COMPUTER SCIENCE 1: STARTING COMPUTING CSCI 1300

The image part with relationship ID rId3 was not found in the file.

Ioana Fleming / Vipra Gupta

Spring 2018

Lecture 14

Agenda

- Today
  - Arrays
  - 2D Array

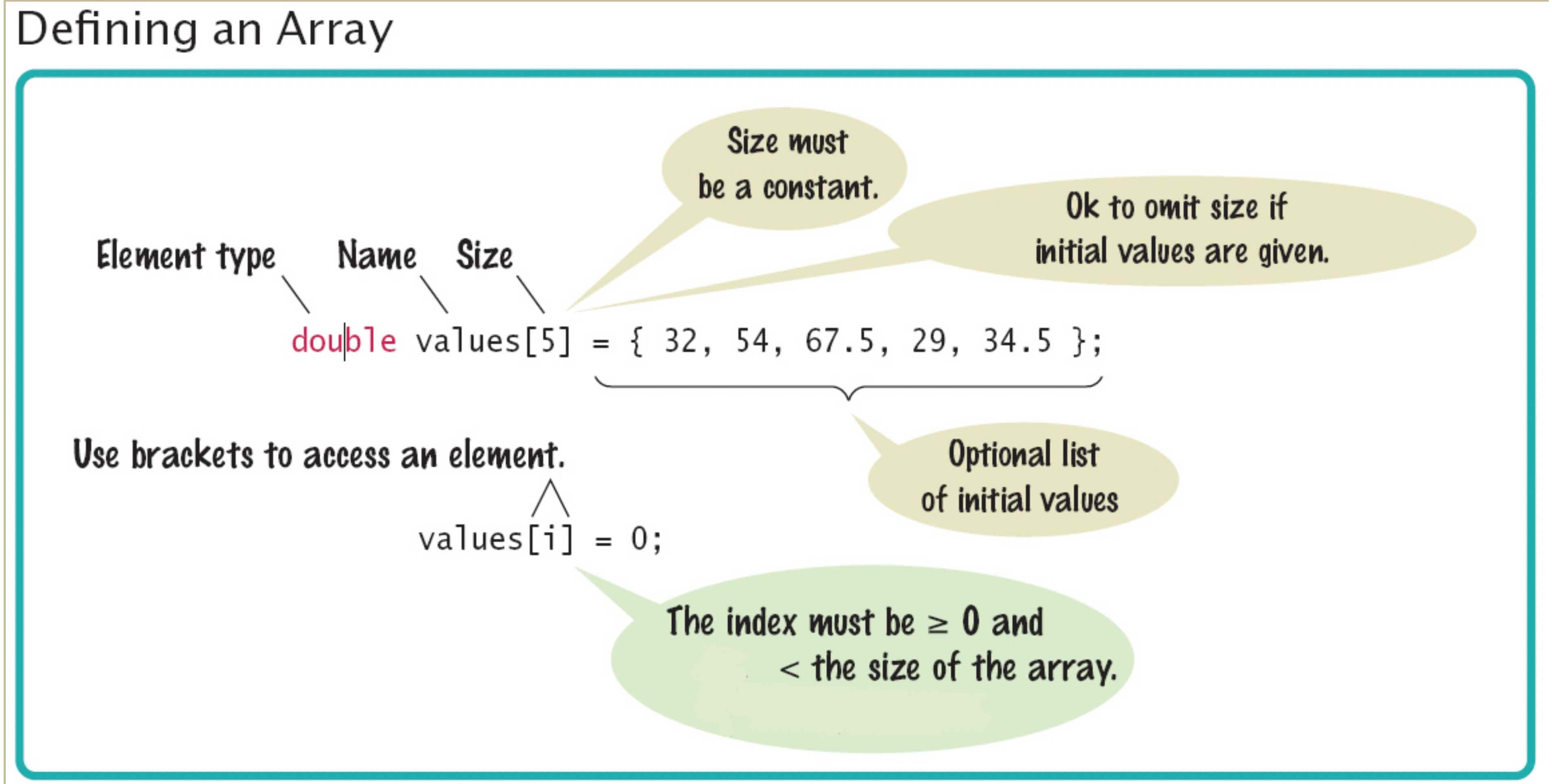
Announcements

- Rec 6 due on 2/24
- Rec 7 due on 3/3
- Hmwk 5 due on 3/4
- Practicum 1: February 21st, 2018
  - Good Job!

