Week 9: In-Class Exercises

Please download the resource from *http://blue.smu.edu.sg/cs101/ice9-resource.zip*

1. **[Difficulty: *]** Implement the `add_or_multiply` function. The function takes in 3 parameters:
   a. `array` (type: `int[]`): The integers to be processed.
   b. `n` (type: `int`): The size of the array. Assume n > 0.
   c. `op` (type:`char`): Either `'+'` or `'*'`

   This function either adds or multiplies all items in the array, and returns the result.

2. **[Difficulty: *]** Implement `print_large_two` function. It takes in an array of integers and prints the two largest values.
   Note: You are **NOT** allowed to sort the array.

3. **[Difficulty: *]** Implement the `my_strlwr` function. The function takes in a string (type: `char[]`) and converts all uppercase letters to lowercase letters. It will leave all other characters untouched.
   Note:
   a. Do not use any string methods in the string library (i.e. do not include `<string.h>`).
   b. `strlwr` is not a standard function available in the string library

4. **[Difficulty: *]** Implement the `my_strupr` function. The function takes in a string (type: `char[]`) and converts all lowercase letters to uppercase letters. It will leave all other characters untouched.
   Note:
   a. Do not use any string methods in the string library (i.e. do not include `<string.h>`).
   b. `strlwr` is not a standard function available in the string library

5. **[Difficulty: *]** Implement the `my_strcat` function. The function takes in 2 parameters:
   a. `dest` (type: `char[]`)
   b. `src` (type: `char[]`)

   It will copy all the characters(including `'\0'`) from `src` to the end of the string of `dest`.
   Note: Do not use any string methods in the string library (i.e. do not include `<string.h>`).

6. **[Difficulty: *]** Implement the `my_strcpy` function. The function takes in 2 parameters:
   a. `dest` (type: `char[]`)
   b. `src` (type: `char[]`)

   It will copy all the characters(including `'\0'`) from `src` to `dest`.
   Note: Do not use any string methods in the string library (i.e. do not include `<string.h>`).

7. **[Difficulty: *]** Implement the `str_chr` function. The function takes in 2 parameters:
   a. `array` (type: `char[]`): the string to be searched
   b. `look_for` (type:`char`): the character we are looking for.

   This function returns a pointer to the located character, or a null pointer if the character does not occur in the string.
   Note: Do not use any string methods in the string library (i.e. do not include `<string.h>`).

8. **[Difficulty: **]** Implement the `str_spn` function. The function takes in 2 parameters:
   a. `s1` (type: `char[]`): The target string in which search has to be made
   b. `s2` (type:`char[]`): string containing characters to match in target string

   This function returns the number of leading characters at the beginning of the string `str1` which are all members of string `s2`.
   Note: Do not use any string methods in the string library (i.e. do not include `<string.h>`).

9. **[Difficulty: ***]** Implement the `my_strstr` function. The function takes in 2 parameters:
   a. `haystack` (type: `char[]`): The string to be scanned
   b. `needle` (type: `char[]`): The shorter string to be searched within haystack string

   This function returns a pointer to the first occurrence of the given string.

10. **[Difficulty: ***]** No matter how fast today's computers and networks are, the users are constantly in need of faster services. To reduce the volume of the transferred data we usually use some sort of compression. Run-length encoding (RLE) is a very simple form of lossless data compression. This technique basically compresses data that contains consecutive occurrences of a character. If a character appears more than once consecutively in the sequence, the characters are compressed into a single character followed by the count of the number of times that the character appears. As an example, "wwww" is compressed into "w4" (See sample run of the program below for more examples). For simplicity, assume that the sequence contains only case-sensitive alphabets (i.e., the character 'W' and the character 'w' are not the same).

    Note: You can assume that only alphabets are entered, there are no digits.

```
Enter word>aaaaaaaaaabbbaxxxxyyyzyx
a10b3a1x4y3z1y1x1
```

.

Reference:
http://stoimen.com/2012/01/09/computer-algorithms-data-compression-with-run-length-encoding/

Hint:

```
// sprintf store output to a char buffer which are specified
// instead of printing it out

char line[50] = {0};
char *ptr = line;

// n stores the number of characters written
int n = sprintf(ptr, "%d", 12);
ptr += n;

n = sprintf(ptr, "%d", 34);
ptr += n;

printf("%s \n", line);
```

OPTIONAL

11. **[Difficulty: **]** The Caesar cipher is one of the earliest known and simplest ciphers. It is a type of substitution cipher in which each letter in the plaintext is 'shifted' a certain number of places down the alphabet. For example, with a shift of 1, A would be replaced by B, B would become C, and so on. The method is named after Julius Caesar, who apparently used it to communicate with his generals.

