Computer Science Project 4

# Overcoming Obstacles

I was beginning to face problems when dealing with the subsequence and the separate functions. At first, I couldn’t handle arrays which had incomplete subsequent elements. Hence I had to alert the function to false alarms, and continue checking for a complete subsequence. The separate function was also challenging to create, especially without additional arrays. I was finally able to create a function that efficiently sorted in the elements according to the specified condition. I tried not to sort in ascending order to avoid going through excess loops.

# Test Data

Legend: GREEN: Normal/Correct values; YELLOW: Boundary values; RED: Erroneous values

## appendToAll function

int appendToAll(string a[], int n, string value);

|  |  |  |  |
| --- | --- | --- | --- |
| No. | Test Case | Return Value | Changes to Array(s) |
| 1 | The value of ‘n’ is greater than 0, but smaller than the actual no. of elements present | n | n elements of the arrays are modified |
| 2 | Value of ‘n’ is zero | 0 | None |
| 3 | Value of ‘n’ is the total no. of elements present in the array | n | n elements of the arrays are modified |
| 4 | Value of ‘n’ is negative | -1 | None |
| 5 | The ‘value’ string in the function parameter is an empty string | n | None |

## lookup function

int lookup(const string a[], int n, string target);

Cases 1-4 assume the ‘target’ string is present in the array.

|  |  |  |  |
| --- | --- | --- | --- |
| No. | Test Case | Return Value | Changes to Array(s) |
| 1 | The value of ‘n’ is greater than 0, but smaller than the actual no. of elements present | position of 'target' | No change |
| 2 | Value of ‘n’ is zero | -1 | No change |
| 3 | Value of ‘n’ is the total no. of elements present in the array | position of 'target' | No change |
| 4 | Value of ‘n’ is negative | -1 | No change |
| 5 | The array has no element equal to the ‘target’ string | -1 | No change |

## positionOfMax function

int positionOfMax(const string a[], int n);

|  |  |  |  |
| --- | --- | --- | --- |
| No. | Test Case | Return Value | Changes to Array(s) |
| 1 | The value of ‘n’ is greater than 0, but smaller than the actual no. of elements present | position of max value | None |
| 2 | Value of ‘n’ is zero | -1 | None |
| 3 | Value of ‘n’ is the total no. of elements present in the array | position of max value | None |
| 4 | Value of ‘n’ is negative | -1 | None |

## rotateLeft function

int rotateLeft(string a[], int n, int pos);

Cases 1 and 3 assume the value of ‘pos’ < value of ‘n’, and that ‘pos’ >= 0

|  |  |  |  |
| --- | --- | --- | --- |
| No. | Test Case | Return Value | Changes to Array(s) |
| 1 | The value of ‘n’ is greater than 0, but smaller than the actual no. of elements present | pos | Eliminates element at position 'pos' and moves all elements after it one place to the left. Then places element at position 'pos' in the end |
| 2 | Value of ‘n’ is zero | -1 | None |
| 3 | Value of ‘n’ is the total no. of elements present in the array | pos | Eliminates element at position 'pos' and moves all elements after it one place to the left. Then places element at position 'pos' in the end |
| 4 | Value of ‘n’ is negative | -1 | None |
| 5 | The value of ‘pos’ is equal to or greater than ‘n’ | -1 | None |

## rotateRight function

int rotateRight(string a[], int n, int pos);

Cases 1 and 3 assume that pos < n, and that pos >= 0

|  |  |  |  |
| --- | --- | --- | --- |
| No. | Test Case | Return Value | Changes to Array(s) |
| 1 | The value of ‘n’ is greater than 0, but smaller than the actual no. of elements present | pos | Eliminates element at position 'pos' and moves all elements before it one place to the right. Then places element at position 'pos' at the starting |
| 2 | Value of ‘n’ is zero | -1 | None |
| 3 | Value of ‘n’ is the total no. of elements present in the array | pos | Eliminates element at position 'pos' and moves all elements before it one place to the right. Then places element at position 'pos' at the starting |
| 4 | Value of ‘n’ is negative | -1 | None |
| 5 | The value of ‘pos’ is equal to or greater than ‘n’ | -1 | None |

