char arrays, strings and string.h

1. Recall, a char array can be initialized with a string literal (characters enclosed in " ")
   Recall the format specifiers: `%s` to print a null terminated string, `%c` to print a char, `%d` to print an integer.
   Consider the following code segments.
   a. Given the following declaration, draw the array that represents `str_0`. Indicate garbage data with a '– '

```
char str_0[5];
```

   b. Given the following declaration, draw the array that represents `str_1`. Indicate garbage data with a '– '
   What is the output of the print statement?

```
char str_1[5] = "Hi";
printf("str_1: %s\n", str_1);
```

   c. Given the following declaration and assignments, draw the array that represents `str_2` and `str_3`.
   Indicate garbage data with a '– '
   What is the output of the print statements?

```
char str_2[5] = "HiHi!";
char str_3[5] = "Bye";
printf("str_2: %s\n", str_2);
printf("str_3: %s\n", str_3);
```

   Update your drawing to reflect the following change. What is the output of the print statements?

```
str_2[2] = '\0';
printf("str_2: %s\n", str_2);
printf("str_3: %s\n", str_3);
```

2. Complete the `string_length` function that takes a character array containing a null terminated string. The function should return the length of that string (ie. the number of characters in the array before the null terminator). Add calls to complete the testing in the main that has been started for you.

```
int string_length(char str[]);

int main() {
    char str_mt[] = "";
    char str_not_full[10] = "Hello!!";
    char str_full[6] = "Hello";
    int len = 0;

    // test empty string:
    len = string_length(str_mt);
    printf("length should be 0: %d\n", len);


    // test not full array:
    len = string_length(str_not_full);
    printf("length should be 7: %d\n", len);


    // test full array:
    len = string_length(str_full);
    printf("length should be 5: %d\n", len);
}

/*
 * Purpose: count characters in src upto but not including
 *      the null terminator.
 * Parameters: char src[], a null terminated string
 * Returns: int, the count of characters
 */
int string_length(char src[]) {









}
```

3. Design and test a function called `string_copy` that takes 2 character arrays, the first being the destination and the second being the source array that is a null terminated string. The function should copy the characters from the source array to the destination array up to and including the null terminator.

4. Design and test a function called `string_reverse` that takes a char array which is a null terminated string and reverses the order of the characters in the string.

5. Design and test a function called `string_append` that takes 2 character arrays, the first being the destination and the second being the source array which are both null terminated strings. The function should copy the characters from the source to destination array starting at the position of the null terminator in the destination, up to and including the null terminator in the source. The destination array is guaranteed to be large enough to hold the appended strings and a null terminator at the end.
   For example if the destination string was `"abc"` and the source string was `"fgh"`, when the function is complete the destination string is `"abcfgh"` and the source string is still `"fgh"`.

6. The following function prototypes/documentation are part of the C Library specified in `string.h`
   Their behavior should look familiar given the functions you have just written.

```
char *strcpy(char *dest, const char *src)
Copies up to n characters from the string pointed to, by src to dest.

char *strcat(char *dest, const char *src)
Appends the string pointed to by src to the end of the string pointed to by
dest.

int strlen (const char *src)
Computes the length of the string str up to, but not including the
terminating null character.
```

   Design a function that takes a string and makes a copy of that string, reverses the copy and appends the reversed string to the original string. Use the built-in functions from `string.h`. Your function can assume that the size of the input array that holds the characters of the original string is large enough to accommodate the appended strings. The original string should **not** be changed.