of a manufacturing process measuring the output of four machines with six operators
  – Array named output depending on integer subscripts machine and operator
  – First subscript is for operator and second is for machine

```
const int maxOp = 6, maxMach = 4;
int output[maxOp][maxMach];
cout << output[3][2];
```


Two-Dimensional Array Data

	M 0	M 1	M 2	M 3	Op tot
Op 0	34	53	43	31	161
Op 1	39	55	42	36	172
Op 2	33	52	45	40	170
Op 3	31	48	39	25	143
Op 4	38	59	48	42	187
Op 5	33	49	48	28	158
M tot	208	316	265	202	991

output[3][2]

Individual data plus totals for operators and machines

Array data

Two-dimensional array Code

```
const int maxOp = 6, maxMach = 4
int output[maxOp][maxMach];
for (int op = 0; op < maxOp; op++)
{
    for ( int mach = 0; mach <
                maxMach; mach++ )
       cout << output[op] [mach] <<
      " units produced at machine "
      << mach << " with operator "
      << op;
}
```

California State University Northridge 43

Other Code

- How would you compute the total units produced by each machine?
- How would you compute the total units produced by each operator?
- How would you compute the average and standard deviation for all the units produced by the operators?

California State University Northridge 44

Units for Each Machine

- This sum is the total output of each machine from all operators (column sum)

```
int outMach[maxMach];
for (int mac = 0; mac < maxMach; mac++)
{
    outMach[mac] = 0;
    for (int op = 0; op < maxOp; op++)
    {outMach[mac] += output[op][mac];}
    cout << "Total machine " << mac <<
        << " output is "<<outMach[mac];
}
```

California State University Northridge 45

Units for Each Operator

- This sum is the total output of each operator from all machines (row sum)

```
int outOp[maxOp];
for ( int op = 0; op < maxOp; op++ )
{
   outOp[op] = 0;
   for ( int m = 0; m < maxMach; m++ )
    { outOp[op] += output[op][m]; }
   cout << "Total operator " << op
        << " output is " << outOp[op];
}
```

California State University Northridge 46

Comments on this Code

- Note that we use one-dimensional arrays to store row (operator) and column (machine) sums
- Note that order of subscripts is always [operator][machine]
- Conventional, but not required, to write tables as arrays with subscript ordered as [row][column]

California State University Northridge 47

Simultaneous Linear Equations

- Example of 3 equations (3 unknowns)
  - $3x + 7y – 3z = 8$
  - $2x – 4y + z = -3$
  - $8x + 6y – 2z = 14$
- How can we develop a general notation for N equations in N unknowns?
  - Call variables x_0, x_1, x_2, etc.
  - Call right hand side b_0, b_1, b_2, etc.
  - Call top row coefficients a_{00}, a_{01}, a_{02}, etc.

California State University Northridge 48

Standard Form

$a_{00}x_0 + a_{01}x_1 + a_{02}x_2 + ... + a_{0N-1}x_{N-1} + a_{0N}x_N = b_0$
$a_{10}x_0 + a_{11}x_1 + a_{12}x_2 + ... + a_{1N-1}x_{N-1} + a_{1N}x_N = b_1$
$a_{20}x_0 + a_{21}x_1 + a_{22}x_2 + ... + a_{2N-1}x_{N-1} + a_{2N}x_N = b_2$
… … … … … … … … … … … … … … … … … … …
$a_{N-1,0}x_0 + a_{N-1,1}x_1 + .. … … … ... + a_{N-1,N}x_N = b_{N-1}$
$a_{N0}x_0 + a_{N1}x_1 + a_{N2}x_2 + ... … … … .. + a_{NN}x_N = b_N$

- Note that subscripts on a are $a_{row,column}$ where row is equation and column is unknown

California State University Northridge 49

Compact Standard Form

$$\sum_{j=1}^{N} a_{ij} x_j = b_i \qquad i = 1, \ldots, N$$
$$\sum_{j=0}^{N-1} a_{ij} x_j = b_i \qquad i = 0, \ldots, N-1$$

- Set of equations defined by N and data on a_{ij} and b_i
- Functions to solve this problem take 2D a array and 1D b array to find array x

