

UNIVERSITY OF
Nebraska
Lincoln

Declaring Arrays I

CSCE150A

Introduction

Declaring,
Referencing

Initialization

Function Args

Constants

Returning
Arrays

Searching &
Sorting

Multi-
dimensional
Arrays

Common
Errors

Exercises

- An array is a collection of two or more adjacent memory cells, called **array elements**
- All elements in an array are associated with a single variable name
- Each element is individually accessed using *indices*

< > <> <><> <><><> <><><><>

[illegible]



Declaring Arrays II

CSCE150A

Introduction

Declaring,
Referencing

Initialization

Function Args

Constants

Returning
Arrays

Searching &
Sorting

Multi-
dimensional
Arrays

Common
Errors

Exercises

- To set up an array in memory, we declare both the *name of the array* and the *number of cells* associated with it:

```
double my_first_array[8];
int students[10];
```

- The first one instructs C to associate 8 memory cells of type **double** with the name **my_first_array**
- The second one instructs C to associate 10 memory cells of type **int** with the name **students**
- In all cases, the memory cells will be adjacent to each other in memory



Notes



Referencing Array Elements I

CSCE150A

Introduction

Declaring, Referencing

Initialization

Function Args

Constants

Returning Arrays

Searching & Sorting

Multi-dimensional Arrays

Common Errors

Exercises

- To process the data stored in an array, each individual element is associated to a reference value
- By specifying the *array name* and identifying the element desired, we can access a particular value
- The subscripted variable `x[0]` (read as *x* sub zero) references the *first* element



[illegible]

Notes

Array Declaration & Initialization

CSCE150A

Introduction
Declaring,
Referencing
Initialization
Function Args
Constants
Returning
Arrays
Searching &
Sorting
Multi-
dimensional
Arrays
Common
Errors
Exercises
10 / 30

- We can declare and initialize an array
- If we initialize when we declare, we can omit the size

```
1 int primeNumbersLessThanHundred[] = {  
2   2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37,  
3   41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83,  
4   89, 97 };
```

Notes

Using for Loops for Sequential Access

CSCE150A

Introduction
Declaring,
Referencing
Initialization
Function Args
Constants
Returning
Arrays
Searching &
Sorting
Multi-
dimensional
Arrays
Common
Errors
Exercises
11 / 30

- Elements of an array are processed in sequence, starting with element **zero**.
- This processing can be done easily using an indexed `for` loop: a counting loop whose loop control variable runs from zero to one less than the array size.
- Using the loop counter as an array index (subscript) gives access to each array element in turn.

```
1 for(i=0; i < SIZE; i++) {  
2     printf("%d ", array[i]);  
3 }
```

Notes

Using Array Elements as Function Arguments

CSCE150A

Introduction
Declaring,
Referencing
Initialization
Function Args
Constants
Returning
Arrays
Searching &
Sorting
Multi-
dimensional
Arrays
Common
Errors
Exercises
12 / 30

You can use `scanf` with array elements just like with regular variables

```
1 int x[10];  
2 int i = 0;  
3 scanf("%d", &x[i]);  
4 printf("Hey, I read %d\n", x[i]);
```

Notes

Arrays as Arguments

CSCE150A

Introduction
Declaring,
Referencing
Initialization
Function Args
Constants
Returning
Arrays
Searching &
Sorting
Multi-
dimensional
Arrays
Common
Errors
Exercises
15/30

- You can also use entire arrays as function arguments
- Passing arrays as arguments to a function means:
 - The function can access any value in the array
 - The function can change any value in the array
- Syntax: specify an array as a parameter by using the square brackets:
`int sum(int array[], int size);`
- Note: what is *actually* being passed is a *pointer* to the first element of the array!
- We could equivalently define:
`int sum(int *array, int size);`

◀ ▶ ◀ ▶ ◀ ▶ ◀ ▶ ◀ ▶ ◀ ▶ ◀ ▶

Notes

Full Example

CSCE150A

Introduction
Declaring,
Referencing
Initialization
Function Args
Constants
Returning
Arrays
Searching &
Sorting
Multi-
dimensional
Arrays
Common
Errors
Exercises
14/30

```
1 #include <stdio.h>
2
3 int sum(int array[], int size);
4
5 int main(void)
6 {
7     int foo[] = {1,2,3,4,5,6,7,8,9,10}, i;
8     printf("sum of all array elements is %d\n", sum(foo, 10));
9     return 0;
10 }
11
12 int sum(int a[], int size)
13 {
14     int i, summation = 0;
15     for(i=0; i<size; i++)
16     {
17         summation += a[i];
18     }
19     return summation;
20 }
```

◀ ▶ ◀ ▶ ◀ ▶ ◀ ▶ ◀ ▶ ◀ ▶ ◀ ▶

Notes

Formal Array Parameter

CSCE150A

Introduction
Declaring,
Referencing
Initialization
Function Args
Constants
Returning
Arrays
Searching &
Sorting
Multi-
dimensional
Arrays
Common
Errors
Exercises
15/30

