Array Lesson 2 Outline

1. Array Lesson 2 Outline
2. Arrays + Loops = Amazing!
3. for Loops for Tasks on Arrays #1
4. for Loops for Tasks on Arrays #2
5. Another for/Array Example #1
6. Another for/Array Example #2
7. Another for/Array Example #3
8. Don’t Need to Use Entire Declared Length
9. Reading Array Values Using `for` Loop #1
10. Reading Array Values Using `for` Loop #2
11. `for` Loop: Like Many Statements #1
12. `for` Loop: Like Many Statements #2
13. `for` Loop: Like Many Statements #3
14. Reading Array on One Line of Input #1
15. Reading Array on One Line of Input #2
16. Reading Array on One Line of Input #3
17. Aside: Why Named Constants Are Good
18. Named Constants as Loop Bounds #1
19. Named Constants as Loop Bounds #2
20. Computing with Arrays #1
21. Computing with Arrays #2
22. Computing with Arrays #3
23. Computing with Arrays #4
24. Computing with Arrays #5

Arrays + Loops = Amazing!

Arrays seem kind of dull and listless,
until you add a key ingredient: loops!

```
for (count = 0; count < number_of_elements; count++) {
    a[count] = 2 * count;
} /* for count */
```

`for` Loops for Tasks on Arrays #1

```c
#include <stdio.h>

int main ()
{ /* main */
    const int number_of_elements   = 5;
    const int program_success_code = 0;
    int a[number_of_elements];
    int count;

    for (count = 0; count < number_of_elements; count++) {
        a[count] = 2 * count;
    } /* for count */
    for (count = 0; count < number_of_elements; count++) {
        printf("a[%2d] = %2d\n", count, a[count]);
    } /* for count */
    return program_success_code;
} /* main */
```

for Loops for Tasks on Arrays #2

```
% gcc -o array_for_mult array_for_mult.c
% array_for_mult
a[ 0] = 0
a[ 1] = 2
a[ 2] = 4
a[ 3] = 6
a[ 4] = 8
```

Another `for`/Array Example #1

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

int main ()
{ /* main */
    const int minimum_number_of_elements =  1;
    const int maximum_number_of_elements = 15;
    const int program_failure_code       = -1;
    const int program_success_code       =  0;
    int a[maximum_number_of_elements];
    int number_of_elements;
    int count;

    printf("How long will the array be (%d to %d)?\n",
        minimum_number_of_elements,
        maximum_number_of_elements);
    scanf("%d", &number_of_elements);
    if ((number_of_elements < minimum_number_of_elements) ||
        (number_of_elements > maximum_number_of_elements)) {
        printf("That’s not a valid array length!\n");
        exit(program_failure_code);
    } /* if ((number_of_elements < ...) || ...) */
```

Another for/Array Example #2

```
    for (count = 0; count < number_of_elements; count++) {
        a[count] = 2 * count;
    } /* for count */
    for (count = 0; count < number_of_elements; count++) {
        printf("a[%2d] = %2d\n", count, a[count]);
    } /* for count */
    return program_success_code;
} /* main */
```

Another for/Array Example #3

```
% gcc -o array_for_mult_read array_for_mult_read.c
% array_for_mult_read
How long will the array be (1 to 15)?
0
That’s not a valid array length!
% array_for_mult_read
How long will the array be (1 to 15)?
16
That’s not a valid array length!
% array_for_mult_read
How long will the array be (1 to 15)?
5
a[ 0] = 0
a[ 1] = 2
a[ 2] = 4
a[ 3] = 6
a[ 4] = 8
```

Don’t Need to Use Entire Declared Length

```
#include <stdio.h>

int main ()
{ /* main */
    const int minimum_number_of_elements =  1;
    const int maximum_number_of_elements = 15;
    const int program_failure_code       = -1;
    const int program_success_code       =  0;
    int a[maximum_number_of_elements];
    ...
} /* main */
...
% array_for_mult_read
How long will the array be (1 to 15)?
5
a[ 0] = 0
a[ 1] = 2
a[ 2] = 4
a[ 3] = 6
a[ 4] = 8
```

Notice that we can **declare** an array to be **longer** than the portion of the array that we actually use, because RAM is cheap.

Reading Array Values Using `for` Loop #1

