CSC111

A sentinel…

- A special value that marks the end of a sequence of values
- When a program reads the sentinel value in a loop, it knows it has reached the end of the sequence, so the loop terminates.

Example:

- a user repeatedly enters positive numbers and enters -1 to stop
- The sentinel here is the -1

Output if -1 is entered when first scanf executes:

```
enter an int, -1 to stop
done
```

```
int val, scanned;
printf(“enter an int, -1 to stop\n”);
scanned = scanf("%d", &val);
while (scanned == 1 && val != -1){
     printf(“%d\n”, val);
     printf(“enter an int, -1 to stop\n”);
     scanned = scanf("%d", &val);
}
printf(“done”);
```

Output if 10 is entered when first line executes:

```
enter an int, -1 to stop
10
enter an int, -1 to stop
...
done
```

Arrays

Declaring individual values is not scalable…

```
int main(){
        int a = 9;
        int b = 8;
        int c = 7;
        int d = 9;
        …
        int z = 4;
}
```

- Declaring primitive type variables

```
int x;
double d;
```

- Declaring + initializing primitive type variables

```
int x = 5;
double d = 5.6;
```

- Declaring an array of a specified size
  - Allocates space for the array of given size, with initial garbage values

```
int my_ints[10];
double my_doubles[5];
```

- Declaring + initializing an array of a specified size
  - Allocates space for the array of given size, with initial values given {…}

```
int my_ints[3] = {1, 2, 3};
double my_doubles[2] = {5.6, 3.2};
```

  - Allocates space for the array of given size, with initial values of 0

```
int more_ints[3] = {0};
```

Accessing values in an array

```
int my_ints[3] = {8,19,10};

printf(“value at index 0: %d\n”, my_ints[0]);
printf(“value at index 1: %d\n”, my_ints[1]);
printf(“value at index 2: %d\n”, my_ints[2]);
int sum = my_ints[0] + my_ints[2];
```

Assigning values to indices of an array

```
int my_ints[3] = {8,19,10};

int sum = my_ints[0] + my_ints[1];

my_int[2] = sum;

my_int[0] = my_int[0] + 1;

my_int[1]++;
```

Using a **for-loop** to traverse an array

```
int my_ints[3] = {8,19,10};
//instead of duplicating code:
printf("value at index 0: %d\n", my_ints[0]);
printf("value at index 1: %d\n", my_ints[1]);
printf("value at index 2: %d\n", my_ints[2]);

//put the repeating code within the body of a loop:
int i;
int length = 3;
for(i=0; i<length; i++){
     printf("value at index %d: %d\n", i, my_ints[i]);
}
```

Array Demo

```c
#include <stdio.h>
void print_array(int numbers[], int length);

int main ( ) {
        int array1[2] = {11,21};
        int array2[4] = {4, 5, 6, 7};

        print_array(array1, 2);
        print_array(array2, 4);

        return 0;
}

/* Purpose: print the values in numbers that contains length elements
 * Parameters:  int numbers[] – array of integers
 *              int length – number of elements in numbers
 */
void print_array (int numbers[], int length) {
        int i;
        for (i=0; i<length; i++) {
                printf("%d\n", numbers[i]);
        }
}
```

RECALL: to call a function, you call it by name and pass it the values it expects

Array Summary

- Declare with fixed size
  - int arr[5];
  - int arr2 ={0,3,4,6,7}
- Unless the values are replaced initial values are garbage values
- Access individual elements by location index
  - Remember it is possible to go beyond the end of the array
- Pass array as parameter – pass array and then its length

What is the output of the following code?
What would you describe the function of this code as?

```
#include <stdio.h>
#define SIZE 4

int main( void ) {
    double data[SIZE] = { 5.1, 23.7, 2.0, -4.3 };
    int i;
    double x = 0.0;

    for (i=SIZE-1; i>=0; i--) {
        x += data[i];
    }

    printf( "%f\n", x / SIZE );

    return 0;
}
```

What is the output of this code?

```
#include <stdio.h>
#define SIZE 6
int main( void ) {
    int data[SIZE] = { 5, 23, 2, -4, 7, 12 };
    int index_l = 0;
    int index_r = SIZE - 1;
    int temp;
    while( index_l < index_r ) {
        temp = data[ index_l];
        data[ index_l] = data[ index_r ];
        data[ index_r] = temp;
        index_l++;
        index_r--;
    }
    for(int i=0; i<SIZE; i++){
        printf("%d ", data[i]);
    }
    return 0;
}
```

Complete the function count_above as described:

```
#include <stdio.h>

int count_above (int data[], int sz, int threshold);

int main( void ) {
   int data_empty[0] = {};
   int data_10[10] = { -5, -7, 3, 1, 0, 23, -14, 35, 12, 16 };
   printf("Returned: %d\n", count_above(data_10, 10, 3);
   printf("Returned: %d\n", count_above(data_empty, 0, 5));
   return 0;
}

/*
 * Purpose:  counts and returns the number of values in data with sz elements
 *    that are above threshold
 * Params: int data[]
 *        int sz – number of elements in data
 *        int threshold – values should be above threshold if counted
 * Returns: int – the count
 */

int count_above (int data[], int sz, int threshold) {

}
```

Complete the function count_above as described:

```
#include <stdio.h>

int get_max(int data[], int sz);

int main( void ) {

   // add test calls to get_max using the following data
   int data1[1] = { 5 };
   int data7[7] = { 5, 3, 12, 34, 2, -17, 6 };
   return 0;
}

/*
 * Purpose:  finds and returns the largest value found in data with sz elements
 * Params: int data[]
 *        int sz – number of elements in data, >0
 * Returns: int – the largest value in data
 */
int get_max (int data[], int sz) {

}
```

Convert code to return index of minimum value?

```
#include <stdio.h>

int get_max(int data[], int sz);

int main( void ) {

    // add test calls to get_max using the following data
    int data1[1] = { 5 };
    int data7[7] = { 5, 3, 12, 34, 2, -17, 6 };
    return 0;
}

/*
 * Purpose:  finds and returns the largest value found in data with sz elements
 * Params: int data[]
 *         int sz – number of elements in data, >0
 * Returns: int – the largest value in data
 */
int get_max (int data[], int sz) {

}
```