CS 31 Worksheet 3

This worksheet is entirely **optional**, and meant for extra practice. Some problems will be more challenging than others and are designed to have you apply your knowledge beyond the examples presented in lecture, discussion or projects. All exams will be done on paper, so it is in your best interest to practice these problems by hand and not rely on a compiler.

Concepts

Arrays (1D, 2D), Pass by reference/value, Arrays as parameters in functions

1) Write a function that takes in two parameters: (1) an array of integers called foo, (2) the size of the array, and (3) an integer called n. Return *true* if they are "equal" and *false* otherwise.

```
int bar[5] = {7, 8, 2, 1, 6};
int m = 78216;
equality(bar, 5, m) → returns True

int baz[3] = {3, 2, 1};
int x = 312;
equality(baz, 3, x) → returns False
```

Function header: `bool equality(int foo[], int n);`

2) Create a function that accepts three parameters: (1) an integer array, (2) the size of the array, and (3) a target value, *k*. Then return *true* or *false* depending on whether there exists at least one pair of numbers within the array whose sum is *k*.

Sample: `[1, 5, 6, 20, 10], size = 5, target = 25`
Output: `True, because 20 + 5 = 25`

Sample: `[1, 23, 3], size = 3, target = 22`
Output: `False. There are no pairs whose sum is 22.`

Function header: `bool isPair(int foo[], int size, int k);`

3) After Halloween, Frank has *N* candies, but he must share *N/2* with his sister, Mandy. He wants to maximize the number of unique candies he can have after dividing his stash in half, with each candy being represented with a different integer. Implement the following function to find the maximum

number of unique candies.

Sample: `[10, 10, 10, 10, 2, 5]`
Output: `3. Frank can keep [2, 5, 10] for himself and share [10, 10, 10] with his sister.`

Sample: `[2, 2, 10, 10, 10, 2]`
Output: `2. Frank can keep [2, 10, 10] or [2, 2, 10] for himself, both having a maximum of 2 unique candies, and share [2, 2, 10] or [10, 10, 2] with his sister.`

Function header: `int maxUnique(int candies[], int k)`

4) Write the function zeroLeft which takes in an array of 1's and 0's and modifies the array so all of the 0's are on the left and all of the 1's are on the right.

```
int foo [7] = {1, 1, 0, 0, 0, 1, 0}
zeroLeft(foo, 7);
// foo = {0, 0, 0, 0, 1, 1, 1}
```

Function header: `void zeroLeft(int array[], int array_size);`

5) Write a function with the following header:

`void exclusiveProduct(int nums[], int len)`
where *nums* is an array of positive numbers and *len* is the length of *nums*.

This function should alter *nums* such that *nums*[i] is the product of all *nums*[j] where j != i (in which *nums*[j] is the value before it was modified).

Example:

```
int foo[3] = {3, 1, 2}
exclusiveProduct(foo, 3)
//foo now equals {2, 6, 3}.
// At index 0, the new value is 2 because it is the product of 1 and 2, all the integers in the array not at index 1.
// At index 2, the new value is 3 because it is the product of 3 and 1, which were the original values of foo at index 0 and 1, respectively.
```

6) Write a function with the following header:

```
bool rangeSearch(const int sorted_nums[], int len, int target,
                 int& start, int& end)
```

sorted_nums is a sorted array of positive numbers
len is the length of *sorted_nums*
target is a number to search for within the array

This function should return *true* if *target* is contained within the array and *false* otherwise.

If the function returns *true*, *start* should be set to the first index where *target* appears and *end* should be set to the last index where *target* appears.

If the function returns *false*, *start* and *end* should not be altered.

Sample: `[1, 2, 3, 3, 3, 4, 5], target = 3`
Output: `True. start = 2, end = 4.`

7) Create a function that accepts three parameters: (1) a SORTED integer array, (2) the size of the array, and (3) a target value, *k*. Then return true or false depending on whether there is at least one triplet of numbers within the array whose sum is *k*. A triplet is any group of three numbers that come from the array.

   Sample: `[1, 5, 6, 20, 40], size = 5, target = 27`
   Output: `True, because 1 + 6 + 20 = 27`

   Sample: `[1, 23, 3], size = 3, target = 22`
   Output: `False. There are no triplets whose sum is 22.`

8) Write a function that takes two parameters: (1) a 2-dimensional integer array of size n x n, and (2) n as an integer. This function should print out the integers on the diagonal of the array.

   Sample:
   ```
   {{1, 2, 3},
    {4, 5, 6},
    {7, 8, 9}}, n = 3
   ```

   Output: `159`

9) You are given an array of integers called A, which has size n and whose elements have values ranging from 1 to n-1. Assume n < 500.

Write a function *findDuplicate* that takes in the array A, and returns any element of A that is not unique (if multiple elements satisfy that property, return any one of them). The function prototype is given to you below:

```
int findDuplicate(const int A[], int n);

int A[7] = {1, 3, 5, 3, 2, 6, 1}
findDuplicate(A, 7) → returns 3 or 1
```

10) Write a function that takes two parameters: (1) a 2-dimensional integer array of size n x n, and (2) n as an integer. This function should not return anything, but simply reflect the original array on the main diagonal, so that the rows of the original array should become the columns of the array and vice versa.

```
Sample: {{1, 2, 3},
         {4, 5, 6},
         {7, 8, 9}}, n = 3

Modified array: {{1, 4, 7},
                 {2, 5, 8},
                 {3, 6, 9}}
```