```c
#include <stdio.h>

int main ()
{ /* main */
    const int z_length                 = 6;
    const int program_success_code = 0;
    float z[z_length], z_squared[z_length];
    int   index;

    for (index = 0; index < z_length; index++) {
        printf("Input z[%d]:\n", index);
        scanf("%f", &z[index]);
    } /* for index */
    for (index = 0; index < z_length; index++) {
        z_squared[index] = z[index] * z[index];
    } /* for index */
    for (index = 0; index < z_length; index++) {
        printf("%19.7f^2 = %19.7f\n",
            z[index], z_squared[index]);
    } /* for index */
    return program_success_code;
} /* main */
```

"Use at least 19 spaces total, 7 of which are to the right of the decimal point."

Reading Array Values Using `for` Loop #2

```
% gcc -o array_for_read_square array_for_read_square.c
% array_for_read_square
Input z[0]:
5
Input z[1]:
1.1
Input z[2]:
-33.33333
Input z[3]:
1.5e+05
Input z[4]:
0.0033333
Input z[5]:
1.5e-05
          5.0000000^2 =          25.0000000
          1.1000000^2 =           1.2100000
        -33.3333282^2 =        1111.1107178
     150000.0000000^2 = 22499999744.0000000
          0.0033333^2 =           0.0000111
          0.0000150^2 =           0.0000000
```

for Loop: Like Many Statements #1

```
#include <stdio.h>

int main ()
{ /* main */
    const int z_length             = 6;
    const int program_success_code = 0;
    float z[z_length], z_squared[z_length];

    printf("Input z[%d]:\n", 0);
    scanf("%f", &z[0]);
    printf("Input z[%d]:\n", 1);
    scanf("%f", &z[1]);
    printf("Input z[%d]:\n", 2);
    scanf("%f", &z[2]);
    printf("Input z[%d]:\n", 3);
    scanf("%f", &z[3]);
    printf("Input z[%d]:\n", 4);
    scanf("%f", &z[4]);
    printf("Input z[%d]:\n", 5);
    scanf("%f", &z[5]);
```

`for` Loop: Like Many Statements #2

```
    z_squared[0] = z[0] * z[0];
    z_squared[1] = z[1] * z[1];
    z_squared[2] = z[2] * z[2];
    z_squared[3] = z[3] * z[3];
    z_squared[4] = z[4] * z[4];
    z_squared[5] = z[5] * z[5];
    printf("%19.7f^2 = %19.7f\n",
        z[0], z_squared[0]);
    printf("%19.7f^2 = %19.7f\n",
        z[1], z_squared[1]);
    printf("%19.7f^2 = %19.7f\n",
        z[2], z_squared[2]);
    printf("%19.7f^2 = %19.7f\n",
        z[3], z_squared[3]);
    printf("%19.7f^2 = %19.7f\n",
        z[4], z_squared[4]);
    printf("%19.7f^2 = %19.7f\n",
        z[5], z_squared[5]);
    return program_success_code;
} /* main */
```

for Loop: Like Many Statements #3

```
% gcc -o array_no_for_read_square \
         array_no_for_read_square.c
% array_no_for_read_square
Input z[0]:
5
Input z[1]:
1.1
Input z[2]:
-33.33333
Input z[3]:
1.5e+05
Input z[4]:
0.0033333
Input z[5]:
1.5e-05
          5.0000000^2 =          25.0000000
          1.1000000^2 =           1.2100000
        -33.3333282^2 =        1111.1107178
     150000.0000000^2 = 22499999744.0000000
          0.0033333^2 =           0.0000111
          0.0000150^2 =           0.0000000
```

Reading Array on One Line of Input #1

Instead of having to explicitly prompt for each array element, you can have a single prompt, and then the user can input all of the array elements’ values in a single line of input text.

Reading Array on One Line of Input #2

```
#include <stdio.h>

int main ()
{ /* main */
    const int z_length             = 6;
    const int program_success_code = 0;
    float z[z_length], z_squared[z_length];
    int   index;

    printf("Input all %d values of z:\n", z_length);
    for (index = 0; index < 6; index++) {
        scanf("%f", &z[index]);
    } /* for index */
    for (index = 0; index < 6; index++) {
        z_squared[index] = z[index] * z[index];
    } /* for index */
    for (index = 0; index < 6; index++) {
        printf("%19.7f^2 = %19.7f\n",
            z[index], z_squared[index]);
    } /* for index */
    return program_success_code;
} /* main */
```

Reading Array on One Line of Input #3