    ROT13 (shift of 13) was used with postings that contained offensive material so that the more sensitive ones wouldn't be inadvertently exposed to them. The idea was that you had to take an action to decode the posting as a way of indicating that you understood that you might find the content offensive. Implement the encode and decode functions.
    Reference: http://practicalcryptography.com/ciphers/caesar-cipher

12. **[Difficulty: **]** Implement the `is_sorted` function. This function returns `true` if the array is sorted in increasing or decreasing order. Otherwise, it returns `false`.

13. **[Difficulty: *]** Implement the `remove_consecutive_duplicates` function that takes as input an array of characters, and removes all consecutive duplicate elements. The function returns the number of items left in the array.

14. **[Difficulty: *]** Implement the `merge` function. It takes in two arrays A and B, and interleaves the arrays by taking A[0], B[0], A[1], B[1] ... If the two arrays have different lengths, the remaining items from the longer array are appended behind.

15. [Difficulty: **] To verify the accuracy and validity of credit card numbers, most credit card numbers are encoded with a "check digit". A check digit is a digit added to a number (at either the end or the beginning) that validates the validity of the number. A simple algorithm is applied to the other digits of the number, which yields the check digit.

The steps for validation are as follows:

A. The alternate digits from the leftmost digit or the digits in the even position starting from the right are multiplied by 2, and then reduced to a single digit by subtracting 9.

B. Add the unaffected digits (shown in blue below) in the original number with the individual digits comprising the products obtained in step (i).

C. The total obtained in step (ii) must be divisible by 10.

For example, given the number 2323 2005 7766 3554, this is how step (i) is performed:

		Multiply by 2	Subtract 9 if value >= 10
1st digit	2	2 * 2 = 4	4
2nd digit	3		
3rd digit	2	2 * 2 = 4	4
4th digit	3		
5th digit	2	2 * 2 = 4	4
6th digit	0		
7th digit	0	0 * 2 = 0	0
8th digit	5		
9th digit	7	7 * 2 = 14	14 - 9 = 5
10th digit	7		
11th digit	6	6 * 2 = 12	12 - 9 = 3
12th digit	6		
13th digit	3	3 * 2 = 6	6
14th digit	5		
15th digit	5	5 * 2 = 10	10 - 9 = 1
16th digit	4		

In step (ii), the following numbers are added together: 4 + 3 + 4 + 3 + 4 + 0 + 0 + 5 + 5 + 7 + 3 + 6 + 6 + 5 + 1 + 4 which yields the answer 60. Because 60 is divisible by 10, this credit card number is a valid one (corresponds to step (iii)).

Implement the `verify` function that returns true if the credit card number is valid. Otherwise, return false. Here are three valid credit card numbers for testing purposes:

A. 4041422060806790
B. 5222747000084993
C. 5256392810443201

16. **Difficulty: *]** Implement the `selection_sort` function. The selection sort algorithm has the following steps:
    a. the smallest value in the array is located, and then it is swapped with the value stored in the first position of the array
    b. The above step is repeated for the rest of the array until the whole array is sorted (2nd smallest swapped with the value stored in the 2nd position of the array, 3rd smallest swapped with the value stored in the 3rd position of the array ...).

    For example, given the following array:

idx	0	1	2	3	4
value	6	5	1	3	2

Step 1: Find the smallest element in the array from **index 0** to index 4, and place it at **index 0**. The value found at index 0 will now be at the index where the smallest element is found.

idx	0	1	2	3	4
value	**1**	5	**6**	3	2

Step 2: Find the smallest element in the array from **index 1** to index 4, and place it at **index 1**. The value found at index 1 will now be at the index where the 2nd smallest element is found.

idx	0	1	2	3	4
value	1	**2**	6	3	**5**

Step 3: Find the smallest element in the array from **index 2** to index 4, and place it at **index 2**. The value found at index 2 will now be at the index where the 3rd smallest element is found.

idx	0	1	2	3	4
value	1	2	**3**	**6**	5

Step 4: Find the smallest element in the array from **index 3** to index 4, and place it at **index 3**. The value found at index 2 will now be at the index where the 3rd smallest element is found.

idx	0	1	2	3	4
value	1	2	3	5	6

The array is now sorted.