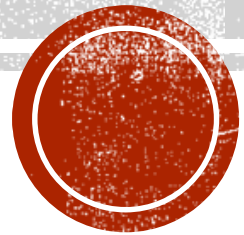
ARRAYS

Lecture 03
Fahad Zafar



7.1 ARRAYS HOLD MULTIPLE VALUES

- An **array** is a data structure which allows a collective name to be given to a group of elements which all have the same type.
- An individual element of an **array** is identified by its own unique index (or subscript).
- An **array** can be thought of as a collection of numbered boxes each containing one data item.
- Unlike regular variables, arrays can hold multiple values.

FIGURE 7-1

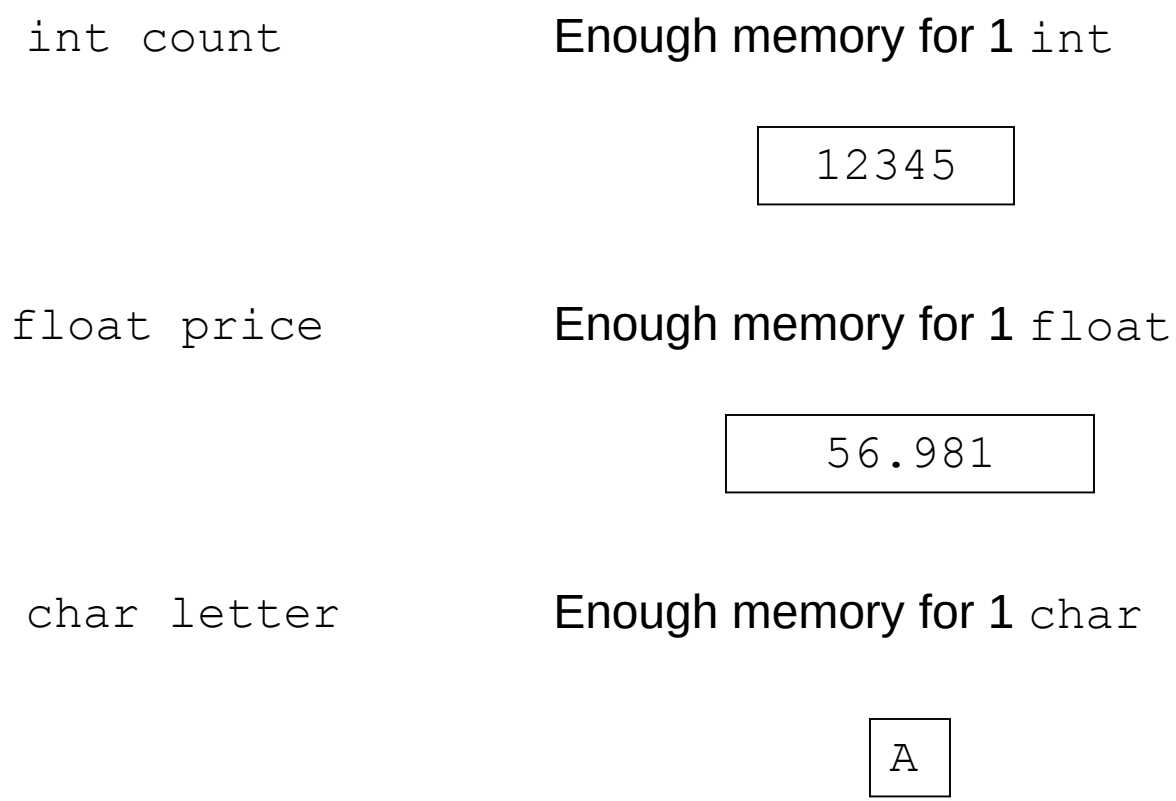

int count
Enough memory for 1 int
12345
float price
Enough memory for 1 float
56.981
char letter
Enough memory for 1 char
A

