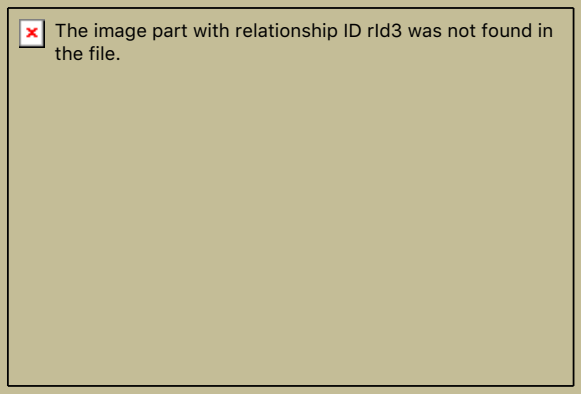
COMPUTER SCIENCE 1: STARTING COMPUTING
CSCI 1300



Ioana Fleming / Vipra Gupta
Spring 2018
Lecture 16

University of Colorado Boulder

Agenda

- Today
  - Arrays – passing to functions, again
  - Multidimensional Arrays

Announcements

- Rec 7 due on 3/3
- Hmwk 5 due on 3/4
- Practicum 2: March 12th - 18th, 2018
  - Loops: while, for
  - Strings
  - Arrays
  - File I/O

Array as function argument

- What does the computer know about an array?
  - The base type
  - The address of the first indexed variable
  - The number of indexed variables
- What does a function know about an array argument?
  - The base type
  - The address of the first indexed variable

Entire Arrays as Arguments

- Formal parameter can be entire array
  - Argument then passed in function call is array name
  - Called "array parameter"
- Send size of array as well
  - Typically done as second parameter
  - Simple int type formal parameter

Entire Array as Argument Example

- In some main() function definition, consider this calls:

```
int score[5], numberOfScores = 5;
fillup(score, numberOfScores);
```

  - 1st argument is entire array
  - 2nd argument is integer value
- Note no brackets in array argument!

Array as Argument: How?

- What's really passed?
- Think of array as 3 "pieces"
  - Address of first indexed variable (`arrName[0]`)
  - Array base type (int or double or float or string, …)
  - Size of array
- Only 1st piece is passed!
  - Just the beginning address of array (the 1st element)
  - Knowing the type helps us retrieve the (2nd – last) elements

Array Parameters

- May seem strange
  - No brackets in array argument
  - Must send size separately
- One nice property:
  - Can use SAME function to fill any size array!
  - Exemplifies "re-use" properties of functions
  - Example:

```
int score[5], time[10];
fillUp(score, 5);
fillUp(time, 10);
```

Cloud9 example: *array_search.cpp*

The const Parameter Modifier

- Recall: array parameter actually passes address of 1st element
- Function can then modify array!
  - Often desirable, sometimes not!
- Protect array contents from modification
  - Use "const" modifier before array parameter
    - Called "constant array parameter"
    - Tells compiler to "not allow" modifications

Example – function definition

```
void addarray(int size,          // IN size of arrays
              const float A[],   // IN input array
              const float B[],   // IN input array
              float C[])         // OUT result array

// Takes two arrays of the same size as input parameters
// and outputs an array whose elements are the sum of the
// corresponding elements in the two input arrays.
{
    int i;
    for (i = 0; i < size; i++)
         C[i] = A[i] + B[i];

  } // End of function addarray
```

Example – function call

The function addarray could be used as follows:

In main():