## flip function

int flip(string a[], int n);

|  |  |  |  |
| --- | --- | --- | --- |
| No. | Test Case | Return Value | Changes to Array(s) |
| 1 | The value of ‘n’ is greater than 0, but smaller than the actual no. of elements present | n | The elements in the array are reversed |
| 2 | Value of ‘n’ is zero | 0 | No change, as array is empty |
| 3 | Value of ‘n’ is the total no. of elements present in the array | n | The elements in the array are reversed |
| 4 | Value of ‘n’ is negative | -1 | No change |

## differ function

int differ(const string a1[], int n1, const string a2[], int n2);

Cases 1 and 3 assume that some equal strings are present in both arrays, while cases 5 and 6 assume that the value of ‘n’ is nonzero.

|  |  |  |  |
| --- | --- | --- | --- |
| No. | Test Case | Return Value | Changes to Array(s) |
| 1 | Both n1 and n2 is greater than 0, but smaller than the actual no. of elements present | position where the corresponding values in the arrays first differ | None |
| 2 | n1 and/or n2 equals 0 | 0 | None |
| 3 | n1 and/or n2 equals the total no. of elements present in the array | position where the corresponding values in the arrays first differ | None |
| 4 | n1 and/or n2 is negative | -1 | None |
| 5 | Both arrays have equal elements until one or both run out of elements | smaller of n1 and n2 | None |
| 6 | The arrays have completely different elements | 0 | None |

## subsequence function

int subsequence(const string a1[], int n1, const string a2[], int n2);

Cases 1 and 3 assume n1 is greater than or equal to n2, and that elements in a2 are subsequently present in a1.

|  |  |  |  |
| --- | --- | --- | --- |
| No. | Test Case | Return Value | Changes to Array(s) |
| 1 | Both n1 and n2 is greater than 0, but smaller than the actual no. of elements present | position in array a1 where the subsequence begins | None |
| 2 | n2 equals 0 | if n2 is zero, then function returns 0, whether n1 is zero or not | None |
| 3 | n1 and/or n2 equals the total no. of elements present in the array | position in array a1 where the subsequence begins | None |
| 4 | n1 and/or n2 is negative | -1 | None |
| 5 | The elements in array a2 are not subsequently present in a1 | -1 | None |

## lookupAny function

int lookupAny(const string a1[], int n1, const string a2[], int n2);

Cases 1 and 3 assume common elements are present in arrays a1 and a2.

|  |  |  |  |
| --- | --- | --- | --- |
| No. | Test Case | Return Value | Changes to Array(s) |
| 1 | Both n1 and n2 is greater than 0, but smaller than the actual no. of elements present | position of first element in a1 that is equal to an element in a2 | None |
| 2 | n1 and/or n2 equals 0 | -1 | None |
| 3 | n1 and/or n2 equals the total no. of elements present in the array | position of first element in a1 that is equal to an element in a2 | None |
| 4 | n1 and/or n2 is negative | -1 | None |
| 5 | No element is common in arrays a1 and a2 | -1 | None |

## separate function

int separate(string a[], int n, string separator);

|  |  |  |  |
| --- | --- | --- | --- |
| No. | Test Case | Return Value | Changes to Array(s) |
| 1 | The value of ‘n’ is greater than 0, but smaller than the actual no. of elements present | position of first element, after rearrangement, not < separator | all elements < separator are placed before all others, and all elements > separator are placed after all others |
| 2 | Value of ‘n’ is zero | 0 | None, as array is empty |
| 3 | Value of ‘n’ is the total no. of elements present in the array | position of first element, after rearrangement, not < separator | all elements < separator are placed before all others, and all elements > separator are placed after all others |
| 4 | Value of ‘n’ is negative | -1 | None |
| 5 | separator is greater than all strings in the array | n | None |
| 6 | separator is smaller than all strings in the array | 0 | None |