Dept. of Computer Science, University of California, Davis
ECS30 Instructor: Rob Gysel 1/30/17

Programming Assignment #4*
Due date: 2/12/17 11:59pm

Programs are to be submitted to Gradescope by the due date. You may work alone or in groups of two. Programs submitted up to 24 hours late will still be accepted but incur a 10% grade penalty. Uploading your programs to gradescope will immediately score your submission. Your program grade will be the score of the *last* submission that you have uploaded. Programs must compile using `gcc -Wall` without any warnings. Each program that compiles with a warning will incur a 10% grade penalty. **Each program will have 20 seconds to compile and run test cases on gradescope.**

In this assignment, you will download `Program4Source.zip` and complete the code fragments as described below.

Problem 1: `simpleyahtzee.c` (50 points, 5 per test case)

Simple yahtzee consists of 6 rounds. In each round, a player rolls 5 dice with up to two re-rolls. For each re-roll, the player chooses which dice to save and which dice to re-roll. At the end of the round, the player picks a category between 1 and 6 to save their score to. Each die that matches the category adds to the score of that category. Each category is picked and scored exactly once. For example, if a round ends with 1 6 3 4 6 and the player chooses category 6, the score for category 6 is 6 + 6 = 12. If the player had chosen category 3, the score would be 3. If the player had chosen category 5, the score would be 0.

This program has been partially written for you in `simpleyahtzee.c`. Write the body of functions that are marked with a comment that begins with

```
// Homework TODO: ...
```

Do not modify other parts of the code.

*Last updated January 31, 2017

Example output:

```
[rsgysel@pc17 \~]\$ ./simpleyahtzee
Enter seed: 10
Dice: 5 4 2 5 6
Rolls left: 2
Enter dice to save or 'r' for reroll: 1
Die 1 is now saved
Enter dice to save or 'r' for reroll: a
Error: invalid command. Enter 'r' for re-roll unsaved die or 1-5 to toggle saving die
Enter dice to save or 'r' for reroll: 0
Error: invalid command. Enter 'r' for re-roll unsaved die or 1-5 to toggle saving die
Enter dice to save or 'r' for reroll: 4
Die 4 is now saved
Enter dice to save or 'r' for reroll: r
Dice: 5 1 1 5 3
Rolls left: 1
Enter dice to save or 'r' for reroll: r
Dice: 5 3 2 5 4
Rolls left: 0
Enter category to save score: 5
Category 1 score: not scored
Category 2 score: not scored
Category 3 score: not scored
Category 4 score: not scored
Category 5 score: 10
Category 6 score: not scored
Total Score: 10
Dice: 4 5 3 1 4
Rolls left: 2
Enter dice to save or 'r' for reroll: 1
Die 1 is now saved
Enter dice to save or 'r' for reroll: 5
Die 5 is now saved
Enter dice to save or 'r' for reroll: r
Dice: 4 5 5 2 4
Rolls left: 1

...

Enter dice to save or 'r' for reroll: 5
Die 5 is now saved
Enter dice to save or 'r' for reroll: r
Dice: 1 6 5 6 6
```

```
Rolls left: 1
Enter dice to save or 'r' for reroll: a
Error: invalid command. Enter 'r' for re-roll unsaved die or 1-5 to toggle saving die
Enter dice to save or 'r' for reroll: r
Dice: 1 6 4 6 6
Rolls left: 0
Enter category to save score: 5
Error: invalid command. Enter 1-6 to save to an unused category
Enter category to save score: a
Error: invalid command. Enter 1-6 to save to an unused category
Enter category to save score: 6
Category 1 score: 2
Category 2 score: 8
Category 3 score: 12
Category 4 score: 8
Category 5 score: 10
Category 6 score: 18
Total Score: 58
Game over.
```

Problem 2: `testmystringfunctions.c` (50 points, 2 per test case, 18 test cases), `gccUsage.txt` (14 pts)

Imagine you are working on a team and one of your team members is writing a library of C functions to work with strings. They decide to name their library `mystringfunctions`, and have two files: a *source file* `mystringfunctions.c` and a *header file* `mystringfunctions.h`.

In this problem, you will write automated tests known as *unit tests*. This program has been partially written for you in `testmystringfunctions.c`. Write the body of functions that are marked with a comment that begins with

```
// Homework TODO: ...
```

Do not modify other parts of the code.

Each unit test must use assert exactly once (in addition to other code). You will need to compile `mystringfunctions.c` into an object file called `mystringfunctions.o` and link it with your executable, which you will call `testmystringfunctions`. The `gcc` commands you use to create the object file `mystringfunctions.o` and executable file `testmystringfunctions`. Turn in a file called `gccUsage.txt` that contains exactly the following two lines:

1. a gcc command to create the object file mystringfunctions.o
2. a gcc command to create the executable testmystringfunctions

Example output, assuming the code from `mystringfunctions` is **correct**:

```
[rsgysel@pc17 \~]\$ ./testmystringfunctions
Which unit test would you like to run?
1) deepCopyStr
a) n = 2, src = "test string"
b) n = 0 returns "\0"
c) negative n returns NULL
2) isLowerOrDigitStr
a) n = 4, src = "testString"
b) n = 5, src = "testString"
c) n = 0
3) concatStrs
a) n = 5, str1 = "test", str2 = "string"
b) n = 5, str1 = "", str2 = "test string" returns "test "
c) n = 5, str1 = "test", str2 = "" returns NULL
pf
Enter 1, 2, or 3 for the function to test.
p
f
Enter 1, 2, or 3 for the function to test.
1
```

```
1
Enter a, b, or c for the test case.
1a
Test successful.
```

Example output, assuming the code from `mystringfunctions` is **incorrect**:

```
[rsgysel@pc17 \~]\$ ./testmystringfunctions
Which unit test would you like to run?
1) deepCopyStr
a) n = 2, src = "test string"
b) n = 0 returns "\0"
c) negative n returns NULL
2) isLowerOrDigitStr
a) n = 4, src = "testString"
b) n = 5, src = "testString"
c) n = 0
3) concatStrs
a) n = 5, str1 = "test", str2 = "string"
b) n = 5, str1 = "", str2 = "test string" returns "test "
c) n = 5, str1 = "test", str2 = "" returns NULL
1a
Assertion failed: (result && result[0] == 't' && result[1] == 'e' && result[2] == '\0'), f
Abort trap: 6
```