CS 31 Worksheet Week 5

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), 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;
}
```

2)
```
int mystery(const 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`

Programming Problems

1) Write a function with the following header:

```
bool equality(const 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 they are "equal" (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.

Example:

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

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

2) Write a function with the following header:

```
bool hasTwoSum(const 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.

Example:

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

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

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 in the array. You may assume that N will never be more than 100.

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

Example:

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

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

4) Write a function with the following header:

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

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

`zeroLeft` should alter `binarry` 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}
```

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 an array `arr`, and returns any element of `arr` 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 `arr` is unique, return 0.

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

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

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 b = hasTriplet(foo, size, target); // True: 1 + 6 + 20 = 27

int foo1[3] = {1, 3, 23};
int size1 = 3, target1 = 22;
bool b1 = hasTriplet(foo1, size1, target1); // False: No triplet
                                            // sums to 22.
```