Name: ________________________________

CSC 101 Final Exam

Instructions: *Answer questions 1 through 8 on the paper provided. Answer the remaining questions by using `handin` on `unix1`, submitting the files indicated in each question. You have up to 3 hours to complete the exam. The test is open book and open note. It is worth a total of 140 points.*

For each of the programming questions, you will be given information about any support files you need to copy, plus instructions for how to run your program and submit it via `handin`.

Important test-taking strategy: *You will get partial credit for programs that don't produce correct output. If you can't get correct output for some questions, move on to another question and write some code for it. After you have at least some code for all of the questions, you can go back to other questions and do debugging, if there's time.*

1. (3 points) Given the following declaration,

```
int dailyTemperatures[365];
```

which of the following sets the last element of `dailyTemperatures` to 56? (circle all that apply)

a. `dailyTemperatures[364] = 56;`

b. `dailyTemperatures[365] = 56;`

c. `dailyTemperatures[366] = 56;`

d. `56 = dailyTemperatures[364];`

e. `56 = dailyTemperatures[365];`

f. `56 = dailyTemperatures[366];`

2. (3 points) An array can contain which of the following types of data? (circle all that apply)

a. boolean

b. string

c. array

d. struct

e. pointer

3. (3 points) Which of the following is NOT a valid function prototype? (circle all that apply)

a. `double getAvgDonutsEaten(double donutsEaten[]);`

b. `int* getPositiveValues(int* values);`

c. `void findSmallest(int values[][25], int &i, int &j);`

d. `void writePrimes(int largest);`

e. `char** copy_deck(char** deck);`

4. (3 points) How many times is the `printf` executed in the following nested loop?

```
int i,j;
for (i = 0; i < 10; i++) {
    for (j = 0; j < 10; j++) {
        printf("%d", i+j);
    }
}
```

a. 10 times

b. 11 times

c. 20 times

d. 100 times

e. *none of the above*

5. (3 points) What does the following code print?

```
int x = 10, y = 20;
if (x = y) printf("equal");
else if (x < y) printf("less");
else printf("greater");
```

a. `equal`

b. `less`

c. `greater`

d. *no output at all, i.e., it doesn't print anything*

6. (3 points) Which of the following statements makes the most sense before an end-of-file controlled `while` loop?

a. `scanf("%d", &num);`

b. `scanf("%d", &num, result);`

c. `scanf("EOF");`

d. `result = scanf("%d", &num);`

e. *none of the above; the first scanf goes inside the loop*

7. (3 points) Which of the following boolean expressions will determine if `x` is between 0 and 10 inclusive? (Circle all that apply.)

   a. `0 <= x <= 10`

   b. `x > -1 || x < 11`

   c. `x != 0 && x < 10`

   d. `0 <= x && x <= 10`

   e. `(x < 11) && (x > -1)`

8. (6 points total) Here's a simple program that reads a word from `stdin` and stores it in a string variable. A word is defined as any sequence of characters without whitespace.

```
#include <stdio.h>
#define LENGTH 10
int main() {
    char word[LENGTH];

    /* Prompt for and read a word. */
    printf("Input a word: ");
    scanf("%s", word);

    /* Print out the word to confirm proper input. */
    printf("The word is: %s \n", word);

    return 0;
}
```

There's a problem with this program when it's run and the user enters the word `incorrectly-implemented`. (This is one word, since dashes aren't considered whitespace.)

a. (3 points) Briefly describe the problem.

b. (3 points) Describe the simplest way to fix the problem, so that the program behaves properly when the user inputs the word `incorrectly-implemented`. "Behaves properly" means the program reads in the full word, prints it out as entered, and doesn't have the problem you described in part (a).

9. (20 points) Write a function named `ball_weight` that calculates the weight of a hollow ball. The function takes three parameters: (1) the radius of the ball, to the outside surface; (2) the thickness of the material the ball is made of; (3) the density of the material the ball is made of.

   Consider, for example, a typical basketball that has a radius of 4.695 inches, is made of material that is 0.125 inches thick, with a material density of 0.037 pounds per cubic inch. The function calculates the weight of this ball as `1.247` pounds.

   For the sake of simplicity, the function ignores the weight of any gas that fills the ball. Therefore the weight of a ball is the product of the amount of material the ball is made of and the density of that material. ***Hint:*** To compute the volume of material the ball is made of, calculate the volume of the outer sphere and subtract the volume of the inner (empty) sphere.

