CS 31 Worksheet Week 6

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: 2D Arrays, C strings, 1D arrays, Pass by value/reference

Reading Problems

1)

a) What does the following program print out?

```
#include <iostream>

using namespace std;

int main () {

      char phrase[] = "How the turntables.";

      for (int i = 0; i < strlen(phrase); i++) { // change in part b

            phrase[strlen(phrase) - 1] = '\0';

      }

      cout << "Result: " << phrase << endl;

}
```

b) Repeat part a), but replace the for loop inside main() to the following code:

```
int n = strlen(phrase);

for (int i = 0; i < n; i++) { // change in part b

      phrase[strlen(phrase) - 1] = '\0';
}
```

Programming Problems

1) Write a function with the following header:

```
void invert(int matrix[][N], int n);
```

where :

> **`matrix`** is a 2-dimensional array of integers of size N x N. In this header, N is to be replaced by a number chosen by the programmer (you).
>
> **`n`** is the value N (passed in so that invert knows how big **`matrix`** is)
>
> **`invert`** should reflect **`matrix`** across the negative-sloping diagonal, so that the rows become the columns and vice versa.

Example:

```
/* The second [] in a 2D array passed as a parameter requires a
number as the size, which restricts the implementation of invert to
be able to work on only one matrix size. The following example works
only on 2D arrays of size 3x3. For invert's declaration, the N in the
function header has been replaced by 3. */

void invert(int matrix[][3], int n) {
      // Implementation goes here...
}


int main() {

      int foobar[3][3] = {{1, 2, 3},{4, 5, 6}, {7, 8, 9}};

      invert(foobar, 3);

/* foobar is now expected to be:

{{1, 4, 7}, {2, 5, 8}, {3, 6, 9}} */

}
```

2) Write a function *charInsert* that inserts a character into a valid C-string at a given position. The function has the following header:

```
bool charInsert(char str[], int n, int ind, char c)
```

The parameter **n** denotes the size of the character array **str**, which is not necessarily equivalent to the string's length. **ind** refers to the index at which the insertion will be done, so if **ind** is 0 then the **char c** will be the first character in the new string. The insertion cannot be performed if ind is negative or greater than the string's length. Additionally, the insertion cannot be performed if the result would exceed the length of the array.

If the insertion is successful, the function returns true. If the insertion cannot be done, the function returns false and leaves **str** unmodified.

Examples:

```
char success[10] = "aaaaa";

bool res = charInsert(success, 10, 1, 'b'); // res should equal true cout
<< success << endl; // abaaaa

char success[10] = "aaaaa";

bool res = charInsert(success, 10, 5, 'b'); // res should equal true cout
<< success << endl; // aaaaab

char failure[6] = "aaaaa";

bool res = charInsert(failure, 6, 1, 'b'); // res should equal false cout
<< failure << endl; // aaaaa
```

3) Write a function wordShiftLeft that takes in a valid C-string and rotates each word left one character. A word is defined as a substring separated by spaces. Each rotated word wraps around, meaning that "CS31" would become "S31C". The function has the following header:

```
void wordShiftLeft(char str[])
```

Example:

```
char test[] = "I love CS31"; wordShiftLeft(test);

cout << test << endl; // "I ovel S31C"



char test[] = "I.love.CS31"; wordShiftLeft(test);

cout << test << endl; // ".love.CS31I"
```

4) Write a function with the following header:

```
bool rangeSearch(int sorted_nums[], int n, int target,
                 int& start, int& end);
```

`sorted_nums` is an array of integers sorted in increasing order
`n` is the number of elements in `sorted_nums`
`target` is a number to search for within `sorted_nums`

`rangeSearch` should return *true* if `target` is found in `sorted_nums` and *false* otherwise.

If `rangeSearch` returns *true*, `start` should be set to the first index where `target` appears and `end` should be set to the last index where `target` appears.

If `rangeSearch` returns *false*, `start` and `end` should not be altered.

Example:

```
int foo[7] = {-3, -2, 1, 3, 3, 4, 5};
int s = 21;
int e = 14;
rangeSearch(foo, 7, 3, s, e); // returns true, now s == 3 and e == 4
rangeSearch(foo, 7, 0, s, e); // returns false, s is still 21, e ==
14
```