```
% gcc -o array_for_read_1line_square \
         array_for_read_1line_square.c
% array_for_read_1line_square
Input all 6 values of z:
5  1.1  -33.33333  1.5e+05  0.0033333  1.5e-05
          5.0000000^2 =          25.0000000
          1.1000000^2 =           1.2100000
        -33.3333282^2 =        1111.1107178
     150000.0000000^2 = 22499999744.0000000
          0.0033333^2 =           0.0000111
          0.0000150^2 =           0.0000000
```

Aside: Why Named Constants Are Good

Consider the previous program.

What if we decide that we want to change the array length?

Then we'd have to go in and change **every** `for` statement in the program.

That may not seem like much work in the previous program, but it can be a lot of work with large programs.

For example, the Weather Research & Forecasting (WRF) code, a numerical weather prediction program in use worldwide, is a Fortran 90 program that is **over 4.5 million lines long**, with **over 30,000 loops**.

Changing the loop bounds on such a program would take a **huge** amount of work.

Named Constants as Loop Bounds #1

```
#include <stdio.h>

int main ()
{ /* main */
    const int z_length             = 6;
    const int lower_bound          = 0;
    const int program_success_code = 0;
    float z[z_length], z_squared[z_length];
    int   index;

    for (index = lower_bound; index < z_length; index++) {
        printf("Input z[%d]:\n", index);
        scanf("%f", &z[index]);
    } /* for index */
    for (index = lower_bound; index < z_length; index++) {
        z_squared[index] = z[index] * z[index];
    } /* for index */
    for (index = lower_bound; index < z_length; index++) {
        printf("%19.7f^2 = %19.7f\n",
            z[index], z_squared[index]);
    } /* for index */
    return program_success_code;
} /* main */
```

Named Constants as Loop Bounds #2

```
% gcc -o array_for_read_named \
         array_for_read_named.c
% array_for_read_named
Input z[0]:
5
Input z[1]:
1.1
Input z[2]:
-33.33333
Input z[3]:
1.5e+05
Input z[4]:
0.0033333
Input z[5]:
1.5e-05
          5.0000000^2 =         25.0000000
          1.1000000^2 =          1.2100000
        -33.3333282^2 =       1111.1107178
     150000.0000000^2 = 22499999744.0000000
          0.0033333^2 =          0.0000111
          0.0000150^2 =          0.0000000
```

Computing with Arrays #1

```
#include <stdio.h>
int main ()
{ /* main */
    const float initial_sum          =  0.0;
    const int   length               = 10;
    const int   lower_bound          =  0;
    const int   upper_bound          = length - 1;
    const int   program_success_code =  0;
    int a[length];
    int sum;
    int index;
    printf("Input values #%d to #%d:\n",
        lower_bound, upper_bound);
    for (index = lower_bound; index < length; index++) {
        scanf("%d", &a[index]);
    } /* for index */
    sum = initial_sum;
    for (index = lower_bound; index < length; index++) {
        sum = sum + a[index];
    } /* for index */
    printf("The sum of those values is %d.\n", sum);
    return program_success_code;
} /* main */
```

Computing with Arrays #2

```
% gcc -o array_sum array_sum.c
% array_sum
Input values #0 to #9:
1  4  9  16  25  36  49  64  81  100
The sum of those values is 385.
```

Computing with Arrays #3

```
#include <stdio.h>

int main ()
{ /* main */
    const int length               = 10;
    const int lower_bound          =  0;
    const int upper_bound          = length - 1;
    const int program_success_code =  0;
    int a[length], b[length], c[length];
    int index;

    printf("Input a values #%d to #%d:\n",
        lower_bound, upper_bound);
    for (index = lower_bound; index < length; index++) {
        scanf("%d", &a[index]);
    } /* for index */
    printf("Input b values #%d to #%d:\n",
        lower_bound, upper_bound);
    for (index = lower_bound; index < length; index++) {
        scanf("%d", &b[index]);
    } /* for index */
```

Computing with Arrays #4

```
    for (index = lower_bound; index < length; index++) {
        c[index] = a[index] + b[index];
    } /* for index */
    printf("The pairwise sums of the ");
    printf("%d array elements are:\n", length);
    for (index = lower_bound; index < length; index++) {
        printf("%d ", c[index]);
    } /* for index */
    printf("\n");
    return program_success_code;
} /* main */
```

Computing with Arrays #5

```
% gcc -o array_add_pairwise array_add_pairwise.c
% array_add_pairwise
Input a values #0 to #9:
1  8  27  64  125  216  343  512  729  1000
Input b values #0 to #9:
1  4    9  16   25   36   49   64   81   100
The pairwise sums of the 10 array elements are:
2 12 36 80 150 252 392 576 810 1100
```