Arrays

2024 Winter APS 105: Computer Fundamentals
Jon Eyolfson

Lecture 13
1.1.0

We Have For Loops for Repetition

However, loops don’t help us when we have multiple variables

What if we want to calculate the average of a bunch of grades?

Calculating the Average is Error-Prone

```c
int main(void) {
    int grade1 = 75;
    int grade2 = 83;
    int grade3 = 99;
    int grade4 = 64;
    int grade5 = 72;
    int average = (grade1 + grade2 + grade3 + grade4 + grade5) / 5;
    printf("Average: %d\n", average);
    return 0;
}
```

Every time we add a new variable, we have to:

1) Add it to the average calculation
2) Change the divisor

Arrays are for Groups of Related Values

We can declare a number of `int` all at once

The syntax for declaring arrays is:

```
  <type> <name>[<array_size>];
```

Where you replace:

- `<type>` by the type for each value (or element) of the array
- `<name>` by a name you want to give the array (group of values)
- `<array_size>` by the number of values you want to create

You Can Access Each Value Using Square Brackets

After you've declared an array, you can access a single element by writing:

```
<array_name>[<index>]
```

Where you replace:

`<array_name>` by the name you've given to an array

`<index>` by the element you'd like to select (as an integer)

You Can Access Each Value Using Square Brackets

After you’ve declared an array, you can access a single element by writing:

`<array_name>[<index>]`

Where you replace:

- `<array_name>` by the name you’ve given to an array
- `<index>` by the element you’d like to select (as an integer)

We call it an index because counting starts at 0

- C in arrays are zero-indexed (and most programming languages)

`a[0]` is the first value (or element), `a[1]` is the second value, etc.

Let's Use an Array Instead

```
int main(void) {
    int grades[5];
    grades[0] = 75;
    grades[1] = 83;
    grades[2] = 99;
    grades[3] = 64;
    grades[4] = 72;
    int average = (grades[0] + grades[1] + grades[2] + grades[3] + grades[4])
                  / 5;
    printf("Average: %d\n", average);
}
```

Can we re-write the average calculation without writing all the values?

We Can Write a Loop Over Each Element of the Array

```c
#include <stdio.h>
#include <stdlib.h>

int main(void) {
    int grades[5];
    grades[0] = 75;
    grades[1] = 83;
    grades[2] = 99;
    grades[3] = 64;
    grades[4] = 72;
    int sum = 0;
    for (int i = 0; i < 5; ++i) {
        sum += grades[i];
    }
    int average = sum / 5;
    printf("Average: %d\n", average);
    return EXIT_SUCCESS;
}
```

We Have a Magic Number We Want to Get Rid Of

In the previous slide, we use 5 in multiple places

If we increase the length of the array to 6, we have to remember to:

- Change the loop bounds to 6
- Change the average calculation to divide by 6

We Have a Magic Number We Want to Get Rid Of

In the previous slide, we use `5` in multiple places

If we increase the length of the array to `6`, we have to remember to:
- Change the loop bounds to `6`
- Change the average calculation to divide by `6`

We can use `#define` to get the C preprocessor to copy and paste for us

The C Preprocessor Does a Search and Replace

We can write:

```
#define <search> <replacement>
```

Where you replace:

`<search>` by the string to replace (use ALL CAPS, separate words with _)
`<replacement>` by what you’d like to be written instead

The C Preprocessor Does a Search and Replace

We can write:

`#define <search> <replacement>`

Where you replace:

`<search>` by the string to replace (use ALL CAPS, separate words with _)

`<replacement>` by what you'd like to be written instead

You can write defines like a function (they can have arguments),
but you won't have to write any yourself in this course

Let’s Remove the Magic Values

```
#include <stdio.h>

#define GRADES_LENGTH 5

int main(void) {
    int grades[GRADES_LENGTH];
    grades[0] = 75;
    grades[1] = 83;
    grades[2] = 99;
    grades[3] = 64;
    grades[4] = 72;
    int sum = 0;
    for (int i = 0; i < GRADES_LENGTH; ++i) {
        sum += grades[i];
    }
    int average = sum / GRADES_LENGTH;
    printf("Average: %d\n", average);
    return 0;
}
```

We Can Assign Values to Elements in the Declaration

The syntax for declaring arrays and assigning values is:

```
<type> <name>[<array_size>] = {<comma_separated_values>};
```

Where you replace:

- `<type>` `<name>` `<array_size>` with the same rules as before
- `<comma_separated_values>` with the values you'd like to assign (in order)
    - The values must match the type of the array