FIGURE 7-2

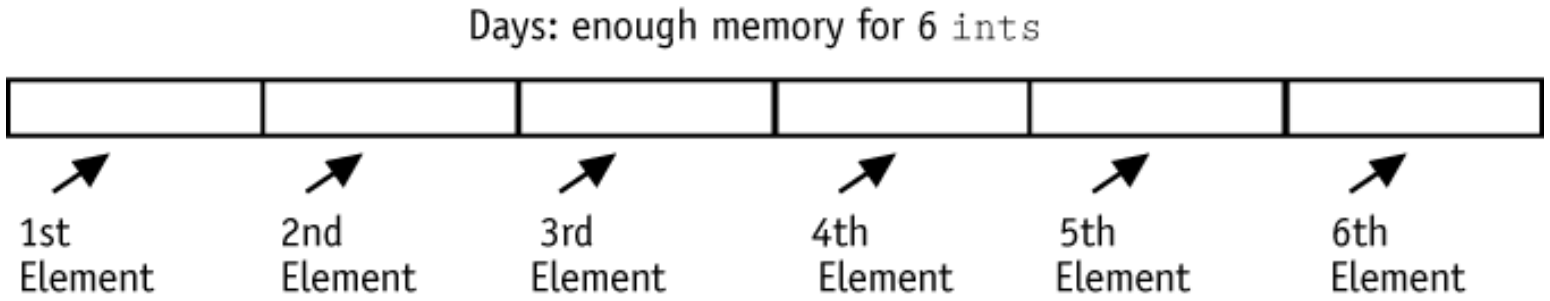

Days: enough memory for 6 ints
1st Element
2nd Element
3rd Element
4th Element
5th Element
6th Element

TABLE 7-1

Array Declaration	Number of Elements	Size of Each Element	Size of the Array
`char letters[25];`	25	1 byte	25 bytes
`short rings[100];`	100	2 bytes	200 bytes
`int miles[84];`	84	4 bytes	336 bytes
`float temp[12];`	12	4 bytes	48 bytes
`doubledDistance[1000];`	1000	8 bytes	8000 bytes

7.2 ACCESSING ARRAY ELEMENTS

- The individual elements of an array are assigned unique subscripts. These subscripts are used to access the elements.

PROGRAM 7-1

```
// This program asks the user for the number of hours worked
// by 6 employees. It uses a 6-element int array to store the
// values.

#include <iostream.h>

void main(void)
{
  short hours[6];

  cout << "Enter the hours worked by six employees: ";
  cin >> hours[0];
  cin >> hours[1];
  cin >> hours[2];
  cin >> hours[3];
```

PROGRAM CONTINUES

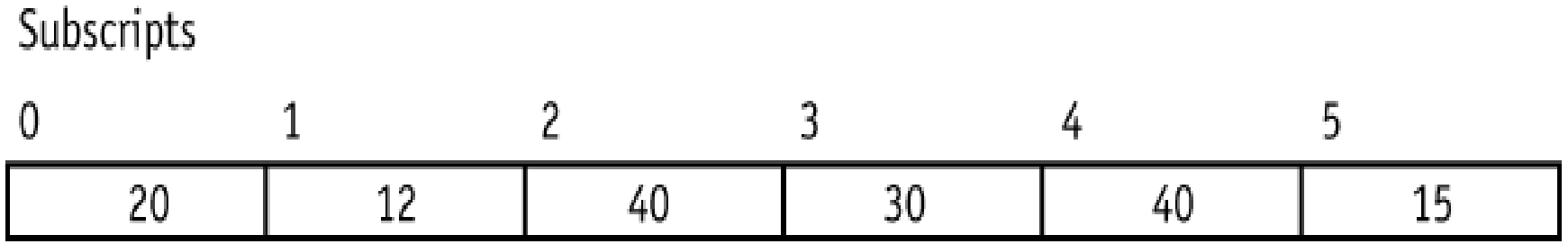
```
 cin >> hours[4];
 cin >> hours[5];
 cout << "The hours you entered are:";
 cout << " " << hours[0];
 cout << " " << hours[1];
 cout << " " << hours[2];
 cout << " " << hours[3];
 cout << " " << hours[4];
 cout << " " << hours[5] << endl;
}
```

PROGRAM OUTPUT WITH EXAMPLE INPUT

Enter the hours worked by six employees: **20 12 40 30 30 15 [Enter]**

The hours you entered are: 20 12 40 30 30 15

FIGURE 7-7



PROGRAM 7-2

```
// This program asks the user for the number of hours worked
// by 6 employees. It uses a 6-element short array to store the
// values.
#include <iostream.h>

void main(void)
{
  short hours[6];

  cout << "Enter the hours worked by six employees: ";
  for (int count = 0; count < 6; count++)
       cin >> hours[count];
  cout << "The hours you entered are:";
  for (count = 0; count < 6; count++)
       cout << " " << hours[count];
  cout << endl;
}
```