Declaring Arrays

- Declare the array → allocates memory
  `int score[5];`
  - Declares array of 5 integers named "score"
  - Similar to declaring five variables:
    `int score[0], score[1], score[2], score[3], score[4]`
- Individual parts can be called many things:
  - Indexed or subscripted variables
  - "Elements" of the array
  - Value in brackets is called index or subscript
    - Numbered from 0 to (size – 1)

Array Syntax


Defining an Array
Size must be a constant.
Ok to omit size if initial values are given.
Element type
Name
Size
double values[5] = { 32, 54, 67.5, 29, 34.5 };
Optional list of initial values
Use brackets to access an element.
values[i] = 0;
The index must be ≥ 0 and < the size of the array.

Accessing Arrays

- Access using index/subscript

```
cout << score[3];
```

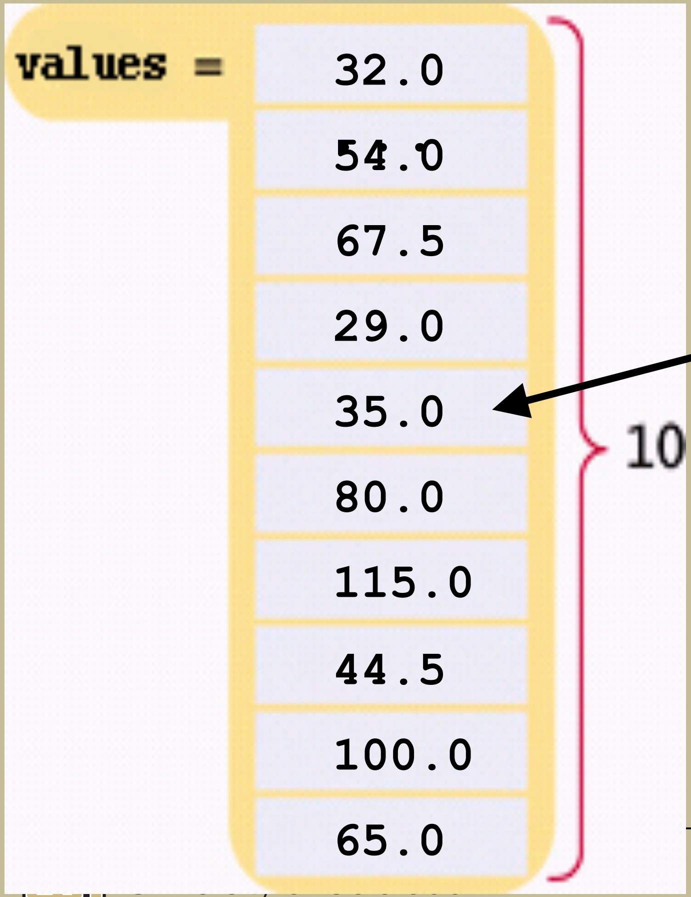
- Note two uses of brackets:
  - In declaration, specifies SIZE of array
  - Anywhere else, specifies a subscript
- Size, subscript need not be literal

```
int score[MAX_SCORES];
score[n+1] = 99;
```

  - If n is 2, identical to: score[3]

Accessing an Array Element

To access the element at index 4 using this notation: **`values[4]`**
4 is the *index*.



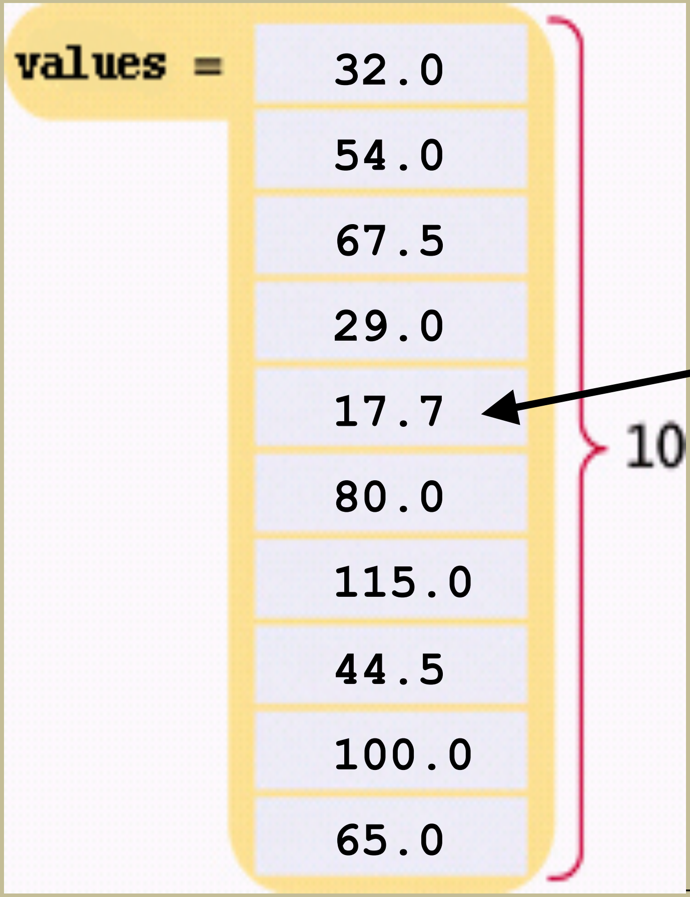
```
double values[10];

cout << values[4] << endl;
```

