

# Assignment 1(V3. Edited 2020.09.28): Overview

---

Modify the provided skeleton code and implement the following five functions as shown below.

- Do not modify the header file and function declaration.
- You can not use any C/C++ library functions, other than malloc, new, free, and delete.
- All the implementation should be executed without any error if there is no assumption, e.g., your implementation should check all corner cases that may cause segmentation faults.

```
int count_odd(const unsigned int* array, int size); (2pt)
int normalize(double* array, int size); (3pt)
int find_nth(const int* array, int size, int n); (3pt)
int count_pattern(const char* str, const char* pattern); (7pt)
char* create_shortest_palindrome(const char* src, char* dst); (10pt)
```

## Due

---

Submit your implementation before Oct 5, Monday, 11:59:59pm, to LMS. We DO NOT allow late submission.

## Submission

---

Submit the cpp file that has your implementation. You have to implement all the functions in the single cpp file. Before the submission, rename the cpp file with this format:

“hw1\_YOURSTUDENTID.cpp”. For example, “hw1\_200012345.cpp”

# Function Specification

---

1. `int count_odd(const unsigned int* array, int size);`
  - a. Return the number of odd numbers in the given array
    - i. You may change the contents of the array if required
  - b. Parameters:
    - i. `array`: an unsigned integer array
    - ii. `size`: the number of the array elements
2. `int normalize(double* array, int size);`
  - a. Implement a function that scales the array elements. The function should modify each array element using the following equation where  $a_i$  is an array element,  $a_{max}$  is the largest element, and  $a_{min}$  is the smallest element in the array.

$$a_i = \frac{a_i - a_{min}}{a_{max} - a_{min}}$$

- \* Note: In this way, you make all elements in the range of [0.0, 1.0]
  - b. Return 0 if it fails due to division-by-zero; return 1 otherwise.
  - c. Parameters:
    - i. `array`: a double array
    - ii. `size`: the number of the array elements
3. `int find_nth(const int* array, int size, int n);`
  - a. Implement a function that finds and returns the n-th largest element in the array. For example, if the array contains {1, 5, 3, 4, 2}, the 3-th (third) largest element is 3 and the 4-th largest element is 2.
  - b. Assume there is no same value in the given array.
  - c. Return 0 if an error occurs due to any reason, e.g., n is out of range.
  - d. Parameters:
    - i. `array`: a double array
    - ii. `size`: the number of the array elements
    - iii. `n`: an integer value for the “n” to find
4. `int count_pattern(char* str, char* pattern);`
  - a. Return how many times pattern appears in str.
  - b. Parameters:
    - i. `str`: a null-terminated byte string (i.e., character array)
    - ii. `pattern`: a null-terminated byte string to search

- c. Assume the character arrays only includes English alphabet
5. `char* create_shortest_palindrome(const char* src, char* dst);`
- a. Fill the `dst` array with the shortest palindrome which starts with the characters in `src`.
    - i. For example, if `src` is "ABCDD", the `dst` is "ABCDDCBA".
    - ii. See the sample out for more examples.
    - iii. Hint: think thoughtfully -- there are many corner cases!
  - b. Return the `dst` variable
  - c. Assume the size of `dst` is sufficient, i.e., always greater than the length of `src * 2`.
  - d. Assume the character arrays only includes English alphabet
  - e. Parameters:
    - i. `str`: a character array for the string
    - ii. `dst` : a null-terminated byte string to store

## Sample

---

Note: We will check with other example inputs for grading.

### Sample Code

```
int main() {
    using std::cout;
    using std::endl;

    // Problem 1
    unsigned int a[] = {0, 1, 2, 3, 5};
    std::size_t size_a = sizeof(a) / sizeof(int);
    cout << "Problem 1: " << count_odd_nums(a, (int)size_a) << endl;

    // Problem 2
    double b[] = {1, 2, 2, 3, 2};
    std::size_t size_b = sizeof(b) / sizeof(double);
    normalize(b, (int)size_b);
    cout << "Problem 2: ";
    for (unsigned int i = 0; i < size_b; ++i) {
        cout << b[i] << " ";
    }
    cout << endl;

    // Problem 3
    int c[] = {1, 5, 3, 2, 4};
    cout << "Problem 3: " << find_nth(c, sizeof(c) / sizeof(int), 3) << endl;
```

```

// Problem 4
cout << "Problem 4: ex1) " << count_pattern("AABBBBAA", "AA") << endl;
cout << "Problem 4: ex2) " << count_pattern("AABBBBAA", "BB") << endl;

// Problem 5
char d[100] = {0};
cout << "Problem 5: ex1) " << create_shortest_palindrome("ABCD", d) << endl;
cout << "Problem 5: ex2) " << create_shortest_palindrome("AABBCC", d) << endl;

return 0;
}

```

## Output

```

Problem 1: 3
Problem 2: 0 0.5 0.5 1 0.5
Problem 3: 3
Problem 4: ex1) 2
Problem 4: ex2) 3
Problem 5: ex1) ABCDCBA
Problem 5: ex2) AABBCCBBAA

```

# Update Log

---

## Changes in V2 from V1

tl;dr: Download the skeleton file again, and follow the instruction described above

1. Skeleton code changes
  - a. Minor bug fix
    - i. Remove “const” typo written at the beginning of the header file
    - ii. Change the function name, “count\_odd\_num” to “count\_num” as written in the instruction
    - iii. The second sizeof(b) is changed to sizeof(c)
  - b. Requirement change
    - i. Add “const” for the array parameters for Problem 1 and 3, i.e.,  
You cannot change the elements of array in the functions
2. Instruction changes
  - a. Fix the sample code for the minor bug (1.c) above
  - b. Clarify Problem 4 and 5
  - c. Change the array modification requirement for Problem 1 and 3
    - i. Now the array parameter is specified with “const”