PROGRAM OUTPUT WITH EXAMPLE INPUT

Enter the hours worked by six employees: **20 12 40 30 30 15 [Enter]**

The hours you entered are: 20 12 40 30 30 15

PROGRAM 7-3

```
// This program asks the user for the number of hours worked
// by 6 employees. It uses a 6-element short array to store the
// values.
#include<iostream.h>

void main(void)
{
  short hours[6];

  cout << "Enter the hours worked by six employees.\n";
  for (int count = 1; count <= 6; count++)
  {
        cout << "Employee " << count << ": ";
        cin >> hours[count - 1];
  }
  cout << "The hours you entered are\n";
```

PROGRAM CONTINUES

```
 for (count = 1; count <= 6; count++)
 {
      cout << "Employee " << count << ": ";
      cout << hours[count - 1] << endl;
 }
}
```

PROGRAM OUTPUT WITH EXAMPLE INPUT

Enter the hours worked by six employees.
Employee 1: **20 [Enter]**
Employee 2: **12 [Enter]**
Employee 3: **40 [Enter]**
Employee 4: **30 [Enter]**
Employee 5: **30 [Enter]**
Employee 6: **15 [Enter]**
The hours you entered are
Employee 1: 20
Employee 2: 12
Employee 3: 40
Employee 4: 30
Employee 5: 30
Employee 6: 15

7.4 ARRAY INITIALIZATION

- Arrays may be initialized when they are declared.

PROGRAM 7-5

```
// This program displays the number of days in each month.
// It uses a 12-element int array.

#include <iostream.h>

void main(void)
{
  int days[12];
  days[0] = 31;  // January
  days[1] = 28;  // February
  days[2] = 31;  // March
  days[3] = 30;  // April
  days[4] = 31;  // May
  days[5] = 30;  // June
  days[6] = 31;  // July
```

PROGRAM CONTINUES

```
 days[7] = 31;  // August
 days[8] = 30;  // September
 days[9] = 31;  // October
 days[10] = 30; // November
 days[11] = 31; // December
 for (int count = 0; count < 12; count++)
 {
      cout << "Month " << (count + 1) << " has ";
      cout << days[count] << " days.\n";
 }
}
```

PROGRAM OUTPUT

```
Month 1 has 31 days.
Month 2 has 28 days.
Month 3 has 31 days.
Month 4 has 30 days.
Month 5 has 31 days.
Month 6 has 30 days.
Month 7 has 31 days.
Month 8 has 31 days.
Month 9 has 30 days.
Month 10 has 31 days.
Month 11 has 30 days.
Month 12 has 31 days.
```

PROGRAM 7-6

```
// This program displays the number of days in each month.
// It uses a 12-element int array.
#include <iostream.h>

void main(void)
{
  int days[12] = {31, 28, 31, 30,
                  31, 30, 31, 31,
                  30, 31, 30, 31};
  for (int count = 0; count < 12; count++)
  {
        cout << "Month " << (count + 1) << " has ";
        cout << days[count] << " days.\n";
  }
}
```

PROGRAM OUTPUT

```
Month 1 has 31 days.
Month 2 has 28 days.
Month 3 has 31 days.
Month 4 has 30 days.
Month 5 has 31 days.
Month 6 has 30 days.
Month 7 has 31 days.
Month 8 has 31 days.
Month 9 has 30 days.
Month 10 has 31 days.
Month 11 has 30 days.
Month 12 has 31 days.
```

PROGRAM 7-7

```
// This program uses an array of ten characters to store the
// first ten letters of the alphabet. The ASCII codes of the
// characters are displayed.
#include <iostream.h>

void main(void)
{
  char letters[10] = {'A', 'B', 'C', 'D', 'E',
                      'F', 'G', 'H', 'I', 'J'};

  cout << "Character" << "\t" << "ASCII Code\n";
  cout << "--------" << "\t" << "----------\n";
  for (int count = 0; count < 10; count++)
  {
        cout << letters[count] << "\t\t";
        cout << int(letters[count]) << endl;
  }
}
```