   When writing `ball_weight`, you may assume the following:

   - The formula for the volume of a sphere is `4/3.0 * M_PI * pow(radius,3)`
   - The constant `M_PI` and the power function `pow` are pre-defined in `<math.h>`.
   - The radius, thickness, and density are given in consistent units, so no unit conversion is required. For example, if the radius is in inches, the thickness is also in inches, and the density is in pounds per cubic inch.
   - All inputs are reasonable, so no error checking is required; in particular no input is zero or negative.

   Support file: There is one support file for this question, named `ball.c`. Copy it with the UNIX command

   ```
   cp ~gfisher/classes/101/final/ball.c .
   ```

   Where to put your answer: Add the definition of the `ball_weight` function into `ball.c`, after the comment `/* Define ball_weight function here: */`

   How to compile and test: Compile with the command

   ```
   gcc ball.c -lm -o ball
   ```

   Note that the normal gcc flags (`-ansi`, etc) are ***intentionally omitted*** here. To test your implementation, run with the command

   ```
   ./ball
   ```

   and compare your output with the output listed after "`should be`". Your specific output for this question should look like this:

   ```
   Weight of standard basketball, should be 1.247: 1.247
   Weight of standard soccer ball, should be 0.942: 0.942
   Weight of standard tennis ball, should be 0.123: 0.123
   ```

   How to hand in:

   ```
   handin gfisher 101_final ball.c
   ```

10. (20 points) Write a function named `contains_prefix` that takes two string parameters. It returns true if the first string starts with the second string (the prefix). It returns false otherwise. For example,

```
contains_prefix("ubermensch", "uber")
```

returns true, whereas

```
contains_prefix("ubermensch", "uebr")
```

returns false. Recall that by convention, true is the integer value 1, false is the value 0.

The cases where one or both inputs is the empty string are handled according to the following table:

first string	**second string**	**return value**
empty	non-empty	0
non-empty	empty	1
empty	empty	1

Support file: There is one support file for this question, named `prefix.c`. Copy it with the UNIX command

```
cp ~gfisher/classes/101/final/prefix.c .
```

Where to put your answer: Add the definition of the `contains_prefix` function into `prefix.c`, after the comment `/* Define contains_prefix function here: */`.

How to compile and test: Compile with the command

```
gcc prefix.c -o prefix
```

To test your implementation, run with the command

```
./prefix
```

and compare your output with the output listed after "`should be`". Your specific output for this question should look like this:

```
contains_prefix("ubermensch", "uber"), should be 1: 1
contains_prefix("ubermensch", "uebr"), should be 0: 0
contains_prefix("ubermensch", "uberr"), should be 0: 0
contains_prefix("", "uber"), should be 0: 0
contains_prefix("ubermensch", ""), should be 1: 1
contains_prefix("", ""), should be 1: 1
```

How to hand in:

```
handin gfisher 101_final prefix.c
```

11. (25 points) Write a function named `repeat_sequences`, that returns the number of sequences of repeating numbers in an array of values. The parameters are an array of integer values and the size of the array. The return type is `int`.

    To be considered a repeating sequence, the numbers must be identical neighboring values in the array. The following are examples of several arrays along with the expected result of the function for each. Note that your function must work for `int` arrays of any size, not just those shown here.

Array Value	Return Value
`int a0[] = {}`	0
`int a1[] = {1}`	0
`int a2[] = {1, 2, 3, 4, 5}`	0
`int a3[] = {3, 2, 4, 5, 4, 2, 3}`	0
`int a4[] = {1, 1, 2, 3, 4, 5, 6}`	1
`int a5[] = {1, 2, 3, 3, 3, 2, 1}`	1
`int a6[] = {1, 1, 1, 1, 1, 1, 1}`	1
`int a7[] = {1, 1, 3, 4, 1, 1, 6}`	2
`int a8[] = {1, 2, 2, 1, 1, 1, 6}`	2
`int a9[] = {1, 1, 2, 2, 2, 4, 4}`	3

Support file: There is one support file for this question, `sequences.c`. Copy it with the UNIX command

```
cp ~gfisher/classes/101/final/sequences.c .
```

Where to put your answer: Add the definition of the `repeat_sequences` function into `sequences.c`, after the comment `/* Define repeat_sequences function here: */`.

How to compile and test: Compile with the command

```
gcc sequences.c -o sequences
```

To test your implementation, run with the command

```
./sequences
```

and compare your output with the output listed after "`should be`". Your specific output for this question should look like this:

```
Output for a0 should be 0: 0
Output for a1 should be 0: 0
Output for a2 should be 0: 0
Output for a3 should be 0: 0
Output for a4 should be 1: 1
Output for a5 should be 1: 1
Output for a6 should be 1: 1
Output for a7 should be 2: 2
Output for a8 should be 2: 2
Output for a9 should be 3: 3
```

How to hand in:

```
handin gfisher 101_final sequences.c
```

12. (15 points) Magicians who do card tricks often have someone choose a random card from a deck of cards. For this question, you are writing a function named `choose_random_card` that performs a random card selection from a sorted deck of cards. The function takes one input parameter that is the sorted deck. It prints output to `stdout`. Its return type is `void`, since it performs all its output by printing.

