CS 31 Worksheet 5

https://tinyurl.com/LA-Feedback-S25

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.

Solutions are written in red. The solutions for **programming** problems are not absolute, it is okay if your code looks different; this is just one way to solve the specific problem.

Concepts

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

Reading Problems

1) What is the output of the following program

```
#include <iostream>
#include <string>
using namespace std;

string foobar(string s[], int n)
{
    int i = n - 1;
    string str;
    for (str = s[i]; i > 2; i--)
    {
        s[i] = s[i - 1];
    }
    string temp = s[4];
    s[4] = s[n - 1];
    s[n - 1] = temp;
    s[i + 1] = str;
    return str;
}

int main()
{
    string sentence[9] = {"the", "quick", "brown", "fox",
```

```
                            "jumps", "over", "the", "lazy", "dog"};
    foobar(sentence, 9);
    for (int i =
0; i < 9; i++)
        cout << sentence[i] << " ";
    cout << endl;
}
```

the quick brown dog lazy jumps over the fox

2)

```
int mystery(int arr[], int n, int target)
{
    int begin = 0;
    int end = n-1;
    while (begin <= end)
    {
        int mid = (begin + end) / 2;
        if (arr[mid] > target)
           end = mid - 1;
        else if (arr[mid] < target)
            begin = mid + 1;
        else
            return mid;
    }
    return begin;
}
```

This function is passed an array for `arr` which has elements in strictly increasing order. `n` is the array's size, and `target` is any integer value. What does the function return?

A. position of smallest element
B. position of largest element
C. number of elements less than `target`
D. number of elements less than or equal to `target`
E. number of elements equal to `target`
F. number of elements greater than or equal to `target`
G. number of elements greater than `target`

C. Specifically, it returns the index of the target element in the array by dividing the array in half until it finds the value of the target element. Since this starts counting from 0, this index can be interpreted as the number of elements before the target element.

Programming Problems

1) Write a function with the following header:

```
bool equality(int arr[], int n, int num);
```

`arr` is an array of non-negative integers
`n` is the number of elements in `arr`
`num` is a non-negative integer

`equality` should return *true* if each digit of `num` is "equal" to the corresponding element in `arr` (i.e. the first digit of `num` is the first element in `arr`, the second digit of `num` is the second element in `arr`, and so forth). It returns *false* otherwise.

Examples:

```
int foo[5] = {7, 8, 2, 1, 6};
int m = 78216;
bool b1 = equality(foo, 5, m); // returns true

int bar[3] = {3, 2, 1};
int x = 312;
bool b2 = equality(bar, 3, x);  // returns false
```

```
We can exploit the fact that some number%10 extracts its last digit,
and some number/10 removes its last digit.

bool equality(int foo[], int n, int num) {
    for (int i = n-1; i >= 0; i--) {
        if ((n <= 0) || foo[i] != (num%10)) {
            return false;
        }
        num /= 10;
    }
    if (num == 0)
        return true;
    else
        return false;
}
```

2) Write a function with the following header:

```
bool hasTwoSum(int arr[], int n, int sum);
```

`arr` is an array of integers
`n` is the number of elements in `arr`
`sum` is a target value

`hasTwoSum` should return *true* if there exists at least one pair of distinct numbers in `arr` which adds up to equal `sum`. It returns false otherwise.

Examples:

```
int foo[5] = {12, 5, -6, 20, 10};
bool b1 = hasTwoSum(foo, 5, 25); // returns true; 20 + 5 = 25

int bar[3] = {1, 23, -3};
bool b2 = hasTwoSum(bar, 3, 22); // returns false; no pairs sum to
22.
```

```
bool hasTwoSum(int arr[], int n, int sum) {
    for (int i = 0; i < n; i++) {
        for (int j = i + 1; j < n; j++) {
            if (arr[i] + arr[j] == sum && arr[i] != arr[j])
                return true;
        }
    }
    return false;
}
```

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 Frank can have. You may assume there will be no more than 100 unique candies.

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

Examples:

```
int foo[6]  = {10, 10, 10, 10, 2, 5};
int i1 = maxUnique(foo, 6); // returns 3

int foo1[6] = {2, 2, 10, 10, 10, 2};
int i2 = maxUnique(foo1, 6); // returns 2

int maxUnique(int candies[], int n) {
    int unique_candies[100];
    int numUnique = 0;
```

```
    int half_candies = n/2;

    for (int i = 0; i < n; i++) {
        bool isNewCandy = true;
        for (int j = 0; j < numUnique; j++) {
            if (candies[i] == unique_candies[j]) {    // compares
what
                                                    // is held in the array of
                                                    //unique candies and what
                                                    //is in the overall stash
                isNewCandy = false;
                break;
            }
        }
        if (isNewCandy) {  // if true
            unique_candies[numUnique] = candies[i];
            numUnique++;
        }
    }

    if (numUnique < half_candies) {
        return numUnique;
    } else {
        return half_candies;  // Frank can only have half of the
stash
    }                         // he can have at most 50 unique candies
}
```

4) Write a function with the following header:

```
void zeroLeft(int binary[], int n);
```

`binary` is an array of only 1's and 0's
`n` is the number of elements in `binary`

`zeroLeft` should alter `binary` so that all of the 0's are on the left and all of the 1's are on the right.

Example:

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

```
void zeroLeft(int binary[], int n) {
```

```
	// counter for number of 0's
	int num0 = 0;
	for (int i = 0; i < n; i++)
	{
		// if the current index contains a 0, increase the counter
		if (binary[i] == 0) {
			num0++;
		}
	}
	// the following lines alter binary to contain the correct
order of 0's and 1's
	// since all 0's go to the left, all indexes less than our
counted number of 0's will now contain 0's
	int i = 0;
	while (i < num0){
		binary[i] = 0;
		i++;
	}
	// the remainder of indexes will contain 1's. note that we
haven't reset the value of i.
	while (i < n) {
		binary[i] = 1;
		i++;
	}
}
```

5) 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:

If every element in A is unique, return 0.

Function header: `int findDuplicate(int A[], int n);`

Example:

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

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

```
    bool isFound[500]; // creates a bool array of fixed size 500
    for (int i = 0; i < 500; i++) { // set each element as false
        isFound[i] = false;
    }
    for (int i = 0; i < n; i++) {
        if (isFound[A[i]]) { // if the array has already this
element, then return the first duplicate element
            return A[i];
        }
        isFound[A[i]] = true; // set value as true if it is the
current element
    }
     return 0; // return 0 if no duplicates
}
```

6) 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.

Example:

```
int foo[5] = {1, 5, 6, 20, 40};
int size = 5, target = 27;
bool b1 = hasTriplet(foo, size, target); // True, since 1+6+20 = 27

int foo1[3] = {1, 23, 3};
int size1 = 3, target1 = 22;
bool b2 = hasTriplet(foo1, size1, target1); // False. There are no
                                            // triplets whose sum is 22.
bool hasTriplet(int arr[], int size, int k) {
    for (int i = 0; i < size - 2; i++) {
        if (i == 0 || (i > 0 && arr[i] != arr[i-1])) {
            int lo = i+1, hi = size - 1, sum = k - arr[i];
            while (lo < hi) {
                if (arr[lo] + arr[hi] == sum)
                    return true;
                else if (arr[lo] + arr[hi] < sum)
                    lo++;
                else
                    hi--;
            }
        }
    }
    return false;
```

```
}
```