PROGRAM OUTPUT

```
Character    ASCII Code
---------    ----------
 A                 65
 B                 66
 C                 67
 D                 68
 E                 69
 F                 70
 G                 71
 H                 72
 I                 73
 J                 74
```

PARTIAL ARRAY INITIALIZATION

- When an array is being initialized, C++ does not require a value for every element.

```
int numbers[7] = {1, 2, 4, 8};
```

PROGRAM 7-8

```
// This program has a partially initialized array.

#include <iostream.h>

void main(void)
{
  int numbers[7] = {1, 2, 4, 8}; // Initialize the
                                 // first 4 elements.

  cout << "Here are the contents of the array:\n";
  for (int index = 0; index < 7; index++)
        cout << numbers[index] << endl;
}
```

PROGRAM OUTPUT

Here are the contents of the array:

1

2

4

8

0

0

0

IMPLICIT ARRAY SIZING

- It is possible to declare an array without specifying its size, as long as you provide an initialization list.

```
float ratings[] = {1.0, 1.5, 2.0, 2.5, 3.0};
```

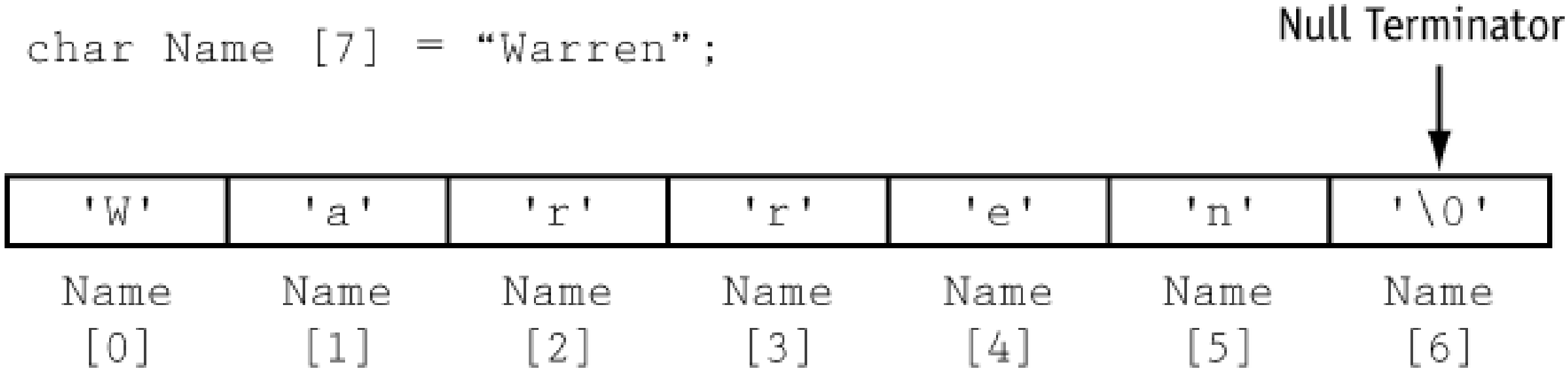
INITIALIZING WITH STRINGS

- When initializing a character array with a string, simply enclose the string in quotation marks:

```
char name[] = "Warren";
```

FIGURE 7-11

```
char Name [7] = "Warren";
```



PROGRAM 7-9

```
// This program displays the contents of two char arrays.
#include <iostream.h>

void main(void)
{
  char name1[] = "Holly";
  char name2[] = {'W', 'a', 'r', 'r', 'e', 'n', '\0'};

  cout << name1 << endl;
  cout << name2 << endl;
}
```

PROGRAM OUTPUT

Holly

Warren

7.8 PRINTING THE CONTENTS OF AN ARRAY

- To display the contents of an array, you must use a loop to display the contents of each element.

```
int array[5] = { 10, 20, 30, 40, 50 };
for (int count = 0; count < 5; count++)
      cout << array[count] << endl;
```

7.9 ARRAYS AS FUNCTION ARGUMENTS

- To pass an array as an argument to a function, pass the name of the array.

