# Wesleyan University, Fall 2023, COMP 211

# Lab 6: Duplicate and Distinct Array Entries

---

## 1. Overview

The goal of this lab is to give you a chance to practice solving a few small programming problems and to get more familiar with testing and checking code correctness.

## 2. Create and organize your program files

For this lab, you should create a new directory lab6 in your labs directory to hold your code. You should download the `Makefile` from lab6 webpage, and put it in your `lab6` directory. You should then create 3 files within the `lab6` directory.

```
array_func.c
array_func.h
tests.c
```

Be sure to include `array_func.h` in both `array_func.c` and `tests.c`. Include `stdio.h`, `stdbool.h`, `string.h` in `array_func.h`. You should then create a main function in the `tests.c` file.

You will use the `Makefile` to compile the program you create (which currently does nothing) and run it by typing `./tests` at the command line in the terminal.

## 3. Check for duplicate array entries

Implement a function in `array_func.c` matching the following function signature. Make sure to add the function signature to `array_func.h`.

```
bool all_distinct(char A[], int n)
```

This function is passed an array `A` and a number `n` and examines the subarray from element 0 up to and including element `n-1`. This function should return true if the specified sub-array contains no repeated characters and false otherwise. The function assumes that the given array `A` is sorted alphabetically and that `n` is less than the length of `A`.

Create a test function, `void test_all_distinct()` in the file `tests.c` and add a call to `test_all_distinct` in `main`. Add some tests to your test function. Three possible tests you might consider adding are to test (1) when the only duplicate is the first element of the array, (2) when the only duplicate is the last element of the array, and (3) when there are duplicates in the array but beyond the `n`th element.

## 4. Count distinct array entries in linear time

Implement a function in `array_func.c` matching the following function signature. Make sure to add the function signature to `array_func.h`.

```
int count_distinct(char A[], int n)
```

This function is passed an array `A` and a number `n` and examines the subarray from element 0 up to and including element `n-1`. This function should return the number of distinct characters in the array (i.e., if the same value occurs more than once it should contribute only one to the count). The function assumes that the given array `A` is sorted alphabetically and that `n` is less than the length of `A`.

Your implementation should have a linear running time. Think carefully about this: what precondition should you exploit to reduce the running time of the function?

In the file `tests.c`, write a function `void test_count_distinct()` with some tests for `count_distinct`. Make sure to add a call to `test_count_distinct` in `main`.

## 5. Remove duplicates in linear time

Implement a function in `array_func.c` matching the following function signature. Make sure to add the function signature to `array_func.h`.

```
void remove_duplicates(char A[], int n, char B[], int m)
```

This function should examine only the characters in `A` from element 0 up to and including element `n-1`. The characters in `A` should be sorted before the array is passed to your function. This function should assign to `B` only one copy of each distinct character in `A` (i.e., only those characters that are not duplicates). The size of `B` is given by `m` and can be computed by first calling `count_distinct` before calling `remove_duplicates`.

Just like `count_distinct`, your implementation should have a linear running time.

In the file `tests.c`, write a function `void test_remove_duplicates()` with some tests for `remove_duplicates`. Make sure to add a call to `test_remove_duplicates` in `main`.

## 6. Add assertions to your functions

We can use assertions to assert that something should be true about your code at any given point in the program. For example you might wish to use assertions to check that the value of a variable is as you think it should be.

To use assertions, include the header file `assert.h` in `array_func.h`. The function call `assert(e)` takes a boolean expression `e` and evaluates it. This expression could be another function call. If `e` evaluates to true, nothing happens and control passes to next instruction in your code. If `e` evaluates to false, then an error message is printed to the terminal and execution terminates without executing any more lines of code.

Now, go back and look at the functions you have just created. Where might you want to add assertions? How would you go about putting your testing function calls within asserts?

**Challenge.** For any of functions you've written in your current or previous homework, where and what assertions might you want to add to your code?