Chapter 8

8.1 Declaring and Referencing Arrays
8.2 Array Subscripts
8.3 Using For Loops for Sequential Access
8.4 Using Array Elements as Function Arguments
8.5 Array Arguments
8.6 Searching and Sorting an Array
8.7 Multidimensional Arrays
8.9 Common Programming Errors

Introduction

Simple data types use a single memory cell to store a variable
Collections of data should be logically grouped
Example: 75 students in the class; should we declare 75 separate
variables to hold grades?
Grouping related data items together into a single composite data
structure is done using an array

Declaring Arrays I

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

Declaring Arrays II

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

Referencing Array Elements I

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
### Array Initialization

- You can declare multiple arrays along with regular variables:
  ```c
  double cactus[5], needle, pins[7];
  ```
- We can initialize a simple variable when we declare it:
  ```c
  int sum = 0;
  ```
- We can initialize when we declare, we can omit the size
  ```c
  int x[10];
  ```

### Array Declaration & Initialization

- We can declare and initialize an array
  ```c
  int primeNumbersLessThanHundred[] = { 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97 }; 
  ```

### Using for Loops for Sequential Access

- Elements of an array are processed in sequence, starting with element zero.
- Using a counting loop whose loop control variable runs from zero to one less than the array size.
- We can use `scanf` with array elements just like with regular variables
  ```c
  int x[10];
  int i = 0;
  scanf("%d", &x[i]);
  printf("Hey, I read %d
", x[i]);
  ```

### Using Array Elements as Function Arguments

- You can use `scanf` with array elements just like with regular variables
  ```c
  int x[10];
  int i = 0;
  scanf("%d", &x[i]);
  printf("Hey, I read %d
", x[i]);
  ```

### Referencing Array Elements I

- Take care that you do not reference an index outside the array:
  ```c
  double grades[75];
  printf("75th grade is \%f\n", grades[74]);
  ```

### Referencing Array Elements II

- Other elements can be accessed similarly: `x[1], x[2], …`
  ```c
  printf("value of second element=\%d",myArray[1]);
  ```
- For an array of size `n`, we index `0, 1, ..., n - 1`
  ```c
  scanf("input a number: %d",anotherArray[9]);
  ```
- An array must be an integer (no such thing as half an element)
  ```c
  printf("Hey, I read %d
", x[i]);
  ```
- Similarly, each index is also an integer
  ```c
  int primeNumbersLessThanHundred[] = { 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97 }; 
  ```

### Initialization

- You can use `scanf` with array elements just like with regular variables
  ```c
  int x[10];
  int i = 0;
  scanf("%d", &x[i]);
  printf("Hey, I read %d\n", x[i]);
  ```

### Referencing Array Elements II

- Other elements can be accessed similarly: `x[1], x[2], …`
  ```c
  printf("value of second element=\%d",myArray[1]);
  ```
- For an array of size `n`, we index `0, 1, ..., n - 1`
  ```c
  scanf("input a number: %d",anotherArray[9]);
  ```
- An array must be an integer (no such thing as half an element)
  ```c
  printf("Hey, I read %d
", x[i]);
  ```
- Similarly, each index is also an integer
  ```c
  int primeNumbersLessThanHundred[] = { 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97 }; 
  ```

### Initialization

- You can use `scanf` with array elements just like with regular variables
  ```c
  int x[10];
  int i = 0;
  scanf("%d", &x[i]);
  printf("Hey, I read %d\n", x[i]);
  ```

### Referencing Array Elements II

- Other elements can be accessed similarly: `x[1], x[2], …`
  ```c
  printf("value of second element=\%d",myArray[1]);
  ```
- For an array of size `n`, we index `0, 1, ..., n - 1`
  ```c
  scanf("input a number: %d",anotherArray[9]);
  ```
- An array must be an integer (no such thing as half an element)
  ```c
  printf("Hey, I read %d
", x[i]);
  ```
- Similarly, each index is also an integer
  ```c
  int primeNumbersLessThanHundred[] = { 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97 }; 
  ```

### Initialization

- You can use `scanf` with array elements just like with regular variables
  ```c
  int x[10];
  int i = 0;
  scanf("%d", &x[i]);
  printf("Hey, I read %d\n", x[i]);
  ```

### Referencing Array Elements II

- Other elements can be accessed similarly: `x[1], x[2], …`
  ```c
  printf("value of second element=\%d",myArray[1]);
  ```
- For an array of size `n`, we index `0, 1, ..., n - 1`
  ```c
  scanf("input a number: %d",anotherArray[9]);
  ```
- An array must be an integer (no such thing as half an element)
  ```c
  printf("Hey, I read %d
", x[i]);
  ```
- Similarly, each index is also an integer
  ```c
  int primeNumbersLessThanHundred[] = { 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97 }; 
  ```

### Initialization

- You can use `scanf` with array elements just like with regular variables
  ```c
  int x[10];
  int i = 0;
  scanf("%d", &x[i]);
  printf("Hey, I read %d\n", x[i]);
  ```
Arrays as Arguments

- 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:
  ```c
  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:
  ```c
  int sum(int *array, int size);
  ```

Formal Array Parameter

- 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

Returning an Array Result

- 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

Arrays as Input Arguments

- 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

Full Example

```c
#include <stdio.h>

int sum(const int foo[], int size)
{
    int i, summation = 0;
    for(i=0; i<size; i++)
        summation += foo[i];
    return summation;
}

int main(void)
{
    int foo[] = {1,2,3,4,5,6,7,8,9,10}, i;
    int sum(int array[], int size);
    printf("sum of all array elements is %d\n", sum(foo, 10));
    return 0;
}
```

Arrays as Arguments

- 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

Returning an Array Result

- 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

Arrays as Input Arguments

- 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

Full Example

```c
#include <stdio.h>

int sum(const int foo[], int size)
{
    int i, summation = 0;
    for(i=0; i<size; i++)
        summation += foo[i];
    return summation;
}

int main(void)
{
    int foo[] = {1,2,3,4,5,6,7,8,9,10}, i;
    int sum(int array[], int size);
    printf("sum of all array elements is %d\n", sum(foo, 10));
    return 0;
}
```
Searching an Array

Algorithm 1: Searching Algorithm

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

Searching an Array

C code

```c
int search(int array[], int size, int target)
{
// implementation of algorithm
}
```

Sorting an Array - Selection

Algorithm 2: Selection Sort Algorithm

1. foreach index value \( i = 0, \ldots, n - 2 \) do
2. Find the index of the smallest element in the subarray \( a[i], \ldots, a[n-1] \)
3. Swap the smallest element with the element stored at index \( i \)

```c
void selectionSort ( int *a, int size )
{
// implementation of algorithm
}
```

Sorting an Array - Bubble Sort

Algorithm 3: Bubble Sort Algorithm

1. while \( i \leq n - 1 \) do
2. while \( j \leq n - i - 1 \) do
3. if \( a[j] > a[j+1] \) then
4. Swap \( a[j] \) and \( a[j+1] \)
5. end
6. end
7. end

```c
void bubbleSort(int *a, int size)
{
// implementation of algorithm
}
```
Multidimensional Arrays I

- 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

Initialization of Multidimensional Arrays

You can initialize multidimensional arrays when declaring

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

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

Common Programming Errors

- 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

Exercises

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
```

Multidimensional Arrays II

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

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