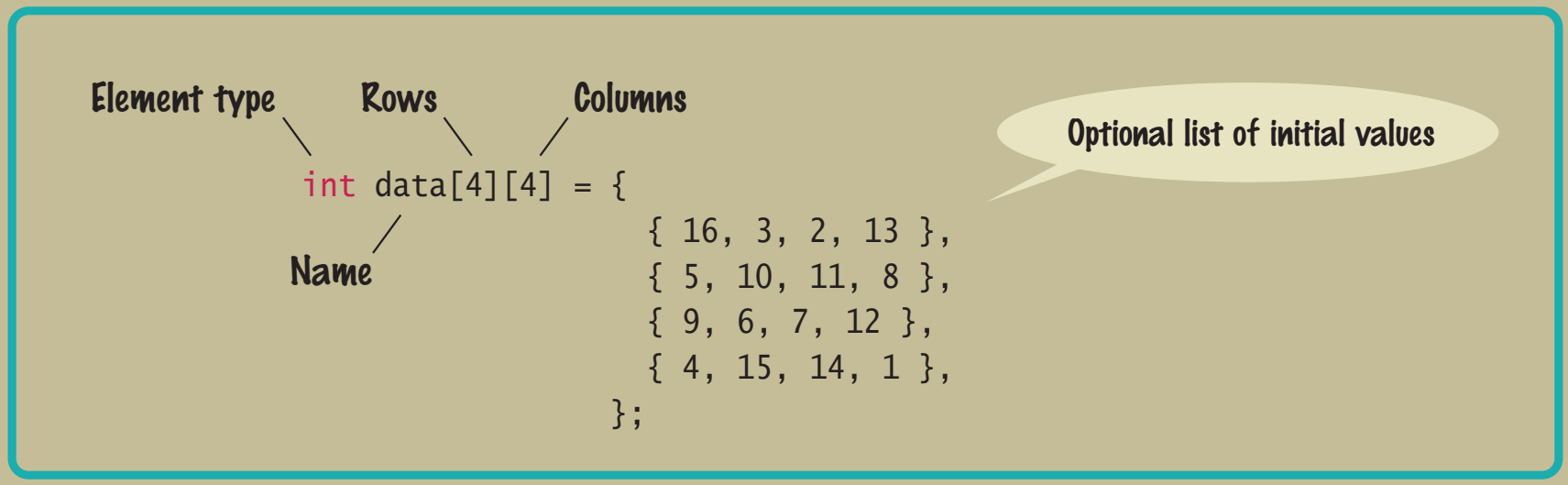
```
int one[50], two[50], three[50];
            //
            //
addarray(50, one, two, three);

// but also:
addarray(20, one, two, three);

// it will only do the addition on the first 20 elements
of each array
```

Multidimensional Arrays

- Arrays with more than one index
  - char page[30][100];
    - Two indexes: An "array of arrays"
    - Visualize as:
      page[0][0], page[0][1], …, page[0][99]
      page[1][0], page[1][1], …, page[1][99]
      …
      page[29][0], page[29][1], …, page[29][99]
- C++ allows any number of indexes
  - Typically no more than two

Defining 2D arrays

Two-Dimensional Array Definition



Multidimensional Array Parameters

- Similar to one-dimensional array
  - 1st dimension size not given
    - Provided as second parameter
  - 2nd dimension size IS given
- Example:

```
void DisplayPage(const char p[][100], int
sizeDimension1)
{
 for (int index1=0; index1<sizeDimension1; index1++)
 {
    for (int index2=0; index2 < 100; index2++)
       cout << p[index1][index2];
    cout << endl;
 }
}
```

Omitting the Column size of a two-dimensional Array Parameter

When passing a one-dimensional array to a function, you specify the size of the array as a separate parameter variable:

```
void print(double values[], int size)
```

This function can print arrays of any size. However, for two-dimensional arrays you cannot simply pass the numbers of rows and columns as parameter variables:

```
void print(double table[][], int rows, int cols) //NO!

const int COLUMNS = 3;
void print(const double table[][COLUMNS], int rows) //OK
```

This function can print tables with any number of rows, but the column size is fixed.

Summary 1

- Array is collection of "same type" data
- Indexed variables of array used just like any other simple variables
- for-loop "natural" way to traverse arrays
- Programmer responsible for staying "in bounds" of array
- Array parameter is "new" kind

Summary 2

- Array elements stored sequentially
  - "Contiguous" portion of memory
  - Only address of 1st element is passed to functions
- Partially-filled arrays → more tracking
- Constant array parameters
  - Prevent modification of array contents
- Multidimensional arrays
  - Create "array of arrays"

Programming with Arrays

- Plenty of uses
  - Partially-filled arrays
    - Must be declared some "max size"
  - Sorting
  - Searching

Partially-filled Arrays

- Difficult to know exact array size needed
- Must declare to be largest possible size
  - Must then keep "track" of valid data in array
  - Functions dealing with the array may not need to know the declared size of the array, only how many elements are stored in the array
    - int numberUsed;
    - Tracks current number of elements in array