The output will be `35.0`.

Accessing an Array Element

The same notation can be used to change the element.



```
values[4] = 17.7;
cout << values[4] << endl;
```

The output will be `17.7`.

Accessing an Array Element

That is, the legal elements for the **`values`** array are:

`values[0]`, the ***first*** element
`values[1]`, the second element
`values[2]`, the third element
`values[3]`, the fourth element
`values[4]`, the fifth element
`. . .`
`values[9]`, the tenth ***and last legal*** element
recall: **`double values[10];`**

The index must be **`>= 0`** and **`<= 9`**.
0, 1, 2, 3, 4, 5, 6, 7, 8, 9 is 10 numbers.

Array Usage

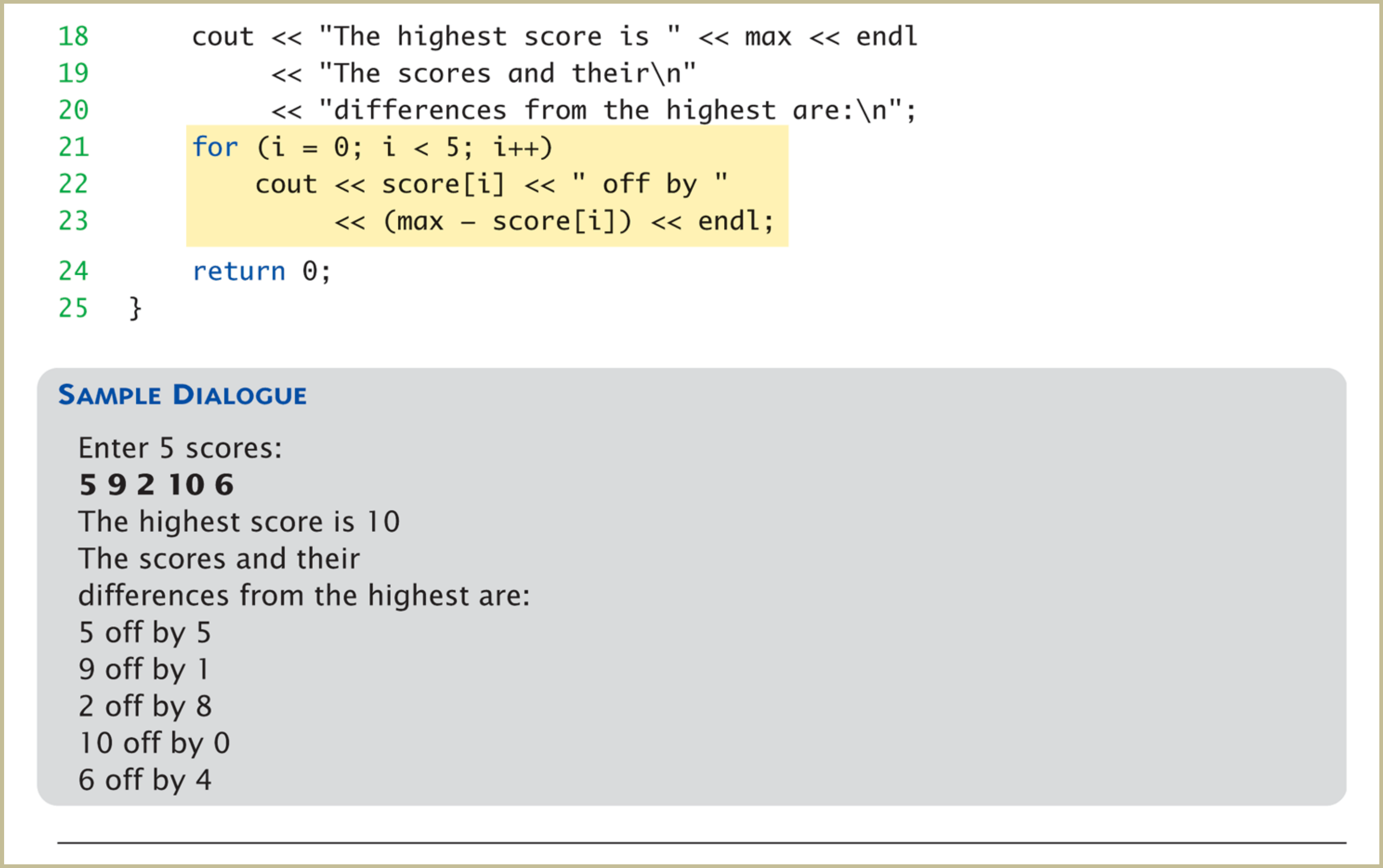
- Powerful storage mechanism
- Can issue commands like:
  - "Do this to i^{th} indexed variable", where i is computed by program
  - "Display all elements of array score"
  - "Fill elements of array score from user input"
  - "Find highest value in array score"
  - "Find lowest value in array score"
- Disadvantages: size MUST BE KNOWN at declaration

