Spring, 2019 Name: ______________________

(Please *don't* write your id number!)

Exam 3: C Programming with Conditionals and Loops

Directions

This exam is closed book and notes.
If you need more space, use the back of a page. Note when you do that on the front.
Before you begin, please take a moment to look over the entire test so that you can budget your time.
Clarity is important; if your answers are sloppy and hard to read, you may lose some points.

Some C Functions You May Find Helpful

The function specifications given below may be helpful in solving some problems. Note that you do *not* need to write any of these functions for this exam, but can assume that they are provided to you.

Locally-Provided Function

Some tests use the `approx` function specified in the header file `approx.h` shown below.

```
#include <stdbool.h>
// Requires: neither argument is NaN
// Ensures: result is true just when the absolute value of actual - expected is less than epsilon.
extern bool approx(double actual, double expected, double epsilon);
```

Standard C Library Functions

The `toupper` function is declared in the standard header file `ctype.h`. The `toupper` function takes a character and returns it unchanged, unless the argument is a lowercase letter, which is then returned as the corresponding uppercase letter. (Note that a **`char`** can be passed to this function, and its result can be interpreted as a **`char`**.)

```
extern int toupper (int c);
```

The `fgets` function is declared in the standard header file `stdio.h`. The `fgets` function takes a char array, `str`, a maximum number of characters to read, `num`, and a pointer to a `FILE` to read from, such as `stdin`, called `stream`. It reads in a line of text (from `stream`) into `str`, assuming that the text input is no more than the number of characters specified, including the trailing newline (`'\n'`) character. A null character (`'\0'`) is added to the end of the input in `str`. The return value is `str` when the function is successful.

```
extern char *fgets (char str[], int num, FILE *stream);
```

For Grading

Question:	1	2	3	4	Total
Points:	15	20	30	35	100
Score:					

1. (15 points) [Programming] In C, define the function specified in the header file `carryOnOk.h`, shown below:

   ```
   #include <stdbool.h>
   // maximum dimensions in inches
   #define MAX_DEPTH 9
   #define MAX_WIDTH 14
   #define MAX_HEIGHT 22
   // requires: none of depth, width, or height is NaN
   // ensures: result is true just when depth <= MAX_DEPTH,
   //          and width <= MAX_WIDTH, and height <= MAX_HEIGHT;
   //          returns false otherwise.
   extern bool carryOnOk(double depth, double width, double height);
   ```

   that takes three doubles, representing a bag's measurements in inches, and returns true just when these dimensions are no larger than that permitted by the airline for carry on luggage. Examples are contained in the following test code.

   ```
   #include <stdlib.h>
   #include "tap.h"
   #include "carryOnOk.h"
   int main() {
       // testing calls that should return true
       ok(carryOnOk(MAX_DEPTH, MAX_WIDTH, MAX_HEIGHT));
       ok(carryOnOk(5.0, 10.0, 20.0));
       ok(carryOnOk(0.5, 6.0, 15.0));
       ok(carryOnOk(0.01, 14.0, 22.0));
       // testing calls that should return false
       ok(!carryOnOk(MAX_DEPTH+1.0, MAX_WIDTH, MAX_HEIGHT));
       ok(!carryOnOk(MAX_DEPTH, MAX_WIDTH+0.5, MAX_HEIGHT));
       ok(!carryOnOk(MAX_DEPTH, MAX_WIDTH, MAX_HEIGHT+0.25));
       ok(!carryOnOk(100.0, 100.0, 121.345));
       return exit_status();
   }
   ```

2. (20 points) [Programming] In C, define the function specified in the header file `dot_product.h`, shown below:

```
// requires: both a and b have sz elements and no element is NaN.
// ensures: result is the dot product of a and b, that is
//          result is a[0]*b[0] + a[1]*b[1] + ... + a[sz-1]*b[sz-1].
extern double dot_product(const double a[], const double b[], int sz);
```

