CS 101, Spring 2017 — May 4th — Exam 4 Name: Solution

Question 1. [5 points] Consider the following partially-complete program, which begins on the left and continues on the right:

```
#include <stdio.h>

void set_to_ten(int *p);

void set_to_ten(int *p) {
  *p = 10;
}
```

```
int main(void) {
  int x = 4;

  HERE

  printf("x=%i\n", x);
  return 0;
}
```

Show a statement that can be substituted for `HERE` so that the output of the program will be `x=10`. Hint: call the `set_to_ten` function.

```
set_to_ten(&x);
```

Question 2. [5 points] Consider the following struct data type:

```
struct Point {
    double x, y;
};
```

(a) Show code to declare a variable called `planet` whose data type is `struct Point`.

```
struct Point planet;
```

(b) Show code to initialize the x and y coordinates of `planet` to 45.2 and 3.07, respectively.

```
planet.x = 45.2;
planet.y = 3.07;
```

Question 3. [5 points] Consider the following partially-complete program, which begins on the left and continues on the right:

```
#include <stdio.h>

struct Point {
  double x, y;
};

void point_print(struct Point p);

void point_print(struct Point p) {
  printf("x=%.2lf, y=%.2lf\n",
         p.x, p.y);
}
```

```
int main(void) {
  struct Point p;
  printf("Enter x and y values: ");
  scanf("%lf %lf", &p.x, &p.y);

  HERE

  return 0;
}
```

Show code that can be substituted for `HERE` so that the `point_print` function will print the x and y coordinate values entered by the user. (*I.e.*, your code should be a call to the `point_print` function.)

point_print(p);

Question 4. [15 points] Consider the following partially-complete program, which begins on the left and continues on the right:

```
#include <stdio.h>

#define NUM_PIXELS 100

struct Point {
  double x, y;
};

struct Velocity {
  double dx, dy;
};

struct Pixel {
  int red;
  int green;
  int blue;
  struct Point location;
  struct Velocity velocity;
};
```

```
void move_pixels(struct Pixel p[], int size);

int main(void) {

  HERE1

  // assume pixels array has been initialized

  HERE2

  return 0;
}

void move_pixels(struct Pixel p[], int size) {

  HERE3

}
```

(a) Show code that can be substituted for HERE1 to declare an array, named `pixels`, composed of `NUM_PIXELS` elements of type `struct Pixel`.

```
struct Pixel pixels[NUM_PIXELS];
```

(b) Show code that can be substituted for HERE2 to call the `move_pixels` function, passing in the `pixels` array, and the size of the array. (Assume that the elements of the `pixels` array have been initialized.)

```
move_pixels( pixels, NUM_PIXELS );
```

(c) Show code that can be substituted for HERE3 that will cause `move_pixels` function to update each pixel's location based on its respective velocity, i.e., add dx to x and store in x, add dy to y and store in y. HINT: This will require a `for` loop inside the `move_pixels` function.

```
for (int i = 0; i < size; i++) {
    p[i].location.x += p[i].velocity.dx;
    p[i].location.y += p[i].velocity.dy;
}
```

Question 5. [10 points] Consider the following partially-complete program, which begins on the left and continues on the right:

```
#include <stdio.h>

struct Point {
  double x, y;
};

struct Point point_create(
  double xval, double yval);

struct Point point_create(
  double xval, double yval) {

  HERE1

}
```

```
int main(void) {
  struct Point p;

  HERE2

  printf("x=%.2lf, y=%.2lf\n",
    p.x, p.y);
  return 0;
}
```

(a) Show code to substitute for `HERE1` that will return a `struct Point` initialized by the x and y coordinate values specified by the parameters `xval` and `yval`.

```
struct Point result;
result.x = xval;
result.y = yval;
return result;
```

(b) Show code to substitute for `HERE2` that uses the `point_create` function to initialize the variable `p` such that the `printf` statement will print the output `x=4.50, y=6.33`.

```
p = point_create( 4.50, 6.33 );
```

Question 6. [15 points] Consider the following partially-complete program, which begins on the left and continues on the right:

```
#include <stdio.h>

#define NUM_PIXELS 100

struct Point {
  double x, y;
};

struct Velocity {
  double dx, dy;
};

struct Pixel {
  int red;
  int green;
  int blue;
  struct Point location;
  struct Velocity velocity;
};
```

```
HERE1

int main(void) {
  struct Pixel pixel;
  struct Point location;
  struct Velocity velocity;

  // assume that location and velocity
  // have been initialized

  HERE2

  return 0;
}

HERE3
```

(a) Show code that can be substituted for `HERE1` to declare a prototype for the `init_pixel` function. The function should return `void` and accept the following parameters: a pointer to a `struct Pixel` instance, a pointer to a `struct Point` instance, and a pointer to a `struct Velocity` instance.

void init_pixel(struct Pixel *pix, struct Point *pt, struct Velocity *v);

(b) Show code that can be substituted for `HERE2` that calls the `init_pixel` function, passing `pixel`, `location`, and `velocity` to `init_pixel` by reference (using pointers). You may assume that all three structs have been initialized.

init_pixel(&pixel, &location, &velocity);

[Question 6 continues on next page.]