Array Program Example (posted on Moodle)

Display 5.1 Program Using an Array

```
//Reads in five scores and shows how much each
//score differs from the highest score.
#include <iostream>
using namespace std;
int main( )
{
    int i, score[5], max;
    cout << "Enter 5 scores:\n";
    cin >> score[0];
    max = score[0];
    for (i = 1; i < 5; i++)
    {
        cin >> score[i];
        if (score[i] > max)
            max = score[i];
        //max is the largest of the values score[0],..., score[i].
    }
```

Array Program Example: **Display 5.1** Program Using an Array (2 of 2)



```
    cout << "The highest score is " << max << endl
         << "The scores and their\n"
         << "differences from the highest are:\n";
    for (i = 0; i < 5; i++)
        cout << score[i] << " off by "
             << (max - score[i]) << endl;

    return 0;
}
```

Sample Dialogue

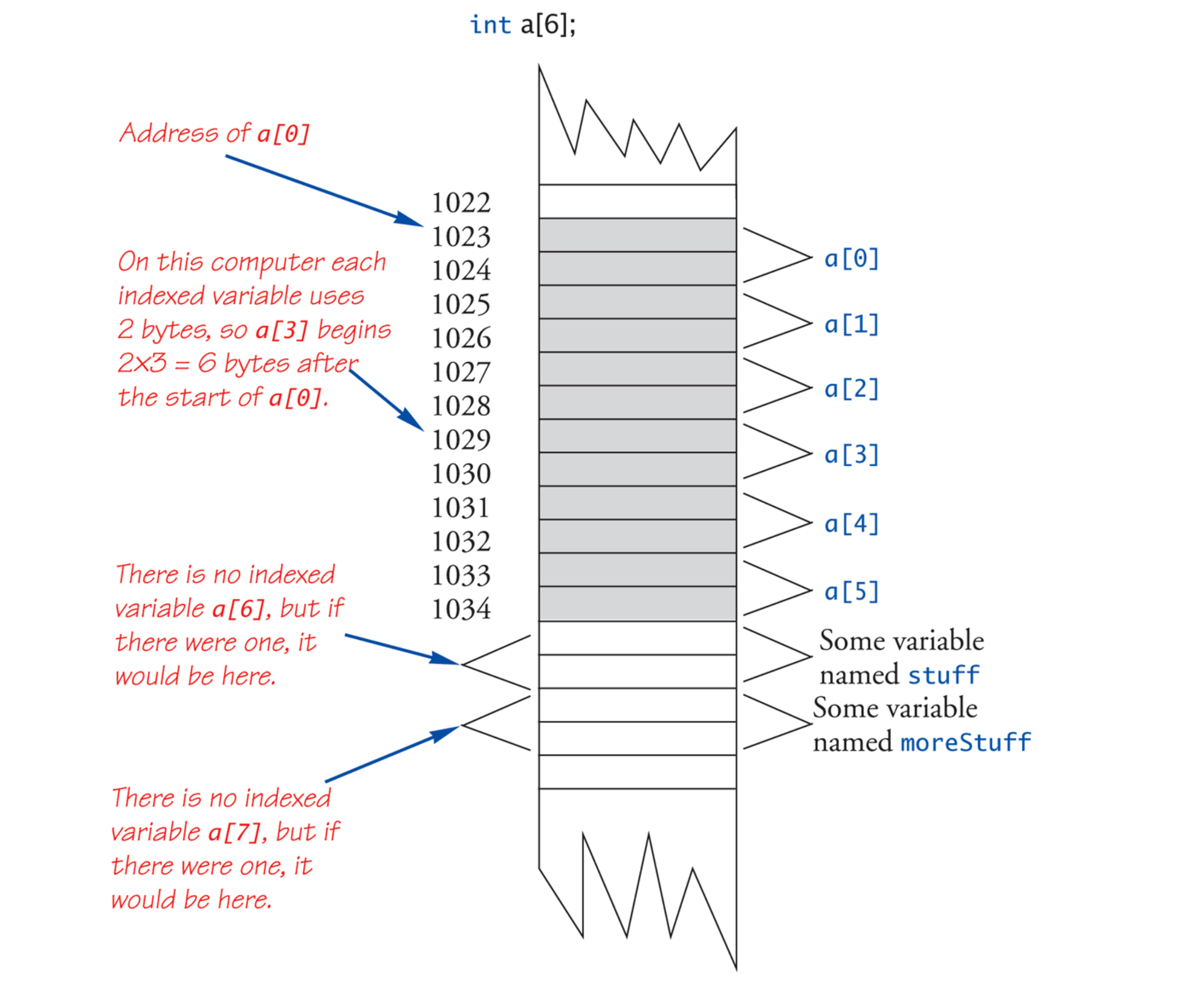
```
Enter 5 scores:
5 9 2 10 6
The highest score is 10
The scores and their
differences from the highest are:
5 off by 5
9 off by 1
2 off by 8
10 off by 0
6 off by 4
```