- It was necessary to pass an additional variable `size` to `sum`
- An array does not have an explicit size associated with it
- C does not allocate space in memory for arrays; the operating system does this *at runtime*
- As programmers, we are responsible for:
 - Memory management
 - For keeping track of the size of an array
 - For ensuring that we do not access memory outside the array
- If a function accesses an array, it needs to be told how big it is

◀ ▶ ◀ ▶ ◀ ▶ ◀ ▶ ◀ ▶ ◀ ▶ ◀ ▶

Notes

Arrays as Input Arguments

CSCE150A

Introduction
Declaring,
Referencing
Initialization
Function Args
Constants
Returning
Arrays
Searching &
Sorting
Multi-
dimensional
Arrays
Common
Errors
Examples
16 / 30

- Since arrays are passed *by reference* (like `scanf`), functions can modify their values
- Sometimes, we would like to pass arrays as arguments, but do not want to change their values.
- We can do this by using the `const` quantifier in the function declaration: `int sum(const int foo[], int size) ...`
- Specifies to the compiler that the array is to be used *only* as an input
- The function does not intend to modify the array
- The compiler enforces this: any attempt to change an array element in the function as an error

Notes

Returning an Array Result

CSCE150A

Introduction
Declaring,
Referencing
Initialization
Function Args
Constants
Returning
Arrays
Searching &
Sorting
Multi-
dimensional
Arrays
Common
Errors
Examples
17 / 30

- C only allows us to return a single item
- It is not possible to return an array (a collection of items)
- We can, however, return a *pointer* to an array
- We cannot return a pointer to a *local* array (dangerous; undefined behavior)
- Requires knowledge of *dynamic memory* and `malloc`
- More later this semester; for now: simply declare an array large enough for your purposes
 - Might need two controlling variables: one for max array size and one for number of cells used

Notes

Searching and Sorting an Array

CSCE150A

Introduction
Declaring,
Referencing
Initialization
Function Args
Constants
Returning
Arrays
Searching &
Sorting
Multi-
dimensional
Arrays
Common
Errors
Examples
18 / 30

Two common problems with array processing:

- 1 Searching - Finding the index of a particular element in an array
- 2 Sorting - rearranging array elements in a particular order

Notes

University of Nebraska
Lincoln

CSCE150A

Introduction
Declaring,
Referencing
Initialization
Function Args
Constants
Returning
Arrays
Searching &
Sorting
Multi-
dimensional
Arrays
Common
Errors
Exercises
19 / 30

Searching an Array

1 Assume the target has not been found

2 Start with the initial array element, $a[0]$

3 while the target is not found and there are more array elements do

4 if the current element matches array then

5 set flag true and store the array index

6 end

7 advance to next array element

8 end

9 if flag is set to true then

10 return the array index

11 end

12 return -1 to indicate not found

Algorithm 1: Searching Algorithm

Notes

University of Nebraska
Lincoln

CSCE150A

Introduction
Declaring,
Referencing
Initialization
Function Args
Constants
Returning
Arrays
Searching &
Sorting
Multi-
dimensional
Arrays
Common
Errors
Exercises
20 / 30

Searching an Array

C code

```
1 int search(int array[], int size, int target)
2 {
3     int found = 0, index = -1;
4     while ( !found && (i < size) )
5     {
6         if ( array[i] == target ) {
7             found = 1;
8             index = i;
9         }
10        else {
11            i++;
12        }
13    }
14    if(found)
15        return index;
16    else
17        return -1;
18 }
```

Notes

University of Nebraska
Lincoln

CSCE150A

Introduction
Declaring,
Referencing
Initialization
Function Args
Constants
Returning
Arrays
Searching &
Sorting
Multi-
dimensional
Arrays
Common
Errors
Exercises
21 / 30

Sorting an Array - Selection

1 foreach index value $i = 0, \dots, n - 2$ do

2 Find the index of the smallest element in the
subarray $a[i, \dots, n - 1]$

3 Swap the smallest element with the element
stored at index i

4 end

Algorithm 2: Selection Sort Algorithm

Notes

Sorting an Array - Selection

CSCE150A

Introduction

Declaring,

Referencing

Initialization

Function Args

Constants

Returning

Arrays

Searching &

Sorting

Multi-

dimensional

Arrays

Common

Errors

Exercises

22/30

```
1 void selectionSort(int *a, int size)
2 {
3     int i, j, minimum_index, temp;
4     for(i=0; i<size-2; i++)
5     {
6         minimum_index = i;
7         for(j=i+1; j<size; j++)
8         {
9             if(a[minimum_index] > a[j])
10            {
11                index_of_min = j;
12            }
13        }
14        temp = a[i];
15        a[i] = a[minimum_index];
16        a[minimum_index] = temp;
17    }
18 }
```