    The `choose_random_card` function starts by generating a random number between 1 and 52, inclusive. The function then prints this number to `stdout`. On a separate line, the function then prints the value of the card at that random position in the sorted deck. Note that the number is printed to the program user as a value between 1 and 52, even though a C deck array goes from positions 0 to 51. So, for example, if the random number is 19, the card printed is the one at array index 18.

    Here is sample output of a test program that calls the `choose_random_card` function:

    ```
    The random number is 19
    The card at position 19 in the deck is: 6H
    ```

    The program uses the abbreviated card designations that we've used in programming assignments 3 through 5. For example, "`6H`" is the designation of the six of hearts.

    Support files: There are two support files for this question, named `random_card.c`, and `cards.h`. Copy them with the UNIX command

    ```
    cp ˜gfisher/classes/101/final/random_card.c ˜gfisher/classes/101/final/cards.h .
    ```

    The o`main` function in `random_card.c` defines the sorted deck that is sent to `choose_random_card`. The `cards.h` file is the same as we've been using in the programming assignments.

    Where to put your answer: Add the definition of the `choose_random_card` function into `random_card.c`, after the comment `/* Define choose_random_card function here: */`.

    How to compile and test: Compile with the command

    ```
    gcc random_card.c -o random_card
    ```

    To test your implementation, run with the command

    ```
    ./random_card
    ```

    Since the program chooses a random number, there is no fixed output to compare against. You can check that your output is correct by looking in the position of the sorted deck to see that our program prints the correct card for the random number it generates. There are comments in the `random_card.c` file that show the card positions in the sorted deck.

    How to hand in:

    ```
    handin gfisher 101_final random_card.c
    ```

    Note that you do not hand in cards.h, since you don't have to modify it.

13. (33 points) For this question, you are writing a `.c` and `.h` file for a program that processes information on surfboards. The program uses a struct data type named `Surfboard`. This struct stores the following information on a surfboard: length, width, number of fins, and name of the board shaper. The length and width are `double`s, number of fins is an `int`, and the shaper name is a string up to 50 characters.

    Your program must declare and define four functions:

    - `int read_data(Surfboard boards[MAX_NUM_BOARDS])` -- prompt the user to enter data on up to twenty surfboards; read the data from `stdin` and store it in the given boards array; the user enters -1 to indicate end of input
    - `double average_board_length(Surfboard boards[], int n)` -- compute the average length of all boards; the value of `n` is the number of boards
    - `double average_board_width(Surfboard boards[], int n)` -- compute the average width of all boards; the value of `n` is the number of boards
    - `int number_of_two_fins(Surfboard boards[], int n)` -- count the number of boards that have two fins

    The `read_data` function can assume that all input values are the correct types, and does need to perform any error checking. The name of the shaper will contain no whitespace characters.

    After reading the data, your program prints the following information to `stdout`:

    - the average length of all the boards
    - the average width of all the boards
    - the number of boards that have two fins

    Support files: There are two support files for this question, named `surfboard_test.c` and `boards.dat`. Copy them with the UNIX command

    ```
    cp ˜gfisher/classes/101/final/surfboard_test.c ˜gfisher/classes/101/final/boards.dat .
    ```

    Where to put your answer: Put your code in files named `surboard.h` and `surfboard.c`. It is up to you to determine what goes in each file, but they must compile and run successfully with the supplied testing file `surfboard_test.c`

    How to compile and test: Compile with the command

    ```
    gcc surfboard.c surfboard_test.c -o surfboard_test
    ```

    To test your implementation, run with the command

    ```
    ./surfboard_test < boards.dat
    ```

    The `boards.dat` file is a plain text file with data for five different boards. Since your program reads from `stdin`, it will accept data from a redirected input file like `boards.dat`. When your program runs, it should produce output like this:

    ```
    Input length, width, number of fins, and shaper name: ...
    Average board length = 6.85
    Average board width = 1.73
    Number of two-fin boards = 2
    ```

    You can ignore the first line of output, since it's just the prompting strings from the `read_data` function. The last three lines of output are the results printed to `stdout`. Your message text and formatting does not have to match this output format exactly, but the numeric results of `6.85`, `1.73`, and `2` must be there.

How to hand in:

```
handin gfisher 101_final surfboard.h surfboard.c
```

Note that you do not hand in surfboard_test.c, since you don’t have to modify it.