that takes two arrays of doubles, `a` and `b`, of the same size, `sz`, and returns their dot product, which is the sum $\Sigma_{i=0}^{\mathtt{sz}-1}\mathtt{a[i]} \cdot \mathtt{b[i]}$. Tests for this problem appear below.

```
#include "tap.h"
#include "approx.h"
#include "dot_product.h"
int main() {
    double eps = 0.001; // tolerance for testing
    double one[1] = {1.0};  // so one[0] == 1.0
    double three[1] = {3.0};
    ok(approx(dot_product(one, one, 1), 1.0, eps));
    ok(approx(dot_product(three, one, 1), 3.0, eps));
    ok(approx(dot_product(three, three, 1), 9.0, eps));
    double ones[5] = {1.0, 1.0, 1.0, 1.0, 1.0};
    double a5[5] = {0.5, 1.5, 2.5, 3.5, 4.5};
    ok(approx(dot_product(ones, ones, 5), 5.0, eps));
    ok(approx(dot_product(a5, ones, 5), 12.5, eps));
    ok(approx(dot_product(a5, a5, 5), 41.25, eps));
    double a10[10] = {1.0, 0.5, 0.25, 0.125, 0.0625, 0.03125,
                      2.0, 4.0, 8.0, 16.0};
    double b10[10] = {3.14, 2.78, 1.414, 0.2857, 0.42857, 0.57142,
                      3.0, 5.0, 7.0, 11.0};
    ok(approx(dot_product(a10, b10, 10), 262.963855, 0.01));
    return exit_status();
}
```

(The tests above use the function `approx`, which is specified on the first page of this exam.)

3. (30 points) [Programming] In C, define the function specified in the header file `ci_streq.h`, shown below:

```c
#include <stdbool.h>
// requires: both s and s2 are null-terminated ASCII strings.
// ensures: result is true just when s and s2 have the same characters in the same order,
//          regardless of the case of the characters.
extern bool ci_streq(const char s[], const char s2[]);
```

that takes two strings, `s` and `s2`, and true just when they have the same characters in the same order, regardless of case. Your solution can use C library functions, if you include the appropriate header file.

Hint: you may find it helpful to use the function `toupper`, as described on the first page of this exam.

Examples for the function you are to write, `ci_streq`, appear below.

```c
#include "tap.h"
#include "ci_streq.h"
int main() {
    ok(ci_streq("",""));
    ok(ci_streq("A","a"));
    ok(ci_streq("ABBARA CADABRA!", "abbara cadabra!"));
    ok(ci_streq("abbara cadabra!", "ABBARA CADABRA!"));
    ok(ci_streq("Ada Lovelace","aDa loveLace"));
    ok(ci_streq("Old MacDonald Had a Farm, E-I-E-I-O!",
                "old macdonald haD A farM, e-I-e-i-o!"));
    // tests showing a return value of false below
    ok(!ci_streq("b","c"));
    ok(!ci_streq("RETURNS FALSE","see?"));
    ok(!ci_streq("","NOT the same"));
    ok(!ci_streq("1234567","7654321"));
    ok(!ci_streq("BEWARE OF","going off the end!"));
    return exit_status();
}
```

4. (35 points) In C, define a program that prompts, on standard output, with “`message? `” and then reads a line of text, consisting of at most 36 characters, from standard input. This line is then printed on standard output with every character (including spaces) followed by an added space and every lowercase character converted to the corresponding uppercase character. The entire output is followed by a single newline character. The program should then return a success code.

   A first example interaction, with the user’s input following the “`?␣`” is as follows. Note that spaces are show explicitly as “`␣`” in these examples.

   ```
   message?␣Welcome␣to␣the␣Future!
   W␣E␣L␣C␣O␣M␣E␣␣␣T␣O␣␣␣T␣H␣E␣␣␣F␣U␣T␣U␣R␣E␣!␣
   ```

   A second interaction with the program:

   ```
   message?␣UCF␣is␣the␣best!
   U␣C␣F␣␣␣I␣S␣␣␣T␣H␣E␣␣␣B␣E␣S␣T␣!␣
   ```

   A third interaction with the program:

   ```
   message?␣Friday␣is␣a␣good␣day
   F␣R␣I␣D␣A␣Y␣␣␣I␣S␣␣␣A␣␣␣G␣O␣O␣D␣␣␣D␣A␣Y␣
   ```

   Hint: You may find it helpful to use the functions `toupper`, and `fgets`, which are described on the first page of this exam.

```
//more space for your solution
```