(c) Show code that can be substituted for HERE3 that provides a definition for the `init_pixel` function. The function should assign the provided ~~Location~~ Point and `Velocity` to the respective elements of the `Pixel` passed to the function. It should also initialize the `red`, `green`, and `blue` elements to 0.

```
void init_pixel(struct Pixel *pix, struct Point *pt, struct Velocity *v);

    pix -> location = *pt;
    pix -> velocity = *V;
    pix -> red = 0;
    pix -> green = 0;
    pix -> blue = 0;

}
```

Programming Questions

Note: For all of the programming questions, you should use `scanf` to read the input value(s) required by the program.

Note: Make sure your programs produce the output in **exactly** the format described, including capitalization and punctuation. You may not receive credit for programs that produce incorrectly-formatted output.

Getting started: Start **Cygwin Terminal** and **Notepad++** and make sure ALL TABS are closed. (Note: do *not* open any other programs.) Your instructor will give you the name of a zip file. In your terminal, run the following commands:

```
cd h:
mkdir -p CS101
cd CS101
curl -O http://faculty.ycp.edu/~dhovemey/spring2017/cs101/zipfile
unzip zipfile
cd CS101_Exam04
```

Note that in the `curl` command, the `-O` has the letter 'O', not the digit '0'.

Substitute the name of the zip file for *zipfile*.

Editing code: Use your text editor to open the source file (e.g., `question7.cpp`) referred to in the question. Do not open any files other than the ones for the exam.

Compiling: To compile the program for Question 7, run the following command in the terminal:

```
make question7.exe
```

Change the number as appropriate for the other questions (e.g., `question8.exe`).

Running: To run the program for Question 7, run the following command in the terminal:

```
./question7.exe
```

Change the number as appropriate for the other questions (e.g., `question8.exe`).

To submit: In Cygwin Terminal, run the command

```
make submit
```

Enter your Marmoset username and password when prompted.

Good luck!

Question 7. [30 points] Complete the program in `question7.cpp` by implementing the `distance_between` and `find_closest` functions.

The program defines a `struct Point` data type to represent a 2-D point. The `main` function, which is provided for you and which you may *not* modify, does the following:

1. Prompts the user to enter an integer count (number of points)
2. Prompts the user to enter x and y coordinates for a sequence of points, which is stored in an array of `struct Point` elements
3. Prompts the user to enter the x and y coordinates of a "target" point
4. Uses the `find_closest` function to find the point in the sequence of points that is closest to the target point

Example run (user input in **bold**):

```
How many points? 4
Enter x/y coordinates of points:
103.7 99.5
5.7 -6.9
45.8 -9.3
11.1 35.6
Enter x/y coordinates of target point: -23.4 8.99
Closest point to target is (5.70, -6.90)
Distance to closest point is 33.16
```

Your tasks are to write function definitions of the `distance_between` and `find_closest` functions.

The `distance_between` function takes two `struct Point` parameters and returns the distance between them. The formula to find the distance between two points is

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

where x_1, y_1 and x_2, y_2 are the x/y coordinates of the two points.

The `find_closest` function takes three parameters: `points`, an array of `struct Point` elements, `num_points`, the number of elements contained in the `points` array, and `target`, a `struct Point` instance. The function should return the `struct Point` element of the array which is closest to `target`. (Use the `distance_between` function to compute distances between points.)

Hints:

- Start by assuming that the first element of the array is the closest; then check the distances between the target point and the other elements of the array
- In addition to an index variable you will need at least *two* additional loop variables: one to keep track of the distance between the target and the current closest point, and another to keep track of *which* element of the array is the closest so far
- You may find it helpful to add `printf` statements to the body of the `find_closest` function to print out the progress of the search for the point closest to the target

Question 8. [15 points] Complete the program in `question8.cpp` to add a definition for the `reverse_pixel_dir` function. It takes a pointer to a `struct Pixel` instance as a parameter, and should reverse the direction of the pixel by reversing (negating the sign of) the dx/dy components of its velocity.

The provided `main` function, which you may not modify,

1. Declares and initializes an instance of `struct Pixel`
2. Prints the initial velocity dx/dy values
3. Calls the `reverse_pixel_dir` function, passing a pointer to the `struct Pixel` instance as an argument
4. Prints the updated velocity dx/dy values

Example run (note that there is no user input):

```
Original pixel velocity: dx=-4.50, dy=5.20
Updated pixel velocity: dx=4.50, dy=-5.20
```

Hints:

- Note that the `reverse_pixel_dir` function takes a *pointer* to a `struct Pixel` instance
- Also note that the `dx` and `dy` fields are part of a nested `struct Velocity` instance within the `struct Pixel` instance