Arrays

1. What is the purpose of the following program and what would you rename the variable `x` to? Another way of stating this question is: "What statement would you provide in the opening documentation that describes what this program will do when it runs?"

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

// Calculates and prints the average of the numbers in data
int main( void ) {
    double data[SIZE] = { 5.1, 23.7, 2.0, -4.3 };
    int i;
    double sum = 0.0;

    for (i=0; i<SIZE; i++) {
        sum += data[i];
    }

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

    return 0;
}
```

2. How would you complete the sentence printed by the `printf` statement at the end of the following program to accurately describe how the array has been transformed? Note: you may wish to start by drawing a trace table and trace the value of key variables and expressions until you can figure out what the code does.

```
#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--;
    }

    printf( "Array has been reversed\n" );

    return 0;
}
```

3. Complete the function definition and add test calls for `count_above` so that it behaves as described in the documentation. Note that we must be able to initialize the array with a different set of values (could be less or more values) and your program must still work.

```
#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 };

    int num_above;
    num_above = count_above(data_empty, 0, -1);
    printf("should be 0: %d\n", num_above);

    num_above = count_above(data_10, 10, -7);
    printf("should be 8: %d\n", num_above);

    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) {
    int i, count = 0;

    for(i=0; i<sz; i++) {
        if (data[i]>threshold) {
            count++;
        }
    }

    return count;
}
```

4. Complete the program below so that it prints the largest value found in the arrays `data1` and `data7`. Note that we must be able to initialize the arrays with a different set of values and your program must still work.
NOTICE: from the parameters documentation, the assumption the array has at least one element in it can be made.

```
#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 };

    int max;
    max = get_max(data1, 1);
    printf("max should be 5: %d\n", max);

    max = get_max(data7, 7);
    printf("max should be 34: %d\n", max);

    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) {
    int i, max = data[0];

    for(i=1; i<sz; i++) {
        if (data[i]>max) {
            max = data[i];
        }
    }
    return max;
}
```

5. Complete the function `is_increasing_by_1` that takes an array of integers and the number of elements in the array and determines whether the elements in the array are strictly increasing by 1. It should return 1 if they are and 0 otherwise. Your function can assume the array is not empty and behaves as described in the documentation. Given an array with one element does not contain elements that violate the condition to be strictly increasing by 1, the function called with an empty array should return 1. Note that we must be able to initialize the array with a different set of values (could be less or more values) and your program must still work.

```
#include <stdio.h>
int is_increasing_by_1(int data[], int sz);

int main( void ) {
    int data1_incr[1] = { 5 };
    int data7_notincr[7] = { 5, 3, -12, 34, 2, -17, 6 };
    int data4_incr[7] = { 2, 3, 4, 5 };
    int data4_notincr[7] = { 2, 3, 4, 4 };
    int data5_notincr[7] = { 3, 2, 3, 4, 5 };
    int data6_notincr[7] = { 2, 3, 4, 6, 7, 8 };

    int is_incr;
    is_incr = is_increasing_by_1(data1_incr, 1);
    printf("should be 1: %d\n", is_incr);

    is_incr = is_increasing_by_1(data7_notincr, 7);
    printf("should be 0: %d\n", is_incr);

    is_incr = is_increasing_by_1(data4_incr, 4);
    printf("should be 1: %d\n", is_incr);

    is_incr = is_increasing_by_1(data4_notincr, 4);
    printf("should be 0: %d\n", is_incr);

    is_incr = is_increasing_by_1(data5_notincr, 5);
    printf("should be 0: %d\n", is_incr);

    is_incr = is_increasing_by_1(data6_notincr, 6);
    printf("should be 0: %d\n", is_incr);

    return 0;
}

/*
 * Purpose:  determines whether sz elements in data are in increasing order
 *        going up by strictly 1
 * Params: int data[]
 *         int sz - number of elements in data, >0
 * Returns: int - 1 if data is increasing by 1, 0 otherwise
 */
int is_increasing_by_1(int data[], int sz) {
    int i, prev_val, ret_val;
    prev_val = data[0];

    i=1;
    while (i<sz && prev_val+1 == data[i]) {
        prev_val = data[i];
        i++;
    }

    if (i==sz) {
        ret_val = 1;
    } else {
        ret_val = 0;
    }

    return ret_val;
}
```

6. Design a function that prompts the user for a series of positive integers between 1 and 100. They will enter a -1 when they have entered all of the values. You can assume they will not enter an invalid number.
   The function must then print a histogram (on its side) to show the distribution of numbers in the ranges 1-10, 11-20, 21-30, …, 81-90, 91-100. So, for example, if the user enters the values:

   21 45 63 12 6 89 65 41 27 18 77 54 45 44 -1

   Your program will print:

```
 1 - 10: *
11 - 20: **
21 - 30: **
31 - 40:
41 - 50: ****
51 - 60: *
61 - 70: **
71 - 80: *
81 - 90: *
91 -100:
```

   HINT: What do you need to keep count of as the user enters values? How many variables will you need to keep these counts? What might be a good choice for a data type for the variable that holds these counts?

   Think about what order your program needs to do things… Can you print the distribution before the user has entered all of the numbers?

```c
#include <stdio.h>

#define SENTINEL -1
#define MIN_VAL 1
#define MAX_VAL 100
#define NUM_BINS 10

void print_histogram();
void print_n_chars(int n, char ch);

int main( void ) {
    print_histogram();

    return 0;
}

/*
 * Purpose:  prompts the user for integers between MIN_VAL and MAX_VAL
 *  terminated with SENTINEL.
 *  Prints a histogram of number distribution in NUM_BINS number ranges
 */
void print_histogram() {
    int next_val, i;
    int bin_size = MAX_VAL/NUM_BINS;
    int counts[NUM_BINS] = {0}; // initializes all values in counts to 0

    printf("enter a number %d to %d, -1 to stop\n", MIN_VAL, MAX_VAL);
    scanf("%d", &next_val);

    while(next_val != SENTINEL) {
        int index = (next_val-1)/bin_size;  // index of counts array to be incremented
        counts[index] = counts[index] + 1;  // can be written as counts[index]++
        printf("enter a number %d to %d, -1 to stop\n", MIN_VAL, MAX_VAL);
        scanf("%d", &next_val);
    }

    for (i=0; i<NUM_BINS; i++) {
        printf("%3d  -%5d: ", i*bin_size+1, (i+1)*bin_size);
        print_n_chars(counts[i], '*');
        printf("\n");
    }
}

/*
 * Purpose: print n copies of ch on one line
 * Parameters: int n,  >=0
 *       char ch – character in single quotes (ie. ‘a’)
 */
void print_n_chars(int n, char ch) {
    int count;

    for (count = 0; count<n; count++) {
        printf("%c", ch);
    }
}
```