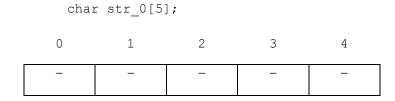
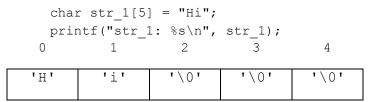
char arrays, strings and string.h

- 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 -



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?



c. Given the following declaration and assignments, draw the array that represents str_2 and str_3.
 Indicate garbage data with a -

4

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); // output: HiHi!Bye
printf("str_3: %s\n", str_3); // output: Bye
```

```
0 1 2 3
```

'H'	'i'	'H'	'i'	'!'
0	1	2	3	4
'B'	'Y'	'e'	'\0'	'\0'

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); // output: Hi
printf("str_3: %s\n", str_3); // output: Bye

only str 2 changes:

0	1	2	3	4
'H'	'i'	'\0'	'i'	'!'

 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[]) {
    int index = 0;
    while (src[index] != ' \setminus 0') {
       index++;
    }
    return index;
}
//alternative solution:
int string length(char src[]) {
    int count = 0;
    while(*src != ' \setminus 0') {
       src++;
       count++;
    }
   return count;
}
```

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.

```
/* Purpose: copies characters from src to dest, stores
 * a null terminator in dest following all characters from src
* Params: char dest[] - array with capacity >= length of src +1
 *
           char src[] - a null terminated string
 */
void string copy(char dest[], char src[]) {
    int index = 0;
    while(src[index] != '\0') {
        dest[index] = src[index];
        index++;
    }
    dest[index] = ' \setminus 0';
}
//alternate solution:
void string copy(char dest[], char src[]) {
    while (*src != ' \setminus 0') {
       *dest++ = *src++;
    }
    *dest = ' \setminus 0';
}
```

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.

```
/\star Purpose: reverses the order of characters in str
* Params: char src[] - a null terminated string
 * Assumes MAX SIZE is defined to be large enough to hold
     the maximum string size + null terminator
 *
*/
void string reverse(char src[]) {
   int len = strlen (src);
    char temp array[MAX SIZE];
   int src front = 0;
   int temp back = len-1;
   temp array[len] = '\0';
   while(src[src front] != '\0') {
        temp array[temp back] = src[src front];
       src front++;
       temp back--;
    }
   strcpy(src, temp array);
}
//alternate solution:
void string reverse(char src[]) {
   int len = strlen(src);
    char temp;
    int front index = 0;
    int back index = len- 1;
   while(front index <= back index) {</pre>
        temp = src[back index];
        src[back index] = src[front index];
        src[front index] = temp;
        front index++;
        back index--;
    }
}
```

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.

```
/* Purpose: copies characters from src to end of dest
 * stores a null terminator in dest after all characters from src
 * Params: char dest[] - a null terminated string with
 *
                          capacity >= (length of dest + length of src +1)
 *
           char src[] - a null terminated string
 */
void string append(char dest[], char src[]) {
    int len = strlen(dest);
    int src index = 0;
    int dest index = len;
    while(src[src index] != ' \setminus 0') {
        dest[dest index++] = src[src index++];
    }
    dest[dest index] = ' \setminus 0';
}
```

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 characters from the string pointed to, by src to dest including the terminating null character. Assumes dest is big enough to hold characters in src + terminating null character. 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.

```
/* Purpose: copies src to dest and then appends the reverse of src to dest
 * Params: char dest[] - a null terminated string with
 * capacity >= (2 *length of src +1)
 * Assumes MAX_SIZE is defined to be large enough to hold
 * the maximum string size + null terminator
 * char src[] - a null terminated string
 */
void strcat_rev(char dest[], char src[]) {
 char rev[MAX_SIZE];
 strcpy(rev, src);
 string_reverse(rev);
 strcpy(dest, src);
 strcpy(dest, src);
 strcat(dest, rev);
}
```