Ch 7. Arrays
Part 1

CS 1428
Fall 2011

Jill Seaman

Lecture 17

Array Data Type

- How many variables do we need for the following problems?
  - Calculate the grade for 1 student in CS1428
    - 8 assignments, 3 exams, 1 lab score, attendence+quizzes
  - Calculate the grades for 30 CS1428 students
  - Read in 1000 integers, output the number of values that are above the mean.

Array Data Type

- Arrays provide a way to
  - declare multiple “variables” at once and
  - refer to these variables using one common name
- So far we have used scalar/primitive data types
  - each variable holds only one value
- Composite data types:
  - a single variable can contain multiple values
  - an array is a composite data type

Array Data Type

- An array contains multiple values of the *same type*.
- values are stored consecutively in memory.
- An array definition in C++:

```
int numbers[10];
```

- Name of the array: `numbers`
- 10 is the size declarator:
  the number of elements (values)
- int is the type of each of the 10 elements

More Arrays

- More examples:

```
float temperatures[100];
char name[51];
long units[50];
```

- The size must be an *integer* and a *constant*:
  - a literal or named constant

```
const int SIZE = 40;
double grades[SIZE];
```


Memory allocation

- When an array is defined, all of the memory it needs is allocated.

```
int numbers[10];
```

- An int requires 4 bytes
- numbers array requires 10 integers:
  - 10 integers * 4 bytes = 40 bytes
- The memory is allocated sequentially

Array Elements

- Individual elements of the array have unique subscripts (index)
- The subscripts are 0-based
  - the first element has subscript 0
  - the second element has subscript 1
  - . . .
  - the last element has subscript (size -1)
- Syntax to access one element:

```
numbers[2]  //the third element of numbers array
```

- Called “numbers at 2” or “numbers sub 2”

Array subscripts

- Square brackets in *definition* indicate size
- Square brackets in an *expression* indicate subscript.
- the subscript is always an integer, regardless of the type of the array elements.
- the subscript can be ANY integer expression
  - literal: 2
  - variable: i
  - expression: (i+2)/2

Array subscripts

- Given the following array definition:

```
double numArray[10];
```

the expression `numArray[i]` may be used exactly like any variable of type double.

Using array elements

- Examples of using array elements.

```
double values[3];

values[0] = 22.3;
values[1] = 11.1;

cout << "Enter a number: ";
cin >> values[2];

double sum = values[0] + values[1] + values[2];
double avg = sum/3.0;

cout << "Values at zero: " << values[0] << endl;

int i=2;
if (values[i] > 32.0)
   cout << "Above freezing" << endl;
```

Array initialization

- You can initialize arrays when they are defined.

```
const int NUM_SCORES = 3;
float scores[NUM_SCORES] = {86.5, 92.1, 77.5};
```

- Values are assigned in order:

  scores[0] = 86.5

  scores[1] = 92.1

  scores[2] = 77.5

- NOTE: uninitialized arrays have unknown values stored in them (not necessarily 0).

Partial array initialization

- You can initialize only the first part of the array.

```
const int NUM_SCORES = 10;
float scores[NUM_SCORES] = {86.5, 92.1, 77.5};
```

- The first three elements get the values specified.
- The remaining 7 elements get initialized to 0.0.
- The list of elements cannot have **more** elements than the size of the array.

Implicit array sizing via initialization

- When you initialize, you don't need to specify the size.

```
float scores[] = {86.5, 92.1, 77.5};
```

- The size of the array is the number of elements listed.

Arrays of char (review)

- We have already seen arrays of char:

```
char word[] = "football";   //automatically adds '\0'
```

- The size of the array is the length of the string *plus one* (for the null character) so 9 here.
- Can also use a list of chars to initialize:

```
char word[] = {'f','o','o','t','b','a','l','l','\0'};
```

- Must include the null character in this case.
- If you forget the null character, operations on the char array may not behave correctly.

Arrays of char (review)

- Arrays of char in C++ are called “C-Strings”
- Note: Arrays of char are sometimes handled differently from other arrays.
- For example, you can output an array of char

```
char word[] = {'f','o','o','t','b','a','l','l','\0'};
cout << word << endl;  // outputs: football
```

- But you cannot output an array of int:

```
int numbers[] = {1, 2, 3};
cout << numbers << endl;   // won't work like you want
```


Operations over arrays

- Generally there are NO operations you can perform over entire arrays.
- Some operations may *appear* to work (no errors) but you don’t get the desired results.

```
int numbers1[] = {1, 2, 3};
int numbers2[] = {4, 5, 6};

cin >> numbers1;            //input, won't work
cout << numbers1 << endl;   //output, won't work
numbers1 = numbers2;        //assignment, won't work
if (numbers1==numbers2)     //comparison, won't work
  ...
numbers3 = numbers1 + numbers2; //addition, won't work
```

- exception: *can* input+output entire char arrays