We Can Remove the Multiple Assignment Statements

```c
#include <stdio.h>
#include <stdlib.h>

#define GRADES_LENGTH 5

int main(void) {
    int grades[GRADES_LENGTH] = {75, 83, 99, 64, 72};
    int sum = 0;
    for (int i = 0; i < GRADES_LENGTH; ++i) {
        sum += grades[i];
    }
    int average = sum / GRADES_LENGTH;
    printf("Average: %d\n", average);
    return EXIT_SUCCESS;
}
```

Beware: You Cannot Reassign an Array Itself

```
#include <stdio.h>
#include <stdlib.h>

#define GRADES_LENGTH 5

int main(void) {
    int grades[GRADES_LENGTH] = {75, 83, 99, 64, 72};
    grades = 0;
    return EXIT_SUCCESS;
}
```

Results in an error like:

```
  error: array type 'int[5]' is not assignable
```

When We Assign Values, We Can Let C Figure out the Length

We can omit the `<array_size>` and use the following:

```
<type> <name>[] = {<comma_separated_values>};
```

C will create an array with a length equal to the number of values

When We Assign Values, We Can Let C Figure out the Length

We can omit the `<array_size>` and use the following:

```
  <type> <name>[] = {<comma_separated_values>};
```

C will create an array with a length equal to the number of values

We can write:

```
  int grades[] = {75, 83, 99, 64, 72};
```

An Array is Large Enough (in Memory) to Hold All the Values

Assuming we have:

```
  int grades[] = {75, 83, 99, 64, 72};
```

The `grades` array contains 5 `int` values

- An `int` is 4 bytes, therefore the total size is 20 bytes

An Array is Large Enough (in Memory) to Hold All the Values

Assuming we have:

```
  int grades[] = {75, 83, 99, 64, 72};
```

The `grades` array contains 5 `int` values

  An `int` is 4 bytes, therefore the total size is 20 bytes

We can check the size (in bytes) of the array using the `sizeof` operator

```
  sizeof(grades); → 20
```

Beware: Programmer's Often Use the Terms Length and Size Interchangeably

Usually we don't need to use the size (in bytes) of an array

So we might say the size of the array in the previous example is `5`
(even though that's different from the `sizeof` operator)

We Can Use a `#define` to Calculate the Length of the Array

We call a `#define` that takes arguments a macro

We can use the following:

```
#define ARRAY_LENGTH(arr) (sizeof(arr) / sizeof((arr)[0]))
```

(you don’t have to understand this)

In the previous example:

```
ARRAY_LENGTH(grades) → 5
```

Now We Can Remove All Magic Values

```c
#include <stdio.h>
#include <stdlib.h>

#define ARRAY_LENGTH(arr) (sizeof(arr) / sizeof((arr)[0]))

int main(void) {
    int grades[] = {75, 83, 99, 64, 72};
    int gradesLength = ARRAY_LENGTH(grades);
    int sum = 0;
    for (int i = 0; i < gradesLength; ++i) {
        sum += grades[i];
    }
    int average = sum / gradesLength;
    printf("Average: %d\n", average);
    return EXIT_SUCCESS;
}
```

We Need to Ensure All Array Indices are Valid

Again, assuming we have:

```
  int grades[] = {75, 83, 99, 64, 72};
```

We can only access:

```
  grades[0] grades[1] grades[2] grades[3] grades[4]
```

We Need to Ensure All Array Indices are Valid

Again, assuming we have:

```
  int grades[] = {75, 83, 99, 64, 72};
```

We can only access:

```
  grades[0] grades[1] grades[2] grades[3] grades[4]
```

In general if we have an array with a length, `arrayLength`, we can only access:

  `array[0]` to `array[arrayLength - 1]`

Very Odd Things May Happen If You Use Invalid Indices

```c
#include <stdio.h>

#define ARRAY_LENGTH(arr) (sizeof(arr) / sizeof((arr)[0]))

int main(void) {
    int x = 1;
    int grades[] = {75, 83, 99, 64, 72};
    int gradesLength = ARRAY_LENGTH(grades);
    int sum = 0;
    for (int i = 0; i < gradesLength; ++i) {
        sum += grades[i];
    }
    int average = sum / gradesLength;
    printf("Average: %d\n", average);
    grades[6] = 2; /* We should not do this! */
    printf("x: %d\n", x);
    return 0;
}
```

One Odd Thing that May Happen in the Previous Slide

On some computers, `grades[6]` will re-assign `x`

We'll see `x: 2` printed, which is very confusing

One Odd Thing that May Happen in the Previous Slide

On some computers, `grades[6]` will re-assign `x`

We'll see `x: 2` printed, which is very confusing

Always make sure your array indices are in bounds!