PROGRAM 7-13

```
// This program demonstrates that an array element is passed
// to a function like any other variable.
#include <iostream.h>

void ShowValue(int);            // Function prototype

void main(void)
{
   int collection[8] = {5, 10, 15, 20, 25, 30, 35, 40};

   for (int Cycle = 0; Cycle < 8; Cycle++)
            ShowValue(collection[Cycle]);
}
```

PROGRAM CONTINUES

```
//*************************************
// Definition of function showValue.              *
// This function accepts an integer argument.  *
// The value of the argument is displayed.       *
//*************************************
void ShowValue(int Num)
{
   cout << Num << " ";
}
```

PROGRAM OUTPUT

5 10 15 20 25 30 35 40

```
// This program demonstrates an array being passed to a function.
#include <iostream.h>

void showValues(int []);       // Function prototype

void main(void)
{
  int collection[8] = {5, 10, 15, 20, 25, 30, 35, 40};

  showValues(collection);  // Passing address of array collection
}

void showValues(int nums[])
{
  for (int index = 0; index < 8; index++)
        cout << nums[index] << " ";
}
```

PROGRAM OUTPUT

5 10 15 20 25 30 35 40

```
// This program demonstrates an array being passed to a function.
#include <iostream.h>

void showValues(int []);      // Function prototype

void main(void)
{
  int set1[8] = {5, 10, 15, 20, 25, 30, 35, 40};
  int set2[8] = {2, 4, 6, 8, 10, 12, 14, 16};
  showValues(set1);
  cout << endl;
  showValues(set2);
}

void showValues(int nums[])
{
  for (int index = 0; index < 8; index++)
        cout << nums[index] << " ";
}
```

PROGRAM OUTPUT

5 10 15 20 25 30 35 40

2 4 6 8 10 12 14 16

PROGRAM 7-16

```
// This program uses a function that can display the contents
// of an integer array of any size.
#include <iostream.h>

void showValues(int [], int); // Function prototype

void main(void)
{
  int set1[8] = {5, 10, 15, 20, 25, 30, 35, 40};
  int set2[4] = {2, 4, 6, 8};
  int set3[12] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12};

  showValues(set1, 8);
  cout << endl;
  showValues(set2, 4);
  cout << endl;
  showValues(set3, 12);
}
```

PROGRAM CONTINUES

```
//*********************************************
// Definition of function showValues.         *
// This function displays the contents of the *
// array passed into nums. The value passed   *
// into elements is the number of elements in *
// the nums array.                            *
//*********************************************

void showValues(int nums[], int elements)
{
  for (int index = 0; index < elements; index++)
        cout << nums[index] << " ";
}
```

PROGRAM OUTPUT

5 10 15 20 25 30 35 40

2 4 6 8

1 2 3 4 5 6 7 8 9 10 11 12

```
#include <iostream.h>

void doubleArray(int [], int);       // Function prototype
const int arraySize = 12;

void main(void)
{
 int set[arraySize] = {1, 2, 3, 4, 5, 6,
                       7, 8, 9, 10, 11, 12};
 cout << "The arrays values are:\n";
 for (int index = 0; index < arraySize; index++)
      cout << set[index] << " ";
 cout << endl;
 doubleArray(set, arraySize);
 cout << "After calling doubleArray, the values are:\n";
```

PROGRAM CONTINUES

```
  for (int index = 0; index < arraySize; index++)
         cout << set[index] << " ";
  cout << endl;
}

//************************************************
// Definition of function doubleArray.           *
// This function doubles the value of each element *
// in the array passed into nums.                 *
// The value passed into size is the number of    *
// elements in the nums array.                    *
//************************************************
void doubleArray(int nums[], int size)
{
  for (int index = 0; index < size; index++)
         nums[index] *= 2;
}
```

PROGRAM OUTPUT

The array values are:

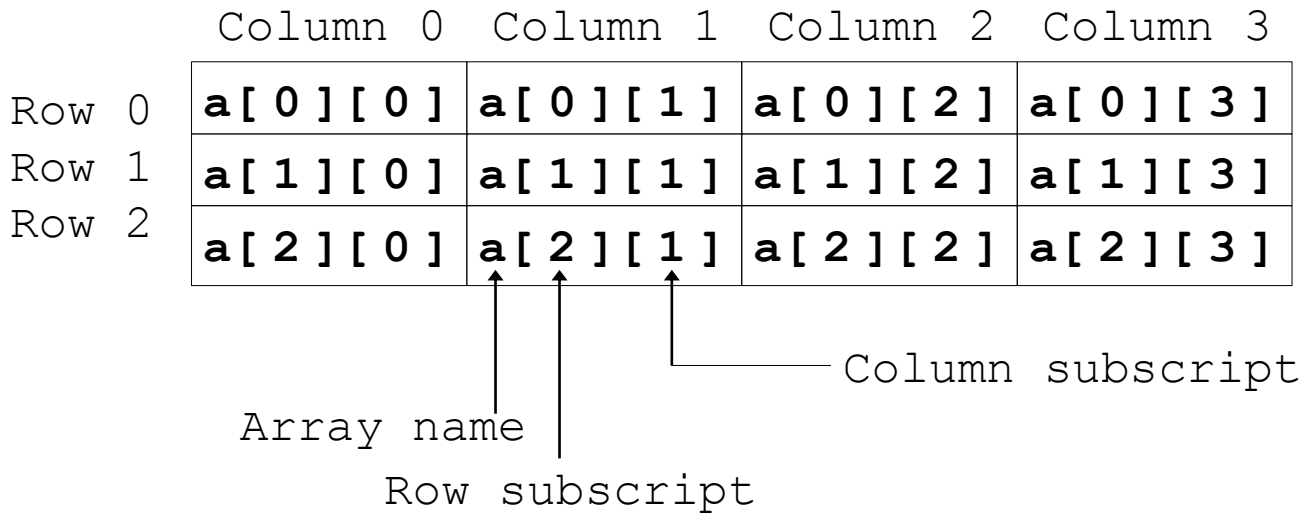
1 2 3 4 5 6 7 8 9 10 11 12

After calling doubleArray, the values are:

2 4 6 8 10 12 14 16 18 20 22 24

MULTIPLE-SUBSCRIPTED ARRAYS

- Multiple subscripted arrays
  - Tables with rows and columns (**m** by **n** array)
  - Like matrices: specify row, then column



MULTIPLE-SUBSCRIPTED ARRAYS

- Initialization
  - `int b[ 2 ][ 2 ] = { { 1, 2 }, { 3, 4 } };`
  - Initializers grouped by row in braces
  - If not enough, unspecified elements set to zero

    `int b[ 2 ][ 2 ] = { { 1 }, { 3, 4 } };`

1	2
3	4

1	0
3	4

- Referencing elements
  - Specify row, then column

    `cout<< b[ 0 ][ 1 ] ;`

DECLARING ARRAYS OF OBJECTS

- Declaring arrays of objects is similar to declaring arrays of built-in types

```
Fraction rationals[20]; // array of 20 Fraction objects
Complex nums[50]; // an array of 50 Complex objects
Hydrant fireplugs[10]; // an array of 10 fireplugs
```

- Each array position is a single object
  - 'Fraction rationals[20];' declares 20 Fraction objects, rationals[0], rationals[1], …, rationals[19].

INITIALIZING THE ARRAY OF OBJECTS

- Similar to a number array declaration.
  - Do nothing to use the default constructor

```
int x;
Fraction num;
Fraction num[4];
```

  - To initialize in a particular way, call an explicit constructor

```
Int x(10);
Fraction num(10, 20);
```

  - How to do array of objects? Need a way to specify different constructors to different elements.

INITIALIZING THE ARRAY OF OBJECTS

- To initialize in a particular way, call an explicit constructor

```
Int x(10);
Fraction num(10, 20);
```

- How to do array of objects? Need a way to specify different constructors to different elements.
  - Use an initializer set to give a constructor to each element

```
Fraction numlist[3] = {Fraction(2, 4),  Fraction(5), Fraction()};
```

  - numlist[0] is initialized with constructor Fraction(2,4);
  - numlist[1] is initialized with constructor Fraction(5);
  - numlist[2] is initialized with constructor Fraction();

USING THE ARRAY OF OBJECTS

- Indexing works the same as with regular arrarys
  - Each object in the array is in the form of ***arrayName[index];***
- The dot-operator works the same as with single names.

  objectName.memberName
- The objectName is in the from of an array item:
  - ***arrayName[index].memberName***

- Example

```
Fraction rationals[20];
…
rationals[2].show();
rationals[6].Input();
for (i=0; i<10; i++) rationals[i].setval(20);
for(i=0; i<20; i++) rationals[i].putval = 50;
```