CS 31 Worksheet 4

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

C-Strings, Passing 2D arrays as parameters, Arrays of C-Strings

1) Write a function with the following header:

```
bool insert(char str[], int max, int ind, char c)
```

This function should insert *c* into *str* of length *max* (specifying the maximum number of elements that can be stored in the array containing the C string) at the index specified by *ind*, resulting in a C string that is one character longer than it was before. If this insertion is successful, the function returns true. If the insertion cannot be done, the function returns false and leaves *str* undone.

Example:

```
char test[20] = "aaaaaaa";
bool res = insert(test, 20, 1, 'b');
// res should now store true and test should now store
"abaaaaaa"

char test[20];
strcpy(test, "abcdefghijklmnopqrs");
bool res - insert(test, 20, 10, 'X');
// res should be false and test is unchanged
```

2) Write a function with the following header:

```
void eraseChar(char str[], char c)
```

This function should erase all instances of *c* from *str*.

Example:

```
char test[4] = "acc";
```

```
eraseChar(test, 'c');
//test should now store "a"
```

3) Implement strcat which allows you to concatenate two c-strings. Assume there is enough space to save the entire result into str1.
   `void strcat(char str1[], char str2[])`

   Example:
   ```
   char str1[20] = "Hello";
   char str2[8] = " World!";
   strcat(str1, str2);
   cout << str1; // Hello World!
   ```

4) Write a function with the following header:

   `void eraseDuplicates(char str[])`

   This function should erase all duplicated characters in the string, so that only the first copy of any character is preserved. Feel free to use helper functions.

   Example:
   ```
   char test[50] = "memesformeforfree123";
   eraseDuplicates(test);
   //test should now store "mesfor123"
   ```

5) Write a function and removes *all* of the instances of a **word** from a **sentence.** Both a sentence and word are defined as C-strings.

   Function header: `void remove(char sentence[], const char word[])`
   Example:
   ```
   const char word[4] = "ask";
   char sentence[50] = "I asked her to ask him about the task";
   remove(sentence, word);
   // sentence should now be "I asked her to him about the task";
   // sentence should not be "I ed her to him about the t";
   // Note: sentence does not end with a period
   ```

6) Write a function that scores a sentence based on the weight for words. Given a c-string sentence, an int array of weights and a string array of words corresponding to the weights. Each instance of the word should be accounted for in the score (i.e. if the word "Smallberg" appears in the sentence twice and has a weight of -50, it should contribute -100 to the score.) Every word does not have to have a weight.

Function header:
```
int score(char sentence[], string words[], int weights[])
```

Example:
```
char sentence[50] = "I love to love computer science";
string words[3] {"love", "computer", "science"};
int weights[3] = {10, 5, 2};
score(sentence, words, weights) → 27

char sentence[70] = "Take computer science 31 with Smallberg";
string words[3] = {"Smallberg", "computer", "science"};
int weights[5] = {-50, 5, 2};
score(sentence, words, weights) → -43
```

7) Write a function that takes a C-string and determines whether that string has balanced parentheses "()". To be balanced, each "(" should come **before** its corresponding ")".

```
bool balancedParens(char str[])
```
Example:
```
balancedParens("()(faker)((dont cry))") == true;
balancedParens("open_paren())") == false;
balancedParens("ab ) cd ( ef") == false;
```

The following problems are extra **challenge** problems. They do not represent content that you will be tested on, but stretch beyond CS31 to include 32 material.

8) Write a function that takes a C-string and determines whether that string has balanced parentheses "()" AND brackets "[]". Note that nesting like this "[(])" is not allowed.

```
bool balancedParensBrackets(char str[])
```
Example:
```
balancedParensBrackets("[(ssg)([]new)((dynasty))]") == true;
balancedParensBrackets("[(])") == false;
```

9) Implement the following function that takes in a 2D character array, (n x 5), and determines if there's a valid path from the starting position to the end position. Note that you may only move horizontally and vertically, not diagonally. "x" represents a wall, and "o" represents a vacant spot.

```
bool hasValidPath(char grid[][5], int totalRows, int sRow, int
sCol, int eRow, int eCol)
```

Example: Red represents the starting position, green the ending position

```
      01234
0     xoxxx
1     xoxoo
2     xoxox
3     ooxxx
```

1) sRow = 3, sCol = 0, eRow = 0, eCol = 1 returns True

```
      01234
0     xoxxx
1     xoxoo
2     xoxox
3     ooxxx
```

2) sRow = 1, sCol = 4, eRow = 1, eCol = 1 returns False