Arrays in Memory

- Recall simple variables:
  - Allocated memory in an "address"
- Array declarations allocate memory for entire array
- Sequentially-allocated
  - Means addresses allocated "back-to-back"
  - Allows indexing calculations
    - Simple "addition" from array beginning (index 0)

An Array in Memory

Display 5.2 An Array in Memory



int a[6];
Address of a[0]
On this computer each indexed variable uses 2 bytes, so a[3] begins 2x3 = 6 bytes after the start of a[0].
There is no indexed variable a[6], but if there were one, it would be here.
There is no indexed variable a[7], but if there were one, it would be here.
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
a[0]
a[1]
a[2]
a[3]
a[4]
a[5]
Some variable named stuff
Some variable named moreStuff

University of Colorado Boulder

Auto-Initializing Arrays

- If fewer values than size supplied:
  - Fills from beginning
  - Fills "rest" with zero of array base type
- If array-size is left out
  - Declares array with size required based on number of initialization values
  - Example:

    ```
    int b[] = {5, 12, 11};
    ```

    - Allocates array b to size 3

Array Syntax

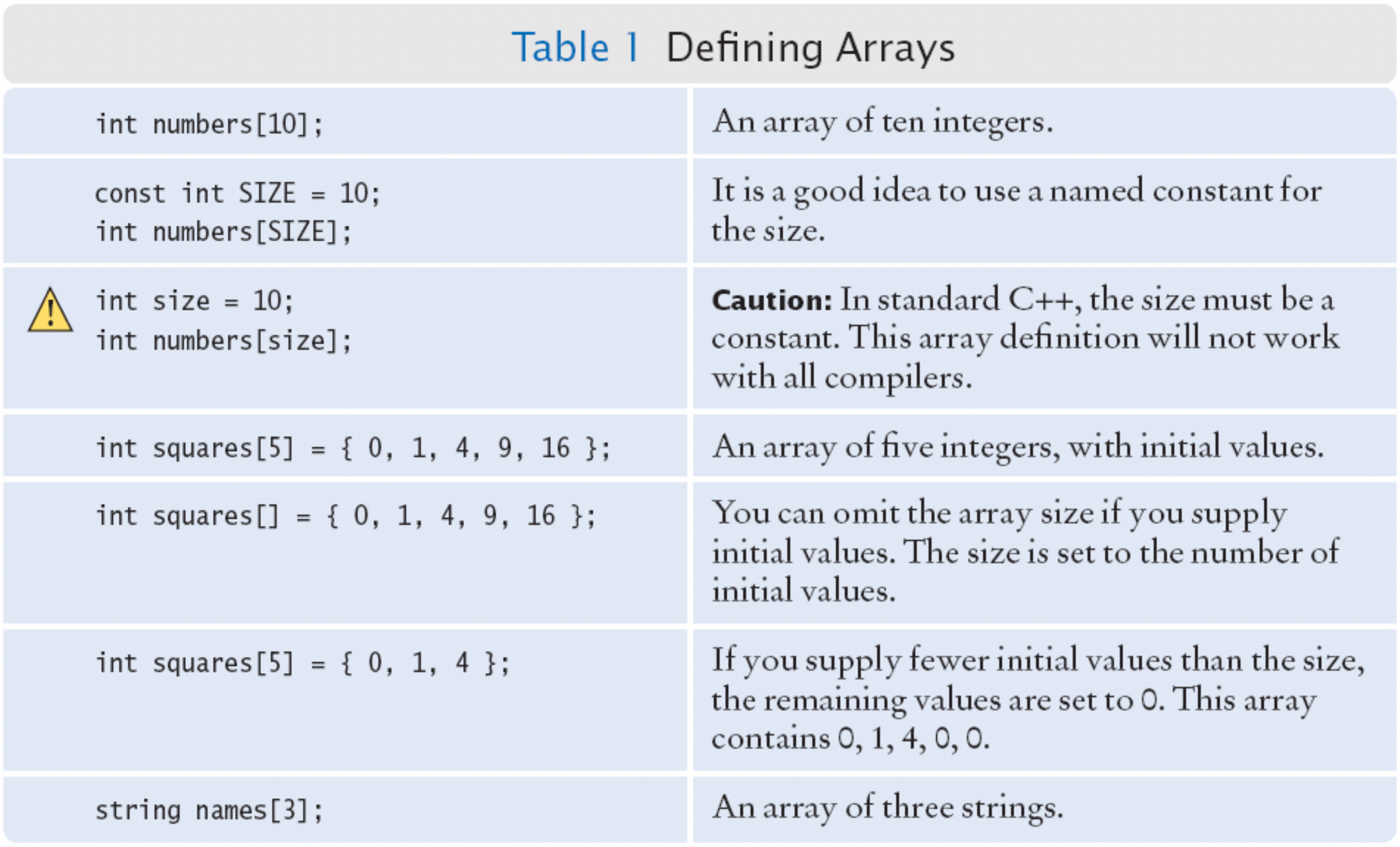


Table 1 Defining Arrays

`int numbers[10];`	An array of ten integers.

`const int SIZE = 10;`
`int numbers[SIZE];`
	It is a good idea to use a named constant for the size.

⚠ `int size = 10;` `int numbers[size];`	**Caution:** In standard C++, the size must be a constant. This array definition will not work with all compilers.
`int squares[5] = { 0, 1, 4, 9, 16 };`	An array of five integers, with initial values.
