Arrays

Outline

- Introduction to Arrays (1-D arrays)
- Defining and Declaring Arrays
- Accessing Array Elements
- Examples

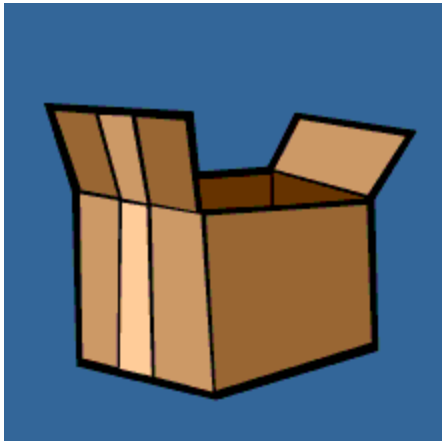
- 2-D Arrays
- Declaring and Initializing 2-D Arrays
- Example
  - Representing matrices and finding the transpose of a matrix

1.1. What is an array?

- An array is a collection of things!

Ordinary Variable

Like a box for storing one value



Array

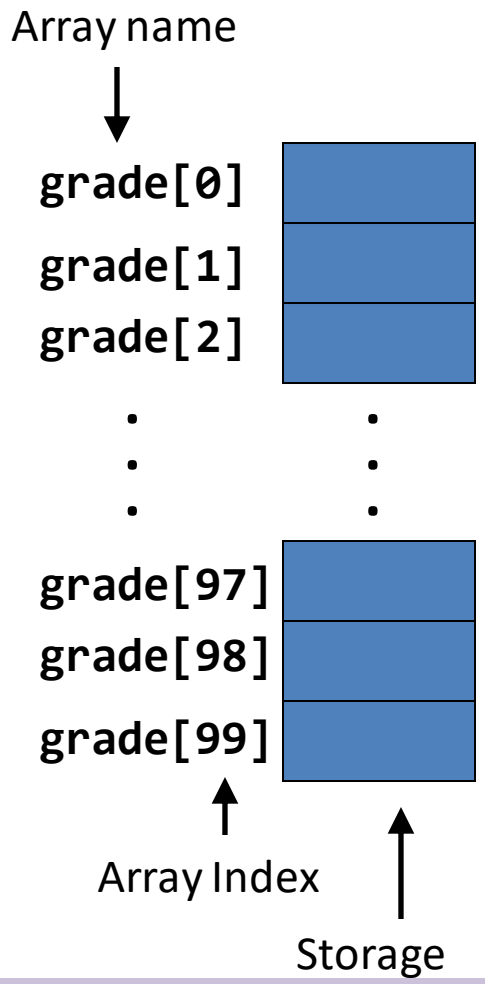
Like a cabinet containing many drawers.

Each drawer stores one value.

We can refer to each drawer as the 1st drawer, the 2nd drawer, the 3rd drawer, etc.

1.2. Characteristics of an Array

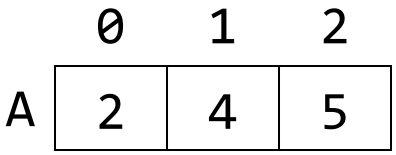
- Stores same type of data
- *Array size* (# of elements in the array) remains unchanged throughout program execution
- The position of an element is indicated by the *index*.
- The indexes to an array of size N ranges from 0 to N-1.
  - Important: The first index must be **0**

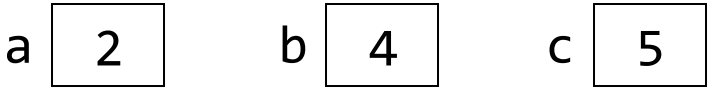


1.3. First look at an Array

```
1  int A[3];
2
3  A[0] = 2;
4  A[1] = A[0] + 2;
5
6  scanf("%d", &A[2]);
7  // Assume input is 5
8  printf("%d", A[2]);
```

```
1  int a, b, c;
2
3  a = 2;
4  b = a + 2;
5
6  scanf("%d", &c);
7  // Assume input is 5
8  printf("%d", c);
```

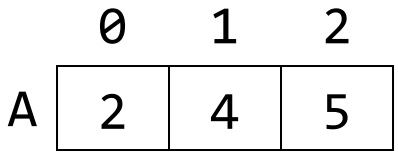




This example illustrates the similarity between an array and ordinary variables

1.3. First Look at an Array

```
1  int A[3];
2
3  A[0] = 2;
4  A[1] = A[0] + 2;
5
6  scanf("%d", &A[2]);
7  // Assume input is 5
8  printf("%d", A[2]);
```



Just like a variable, we need to declare an array before using it.