California State University Northridge 50

Example in Standard Form

- Previous example 3 equations (N = 3)
  - 3x + 7y – 3z = 8
  - 2x – 4y + z = -3
  - 8x + 6y – 2z = 14
- In standard form:
  - x is x_0, y is x_1, and z is x_2
  - $a_{00} = 3$, $a_{01} = 7$, $a_{02} = -3$, b_0, = 8
  - $a_{10} = 2$, $a_{11} = -4$, $a_{12} = 1$, b_1, = -3
  - $a_{20} = 8$, $a_{21} = 6$, $a_{22} = -2$, b_2, = 14

California State University Northridge 51

Standard Form in C++

- Equations represent unknowns as x_i, the right hand sides as b_i, and the left hand side coefficients as a_{ij}
- In C++ we use arrays x[col] for the unknowns, b[row] for the right hand sides, and a[row][col] for the coefficients on the left hand side
- Project three will use library program to solve this system of equations

California State University Northridge 52

Passing 2D Arrays to Functions

- Execution of array code based on computing memory location from address of first array member plus subscript for particular element
- For one-dimensional array we only need the address of the first element to find the location of x[i]
- What about two-dimensional arrays?

California State University Northridge 53

Passing 2D Arrays to Functions II

- Consider an array x with declared as x[maxFirst][maxSecond]
- The location of x[i][j] is computed as i + j*maxSecond locations from the start of the array
- We must know the second dimension to compute the location
- We must pass this to the function that has a two-dimensional array as a parameter

California State University Northridge 54

Passing 2D Arrays to Functions III

- Global constant: `const int maxSecond = 20`
- Function header

```
double getSum ( double x[][maxSecond],…
```

- Function prototype (semicolon at end)

```
double getSum ( double x[][maxSecond],…
double getSum ( double [][maxSecond],…
```

- Calling program

```
const int maxFirst = 20;
double x[20][maxSecond];
// other code assigns values to x array
double result = getSum( x, ….
```

California State University Northridge 55

Passing 2D Arrays to Functions IV

- Global constant not required, but helpful to accommodate changes to size of second dimension
- The second dimension must be the same in the following three statements:
  - The function prototype
  - The function header
  - The declaration of the array passed to the function
- Final project uses two-dimensional arrays

California State University Northridge 56

Passing 2D Arrays to Functions V

- Example: write a function that accepts a two-dimensional array, output, used in the previous example and computes and returns the row sums and columns sums as well as the total
- How to pass information?
  - Pass 2D output array into function
  - Return 1D arrays with row and column sums
  - Return total in function name
  - Pass number of machines and operators, which can be less than the maximum array sizes, into function

California State University Northridge 57

Example of 2D Array Function

```
int getSums( int output[][maxMach],
             int opSum[], int machSum[],
             int Nop, int Nmach)
{
   int total = 0;
   for ( int op = 0; op < Nop; op++ )
   {
      opSum[op] = 0;
      for ( int m = 0; m < Nmach; m++ )
         opSum[op] += output[op][m];

      total += opSum[op];
   }
            // continues on next chart
```

California State University Northridge 58

2D Array Function Concluded

```
   for ( int m = 0; m < Nmach; m++ )
   {
      machSum[m] = 0;
      for ( int op = 0; op < Nop; op++ )
         machSum[op] += output[op][m];
   }
   return total;
}  // closes function opening brace
```

- How do we use this function?
- What is its prototype?

California State University Northridge 59

Using the 2D Array Function

- Start with global constants for common array dimensions in various locations

```
const int maxMach = 10, maxOp = 10;
```

- Prototype is just header with a semicolon

```
int getSums( int output[][maxMach],
             int opSum[], int machSum[],
             int Nop, int Nmach);
```

- Use global constants as array dimensions in calling program

```
int output[maxOp][maxMach],
  opSum[maxOp], machSum[maxMach];
```

California State University Northridge 60

Using the 2D Array Function

- Get data in calling program (usually from file)

```
ifstream inFile( "production.dat" );
inFile >> Nop >> Nmach;
for (op = 0; op < Nop; op++ )
{   for ( m = 0; m < Nmach; m++ )
        infile >> output[op][m]; }
```

- Call function

```
int total = getSums( output, opSum,
machSum, Nop, Nmach);
```