Notes

Sorting an Array - Bubble Sort

CSCE150A

Introduction

Declaring,

Referencing

Initialization

Function Args

Constants

Returning

Arrays

Searching &

Sorting

Multi-

dimensional

Arrays

Common

Errors

Exercises

23/30

```
1 while i ≤ n - 1 do
2     while j ≤ n - 1 do
3         if a[j] > a[j + 1] then
4             Swap a[j] and a[j + 1]
5         end
6     end
7 end
```

Algorithm 3: Bubble Sort Algorithm

Notes

Sorting an Array - Bubble Sort

C code

CSCE150A

Introduction

Declaring,

Referencing

Initialization

Function Args

Constants

Returning

Arrays

Searching &

Sorting

Multi-

dimensional

Arrays

Common

Errors

Exercises

24/30

```
1 void bubbleSort(int *a, int size)
2 {
3     int i, j, temp;
4     for (i=0; i<size-1; i++)
5     {
6         for(j=0; j<size-1; j++)
7         {
8             if (a[j] > a[j+1])
9             {
10                temp = a[j];
11                a[j] = a[j+1];
12                a[j+1] = temp;
13            }
14        }
15    }
16 }
```

Notes

Multidimensional Arrays I

CSCE150A

Introduction
Declaring,
Referencing
Initialization
Function Args
Constants
Returning
Arrays
Searching &
Sorting
Multi-
dimensional
Arrays
Common
Errors
Examples
25 / 30

- A multidimensional array is an array with two or more dimensions
- Two-dimensional arrays represent tables of data, matrices, and other two-dimensional objects
- Declare multidimensional arrays similar to regular arrays:
`int myArray[10][20];`
- This declares a 10×20 sized array
- Interpretation: 10 rows, 20 columns

Navigation icons

Notes

Multidimensional Arrays II

CSCE150A

Introduction
Declaring,
Referencing
Initialization
Function Args
Constants
Returning
Arrays
Searching &
Sorting
Multi-
dimensional
Arrays
Common
Errors
Examples
26 / 30

- Each row/column is still indexed $0, \dots, n-1$ and $0, \dots, m-1$
- Last row, last column: `myArray[9][19] = 29;`
- When iterating over a multidimensional array, use nested for loops

```
1 int a[10][10];
2 for(i=0; i<10; i++)
3     for(j=0; j<10; j++)
4         a[i][j] = 1 + i + j;
```

Navigation icons

Notes

Initialization of Multidimensional Arrays

CSCE150A

Introduction
Declaring,
Referencing
Initialization
Function Args
Constants
Returning
Arrays
Searching &
Sorting
Multi-
dimensional
Arrays
Common
Errors
Examples
27 / 30

You can initialize multidimensional arrays when declaring

```
1 char tictactoe[][3] = { { ' ', ' ', ' ', ' ', ' ' },
2                         { ' ', ' ', ' ', ' ', ' ' },
3                         { ' ', ' ', ' ', ' ', ' ' } };
```

This would initialize a 3×3 the array with all blank spaces.

Navigation icons

Notes

Initialization of Multidimensional Arrays

CSCE150A

Introduction
Declaring,
Referencing
Initialization
Function Args
Constants
Returning
Arrays
Searching &
Sorting
Multi-
dimensional
Arrays
Common
Errors
Exercises
26 / 30

- When declaring and initializing, you must still provide all dimensions *except* the outer-most
- The compiler is able to deduce the outer-most dimension at compile time
- Not sophisticated enough to deduce the rest

◀ ▶ ◀ ▶ ◀ ▶ ◀ ▶ ◀ ▶ ◀ ▶ ◀ ▶

Notes

Common Programming Errors

CSCE150A

Introduction
Declaring,
Referencing
Initialization
Function Args
Constants
Returning
Arrays
Searching &
Sorting
Multi-
dimensional
Arrays
Common
Errors
Exercises
29 / 30

- Most common error: out-of-range access error
 - Segmentation fault, Bus error
 - Error may not be caught in some situations: unexpected results
- Use correct syntax when passing arrays as parameters

◀ ▶ ◀ ▶ ◀ ▶ ◀ ▶ ◀ ▶ ◀ ▶ ◀ ▶

Notes

Exercises

CSCE150A

Introduction
Declaring,
Referencing
Initialization
Function Args
Constants
Returning
Arrays
Searching &
Sorting
Multi-
dimensional
Arrays
Common
Errors
Exercises
30 / 30

Write the following functions and write a main driver program to test them.

- void `printArray(int *array, int size)` – prints the elements of an integer array
- void `printMatrix(int **array, int rows, int columns)` – prints the elements of an integer array
- double `average(int *array, int size)` – computes the average of all elements in the array

◀ ▶ ◀ ▶ ◀ ▶ ◀ ▶ ◀ ▶ ◀ ▶ ◀ ▶

Notes