This statement declares an array to store values of type `int`.

The name of the array is **A**.

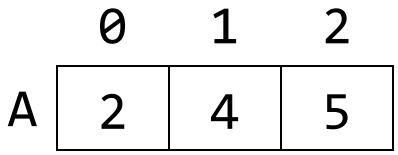
The array *size* is 3.

1.3. First Look at an Array

```
1  int A[3];
2
3  A[0] = 2;
4  A[1] = A[0] + 2;
5
6  scanf("%d", &A[2]);
7  // Assume input is 5
8  printf("%d", A[2]);
```

A single array element is like an ordinary variable.

`A[0]` refers to the 1st element in array **A**.

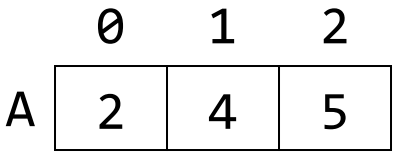


1.3. First Look at an Array

```
1  int A[3];
2
3  A[0] = 2;
4  A[1] = A[0] + 2;
5
6  scanf("%d", &A[2]);
7  // Assume input is 5
8  printf("%d", A[2]);
```

In an array declaration, the number in [...] indicates the size of an array.

In an expression, the number in [...] indicates the index of an array element.



1.4. Syntax: Declaring an Array

```
type arrayName[ arraySize ];
```

- *type*: Data type of the array elements
- *arrayName*: A valid identifier
- *arraySize*: Number of elements in the array

- e.g.,

```
int grade[ 100 ];  // array of 100 integers
double d[ 3284 ];  // array of 3284 doubles
```

- Declaring multiple arrays of the same type in one declaration

```
int arrayA[100], arrayB[27];
```

1.5. Example #1

```
1  int list[4];
2
3  printf("Enter 4 #'s: ");
4  scanf("%d", &list[0]);
5  scanf("%d", &list[1]);
6  scanf("%d", &list[2]);
7  scanf("%d", &list[3]);
8
9  // Print the input values in reverse order
10 printf("You have entered (in reverse): ");
11 printf("%d %d %d %d\n", list[3], list[2], list[1], list[0]);
```

```
Enter 4 #'s: 7 11 45 23↵
You have entered: 23 45 11 7
```

Note: Input values can be separated by any whitespace character.

1.6. Array Bounds

- Indexes to an array of size N range from **`0`** to **`N-1`**.

- An array index that is out of this range can cause a runtime error of "*array index out of bounds*".

```
int c[ 10 ];
c[ -1 ] = 5;          // Index out of bounds
c[ 10 ] = 0;          // Index out of bounds
```

- The consequence of the error is ***unpredictable***.
  - The program may crash.
  - Other variables may get modified unknowingly.

1.7. Syntax: Initializing an Array at Declaration

- Specify a value for each element using an initializer

```
int n[ 5 ] = { 1, 2, 3, 4, 5 };
```

- Not enough values in the initializer ➔ Rightmost elements are set to 0

```
int n[ 5 ] = { 1 };      // n[1], …, n[4] are set to 0
int m[ 5 ] = { 0 };      // All elements are set to 0
```

- Too many values in the initializer ➔ Syntax error

```
int n[ 5 ] = { 1, 2, 3, 4, 5, 6 };   // Error
```

- Array size omitted ➔ Size is determined by the initializer

```
int n[] = { 1, 2, 3, 4, 5 };  // Size of n is 5

int p[];      // Error. Need a size or an initializer
```

