**Notable Obstacles:**

1. Initially I failed to figure out that I could compare two strings simply with a double equals sign. So while defining the functions *count* and *findFirst*, I used two if-else statements to compare both the size of each element in the string and the sequence of characters consisting each element. However, it’s totally unnecessary to do so.
2. The function subsequence is the most troublesome among all required functions when I dealt with string arrays. The first dilemma came from a choice. I was not sure whether I should check each element one by one in the array with more elements or in the array with fewer elements. Actually, it’s more efficient to check the shorter array. Once I have found the same element as a2[i] in a1 with index j, I would start to compare a2[i + 1] with elements with indexes larger than j. This easily avoids repetitive comparisons.
3. When I defined the function *takeSides*, I failed to correctly handle the problem that several elements in the array might equal to barrier. If so, I could not merely move all elements larger than barrier to the end of array. Therefore, I first moved all elements that were the same as barrier to the end. After that, I moved elements larger than barrier to the end. Thus, I not only gathered all barriers, but also put them in the right positions in the array.

**Test Cases:**

1. **string a[10] = { “James”, “Durant”, “Durant”, “Durant”, “Paul”, “Howard”, “Howard”, “Curry”, “Paul”, “Kobe” }**
2. assert(count(a, -1, “Kobe”) == -1); //invalid n
3. assert(count(a, 2, “Durant”) == 1); //normal
4. assert(count(a, 10, “Paul”) == 2);
5. assert(count(a, 10, “Chris”) == 0); //target does not appear in the string
6. assert(count(a, 10, “Durant”) == 3);
7. assert(count(a, 10, “DURANT”) == 0); //target with all letters of an element in the string capitalized
8. assert(findFirst(a, -7, “James”) == -1); //invalid n
9. assert(findFirst(a, 5, “Paul”) == 4); //normal
10. assert(findFirst(a, 10, “Howard”) == 5);
11. assert(findFirst(a, 10, “PAUL”) == -1); //target with all letters of an element in the string capitalized
12. assert(moveToEnd(a, -1, 5) == -1); //invalid n
13. assert(moveToEnd(a, 10, 4) == 4 && a[4] == “Howard” && a[5] == “Howard”); //normal
14. assert(moveToEnd(a, 10, 3) == 3 && a[1] == “James” && a[3] == “Paul”);
15. assert(moveToEnd(a, 10, 0) == 1 && a[0] == “Durant” && a[9] == “James”);
16. assert(removeCopies(a, 10) == 7 && a[2] == “Paul” && a[4] == “Curry”);
17. assert(removeCopies(a, -5) == -1); //invalid n
18. assert(takeSides(a, 10, “Durant”) == 1 && a[7] > “Durant”); //normal
19. assert(takeSides(a, -6, “Paul”) == -1); //invalid n
20. assert(takeSides(a, 10, “ James”) == 6); //normal
21. assert(takeSides(a, 10, “Paul”) == 8);

**string b[3] = { “James”, “Durant”, “Durant”};**

1. assert(firstInequality(a, -1, b, 3) == -1); //invalid n1
2. assert(firstInequality(a, 10, b, -1) == -1); //invalid n2
3. assert(firstInequality(a, 2, b, 2) == 2); //normal
4. assert(firstInequality(a, 3, b, 2) == 2);
5. assert(firstInequality(a, 10, b, 3) == 3);
6. assert(firstInequality(b, 2, a, 1) == 1); //substitute b and a
7. assert(firstInequality(b, 3, a, 10) == 3); //normal
8. assert(subsequence(a, 7, b, 3)); //normal
9. assert(!subsequence(b, 3, a, 5)); //n2 is larger than n1
10. assert(!subsequence(a, -1, b, 3)); //invalid n1
11. assert(!subsequence(a, 10, b, -7)); //invalid n2

**string c[5] = { “James”, “Durant”, “Paul”, “Howard”, “Kobe”};**

1. assert(firstInequality(a, 10, c, 5) == 2); //normal
2. assert(firstInequality(a, 5, c, 5) == 2);
3. assert(firstInequality(a, 10, c, 3) == 2);
4. assert(subsequence(a, 10, c, 5));
5. assert(!subsequence(c, 5, b, 3));

**I do not create new array in defining functions *moveToEnd*, *firstInequality*, *removeCopies,* and *takeSides***