Report

1. Obstacles

I found it hard to write a logic that realize a statement when several conditions are satisfied, especially when such conditions including several for-loops. However, I tried to use the mathematic thinking to deal with the main logic flow first. When the main route appears to be clear, I broke down the problems into several aims. As followed:

int divide(string a[], int n, string divider){

//put the str bigger or equal to devide to the behind.

…

//find the position of devider.

…

//put all the elements equal to devider before the ones greater than devider

    …

    return x;

}

Thus, I can realize each small function incrementally and make plan intentionally and systematically.

Besides, I also found it hard to test each cases due to the tremendous cases of possible difference. However, I created several arrays that can simultaneously be used by different functions involving situations of null array, empty strings, and so on. Thus, a simple system of testing functions is created.

1. Test data

[String cases]

string a[7] = { "greg", "gavin", "ed", "xavier", "john", "eleni", "fiona" };

string space[5] = {“ “, ”dianne”, ” ”, ”gavin”, ”john”};

string null[0] = {};

const string const[5] = {"gavin", "ed", "xavier", "john", "eleni"};

string double[4] = { "greg", "gavin", "ed", "greg"}

1. **int appendToAll (string a[], int n, string value)**
2. **correct tests**
3. normal append: (a, 5, ???)
4. append to array with space(space, 3, ab)
5. append to empty array: (null, 0, !!)
6. **incorrect tests**
7. constant string modified error: (const, 4, ???)
8. **int lookup (const string a[], int n, string target)**
9. normal lookup: (a, 6, john)
10. loopup empty string: (space, 4, )
11. lookup doesn’t exist in array: (a, 5, abc)
12. lookup empty array: (null, 0, !!)
13. lookup with more than two target existing in array: (double, 4, greg)
14. **int positionOfMax (const string a[], int n)**
15. **correct tests**
16. normal positionOfMax: (a, 5)
17. empty array: (null, 0)
18. only one element: (a, 1)
19. more than two max existing in array: (double, 4)
20. **int rotateLeft (string a[], int n, int pos)**
21. **correct tests**
22. normal rotate: (a, 5, 3)
23. rotate in null array: (null, 0)
24. rotate with space elements: (space, 5, 2)
25. **incorrect tests**
26. pos exceeding the range of n: (a, 4, 5)
27. constant string modified error: (const, 4, 2)
28. **int countRuns (const string a[], int n)**

**Arrays: string b[9] = { "xavier", "betty", "john", "john", "ed", "ed", "ed", "john", "john"};**

**string c[6] = { "xavier", "betty", "john", "john",” ”,” “}**

1. **correct tests**
2. normal count without consecutive identical items: (a, 6)
3. normal count with consecutive identical items: (b, 9)
4. normal count with consecutive space: (c, 6)
5. count with null: (null, 0)
6. count with identical but not consecutive items: (double, 4)
7. **int flip (string a[], int n)**
8. **correct tests**
9. flip array with even number of items: (a, 6)
10. flip array with odd number of items: (a, 5)
11. flip with null array : (null, 0)
12. flip with single item: (space, 1)
13. flip with negative number of item: (a, -3)
14. **incorrect test**
15. constant string modified error: (const, 4, 2)
16. **int differ (const string a1[], int n1, const string a2[], int n2)**

**Arrays: string x[6] = { "xavier", "betty", "john", "john",” ”,” “}**

**string y[6] = { "xavier", "betty", "johnny", "john",”Jack”,” “}**

1. **correct tests**
2. normal test: (x, 4, y, 4)
3. test with totally identical elements in two arrays: (x, 2, y, 2)
4. test with more than one difference in two arrays: (x, 5, y, 5)
5. test with different number of interesting elements: (x, 4, y, 6)
6. test with one null array: (null, 0, x, 2)
7. test with no identical elements at all: (a, 2, y, 3)
8. **incorrect tests**
9. test with negative number of interesting element: (a, -1, y, 2)
10. **int subsequence (const string a1[], int n1, const string a2[], int n2)**

**Arrays: string y[6] = { "xavier", "betty", "johnny", "john",”Jack”,” “}**

**string y1[5] = { “johnny", "john",”Jack”,” “, “Jack”}**

**string y2[1] = {“Jack"}**

1. **correct tests**
2. normal test: (y, 6, y1, 3)
3. test with partially identical elements in two arrays: (y, 6, y1, 5)
4. test with more than one difference in two arrays: (y, 5, y1, 5)
5. test with totally identical elements: (y, 4, y, 4)
6. test with one null array: (y1, 2, null, 0)
7. test with single identical element: (y, 6, y2, 1)
8. test with single identical element appears twice in a1[]: (y1, 5, y2, 1)
9. test with no identical elements at all: (a, 2, y, 3)
10. test with n2 bigger than n1: (y, 2, y, 3)
11. **int lookupAny (const string a1[], int n1, const string a2[], int n2)**
12. **correct tests**
13. normal test: (y, 6, y1, 1)
14. test with no identical elements in two arrays: (y, 6, a, 5)
15. test with more than one same item in two arrays: (y, 5, y1, 5)
16. test with totally identical elements: (y, 4, y, 4)
17. test with one null array: (y1, 2, null, 0)
18. **int divide (string a[], int n, string divider)**

**Arrays: string candidate[6] = { "dianne", "fiona", "gavin", "xavier", "ed", "betty" };**

1. **correct tests**
2. normal test with no identical item: (candidate, 6, eleni)
3. test with identical items: (x, 5 john)
4. test with item equals to devider: (candidate, 6, fiona)
5. test with space items: (space, 4, fiona)
6. test with space items: (space, 4, )
7. test with null array: (null, 0, eleni)
8. test with single item: (x, 1, gavin)
9. test with no item smaller than devider: (a, 6, /0)
10. test with no item bigger than devider: (a, 6, XXXXXX)
11. **incorrect tests**
12. constant string modified error: (const, 3, gavin)