`int squares[] = { 0, 1, 4, 9, 16 };`	You can omit the array size if you supply initial values. The size is set to the number of initial values.
`int squares[5] = { 0, 1, 4 };`	If you supply fewer initial values than the size, the remaining values are set to `0`. This array contains `0`, 1, 4, `0`, `0`.
`string names[3];`	An array of three strings.

Arrays in Functions

- As arguments to functions
  - Indexed variables
    - An individual "element" of an array can be function parameter
  - Entire arrays
    - All array elements can be passed as "one entity”

Indexed Variables as Arguments

- Indexed variable handled same as simple variable of array base type (int, double, string, …)
- Given this function declaration:

```
void myFunction(double par1);
```

- And these declarations:

```
int i;

double n, a[10];
```

- These function calls are allowed:

```
myFunction(i);  // i is converted to double
myFunction(a[3]); // a[3] is double
myFunction(n);  // n is double
```

Subtlety of Indexing

- Consider:

```
myFunction(a[i]);
```

  - Value of i is determined first
    - It determines which indexed variable is sent

```
myFunction(a[i*5]);
```

  - Perfectly legal, from compiler's view
  - Programmer responsible for staying "in-bounds" of array

Array as argument - details

- 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
  - Size of array
- Only 1st piece is passed!
  - Just the beginning address of array

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: *medals_1D.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

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;
 }
}
```

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