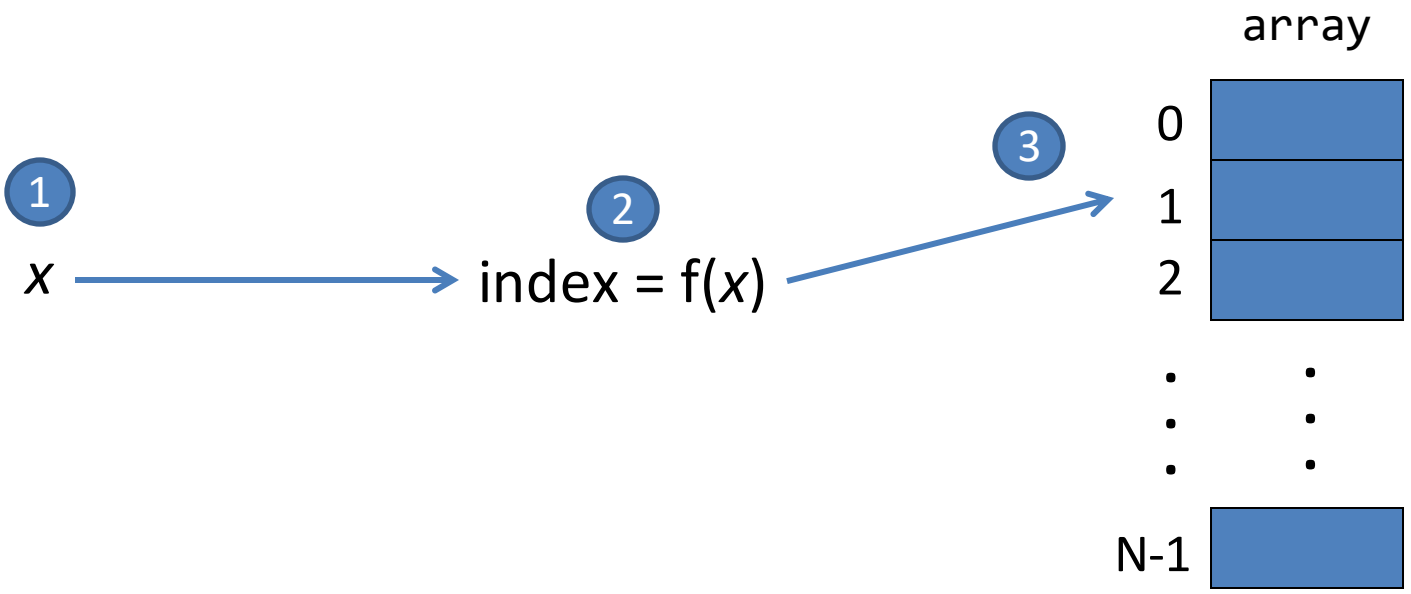
2. More Array Examples

- Array as a lookup table
- Array for counting

2.1. Array as a Lookup Table

General Idea: Map a value to an array index.

1. Given *x*
2. Calculate the *index* to the array element based on the value of *x*
3. Access array[*index*]



2.1. Array as a Lookup Table

- Let's output the number of days of a given month without **`if-else`** statements!

```
1  // monthToDays[month-1] => # of days in "month"
2  int monthToDays[12] = { 31, 28, 31, 30, 31, 30,
3                          31, 31, 30, 31, 30, 31 };
4
5  int month;        // To store user input
6  int days;         // To store # of days in the given month
7
8  scanf("%d", &month);
9
10 // assume we forget the existence of leap year for now
11 days = monthToDays[month - 1];
   printf("The input month has %d days!\n", days);
```

Note: The condition for checking if a year is a leap year is a bit long and is omitted in this example.

2.2. Array for Counting

Let's count the number of star ratings of a movie based on 4 viewers!

```
1  int starCount[5] = { 0 };
2
3  int rating; // To store user input, assuming valid
4  printf("Enter 4 viewers' ratings (1 to 5 stars): ");
5
6  // read 4 numbers from the viewers, one-by-one, and tally
7  scanf("%d", &rating);
8  starCount[rating-1]++;
9  scanf("%d", &rating);
10 starCount[rating-1]++;
11 scanf("%d", &rating);
12 starCount[rating-1]++;
13 scanf("%d", &rating);
14 starCount[rating-1]++;
15
16 printf("No. of viewers giving 1 to 5 stars: %d %d %d %d %d\n",
17   starCount[0],starCount[1],starCount[2],starCount[3],starCount[4]);
18
```

3.1. 2-D Array

- It's like a table consisting of rows and columns
- Usually rectangular in shape
- Stores values of the same type
- It takes two indexes to identify each element

	Column 0	Column 1	Column 2	Column 3
Row 0				
Row 1				
Row 2				

3.2. Declaring a 2-D Array

```
type arrayName[ rowSize ][ colSize ];
```

- *type*: Data type of the array elements
- *arrayName*: A valid identifier
- *rowSize*: Number of rows (size of the 1st dimension)
- *colSize*: Number of columns (size of the 2nd dimension)

- e.g.,

```
int a[3][4];  // 3 rows, 4 columns
```

	Column 0	Column 1	Column 2	Column 3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

1st index to row

2nd index to column

3.3. 2-D Array – Declaration and Initialization

- Declaration

```
int   x[3][4];        // 3 rows, 4 columns
short y[2][10];       // 2 rows, 10 columns
```