- Output results

Array call has only array names

Northridge 61

Input Data Files for Arrays

- Must match input statements in code

```
for (i = 0; i < N; i++) cin >> x[i];
for (i = 0; i < N; i++) cin >> y[i];
```

- Compare above statements with code below

```
for (i = 0; i < N; i++)
{  cin >> x[i] >> y[i]; }
```

- First example read all x data then all y data. Second reads x and y data in pairs
- Usually write code to determine number of array elements by testing for end of file

Northridge 62

Input Data File for 1D Arrays

12
20
32
55
43
19
27
88

12 20 32 55 43 19 27 88

12	20
32	55
43	19
27	88

- How the code below read x and y from each file on this page?

```
for (i = 0; i < 3; i++)
  cin >> x[i] >> y[i];
```

- What about this code?

```
for (i = 0; i < 3; i++)
  cin >> x[i];
for (i = 0; i < 3; i++)
  cin >> y[i];
```

Northridge 63

Input Data Files for 2D Arrays

- Recall input code from example of passing 2D arrays to functions

```
ifstream inFile("production.dat");
inFile >> Nop >> Nmach;
for (op = 0; op < Nop; op++ )
{   for ( m = 0; m < Nmach; m++ )
    {   infile >> output[op][m]; }
}
```

- How would you prepare the data file?

Braces not needed

Northridge 64

Input Data File for 2D Arrays

- Usually prepare data file for 2D arrays to look like row and column data

```
6    4
34   53   43   31
39   55   42   36
33   52   45   40
31   48   39   25
38   59   48   42
33   49   48   28
```

Northridge 65

Is There Life After 2D Arrays

- Yes, we can have arrays with three or more dimensions
- A program to compute emissions of different species, different vehicle types, different model years could use

emissions[species][vehType][modelYear]

- Code structures are similar with use of nested for loops on array subscripts
- Will not cover in this course

Northridge 66

Summary of Arrays

- Used to represent data of one kind with multiple occurrences
- Can have one-way, two-way, etc., classifications of the data
- Math symbols a_{ij} and x_j become C++ arrays a[i][j] and x[i]
- Declaring array size; maximum subscript; no subscript checking

Array Summary Continued

- Use for loops where loop index is array subscript to access array elements
- Array elements like ordinary variables
- Passing whole arrays to functions (header, prototype, call, 1D vs. 2D)
- Nested loops for 2D array code
- Input files for arrays must match input statements

Representing Data

Run	Data	Math	C++
0	12.3	x_0	x[0]
1	14.4	x_1	x[1]
2	11.8	x_2	x[2]
3	12.5	x_3	x[3]
4	13.2	x_4	x[4]
5	14.1	x_5	x[5]

- C++ array, x[i] used to represent data for which x_i is used in mathe-matical notation

Using Arrays

- Declare arrays in typical way, but add maximum elements, e.g. int v[100];
- Refer to arrays as to any other variable using subscript v[3] or v[k]
  - Must assign value to k before using it as variable subscript
  - Major tool in arrays is using variable subscript that is for loop index

```
const int N = 200; double a[N];
for ( int j = 0; j < N; j++) a[j] = 0;
```


Maximum Array Subscript

- Array subscripts start at zero
- A declaration double y[N] declares a y array with N elements numbered from y[0] to y[N-1]
- For loop to handle all elements is

```
for ( int k = 0; k < N; k++ )
```

- C++ does not check to see if an array subscript is in bounds -- an incorrect subscript could affect some other memory location

Arrays and for Loops

- Perhaps the most important array code uses a for loop where the loop index becomes the array subscript

```
const int MAX = 10;
double x[MAX], sum = 0;
// code to input x array goes here
for ( int k = 0; k < MAX; k++ )
    sum += x[k];
```

General Array Processing

- To process each element in an array with N elements, starting with the initial element, use a for loop with index k starting at zero and < N

```
for ( int k = 0; k < N; k++ )
```

- To process a subset of elements in the array starting at element F and ending with (and including) element L

```
for ( int k = F; k <= L; k++)
```

California State University Northridge 73

Array Input and Output

```
const int MAX_SIZE = 100;
double z[MAX_SIZE];
ifstream infile( “array.dat” );
for (int i = 0; i < MAX_SIZE; i++)
{
    infile >> z[i];
    cout << “z[“ << i << “] = ”
         << z[i];
}
```

California State University Northridge 74