- Declaration with initialization

```
// As an array of arrays
int a[2][3] = {{1, 2, 3}, {4, 5, 6}};

// Not enough initializers, the rest got zeros
int b[2][3] = {{1}, {4, 5}};

// Values assigned by order
int c[2][3] = {1, 2, 3, 4 ,5};
```

a[0]	1	2	3
a[1]	4	5	6

b[0]	1	0	0
b[1]	4	5	0

c[0]	1	2	3
c[1]	4	5	0

```
1   // A and B represent two 3x3 matrixes
2   int A[3][3] = { { 1, 2, 3 }, { 0, -1, 2 }, { 0, 0, 1 } };
3   int B[3][3];   // To store the transpose of matrix A
4   int i, j;
5
6   // Store transpose of A in B
7   B[0][0] = A[0][0];
8   B[0][1] = A[1][0];
9   B[0][2] = A[2][0];
10
11  B[1][0] = A[0][1];
12  B[1][1] = A[1][1];
13  B[1][2] = A[2][1];
14
15  B[2][0] = A[0][2];
16  B[2][1] = A[1][2];
17  B[2][2] = A[2][2];
18
19  // See the next page
20
```

Transpose of a matrix refers to the change of rows into columns

Example 3.3. Using 2-D arrays to represent matrixes

```
21
22  // Print matrix A and B
23  printf("%4d%4d%4d\n", A[0][0], A[0][1], A[0][2]);
24  printf("%4d%4d%4d\n", A[1][0], A[1][1], A[1][2]);
25  printf("%4d%4d%4d\n", A[2][0], A[2][1], A[2][2]);
26  printf("\n");
27
28  printf("%4d%4d%4d\n", B[0][0], B[0][1], B[0][2]);
29  printf("%4d%4d%4d\n", B[1][0], B[1][1], B[1][2]);
30  printf("%4d%4d%4d\n", B[2][0], B[2][1], B[2][2]);
31  printf("\n");
32
33
34
35
36
37
38
39
40
```

What is **%4d**? It means you will use a minimum of 4 character spaces to print out the integer; this is useful when you want your output to be tidy.

Try change it to **%10d** and see what would happen.

```
   1   2   3
   0  -1   2
   0   0   1

   1   0   0
   2  -1   0
   3   2   1
```

Example 3.3. Using 2-D arrays to represent matrixes (cont.)

3.4. Applications of 2-D Arrays

- Digital images (2-D array of pixels)
- Matrix in mathematics
- Assignment scores of students
  - Each row represents a student
  - Each column represents the student's scores from different components
- Games (Chess, Minesweeper, etc.)
- Spreadsheet
- etc.

Summary

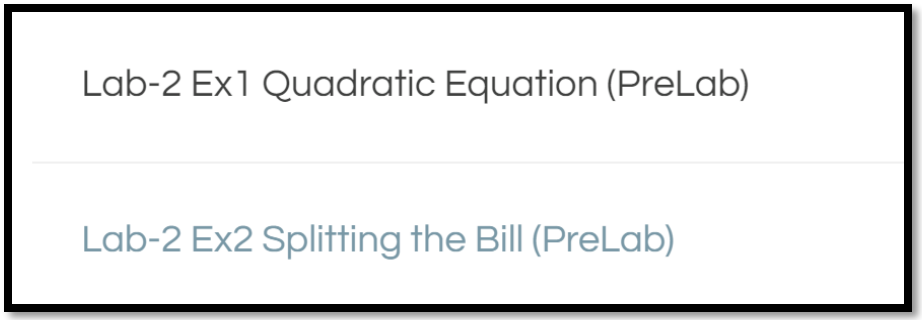
- Understanding the characteristics of 1-D and 2-D arrays
- Knowing how to declare and initialize 1-D and 2-D arrays

Reading Assignment

- C: How to Program, 8th ed, Deitel and Deitel
- Chapter 6 C Arrays
  - Sections 6.1 – 6.4: Basics and examples
  - Sections 6.11: Multidimensional Arrays (2D Arrays)

Reminder: PreLabs are Ready!

- Every Mon afternoon we will release the **PreLabs**
  - Meant to help you prepare for the lab
  - Due **Wed 9:30am** – Please try it after the lecture and submit before Wed!
  - Don't worry – it's super easy (takes < 30 min) and it's very easy marks to get! Don't forget!



*This screenshot is for illustration only; not actual exercises

PreLabs are marked "(PreLab)" on repl.it