CS 101, Spring 2016 — April 14th — Exam 3 Name: _____

Question 1. [4 points] Consider the following function:

```
void double_and_print(int x) {
  printf("%i", x * 2);
}
```

Show how to call the `double_and_print` function so that the output `24` is printed.

Question 2. [5 points] Given the following function prototype:

```
void drawRect(int width, int height);
```

Determine which of the following function calls are valid for the function. Assume variables `a`, `b`, `length`, `width`, `height`, and `x` all have type **int**.

I. `drawRect(int a, int b);`
II. `drawRect(length, length*2);`
III. `void drawRect(width, height);`
IV. `drawRect(10, sqrt(x));`

a. I only
b. II only
c. III only
d. II and IV
e. I, II, and IV

Question 3. [5 points] From the choices below, circle *all* valid function prototypes (there may be more than one).

a. `int sumNum(int x, y, z);`
b. `float avg(float exam1, int exam2);`
c. `void drawSq(int squareHeight);`
d. `printLine(size);`

For Questions 4–6, circle **True** or **False**.

Question 4. [2 points] **True False** A function can use a `return` statement to return more than one value.

Question 5. [2 points] **True False** Every function must return a value.

Question 6. [2 points] **True False** It is legal to declare a variable with the same name in the bodies of two different functions.

Question 7. [5 points] Complete the following function so that it returns the area of the circle whose radius is passed as the parameter to the function. A circle's area is πr^2, where r is the radius of the circle. You can assume that `M_PI` is a constant `double` value that approximates π.

```
double area_of_circle(double r) {
```

Question 8. [5 points] What output is printed by the following program (which begins on the left and continues on the right)? Note there are `printf` statements in both main and the function.

```
#include <stdio.h>

void foo(int x);

int main(void) {
  int y = 3;
  foo(y);
  printf("y=%i\n", y);
  return 0;
}
```

```
void foo(int x) {
  x = x * 2;
  printf("x=%i\n", x);
}
```

Question 9. [5 points] Consider the following partially-specified program (which begins on the left and continues on the right):

```
#include <stdio.h>

int triple(int x);

int main(void) {
  int y = 4;

  missing code

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

```
int triple(int x) {
  int result;
  result = x * 3;
  return result;
}
```

What code can be substituted for *missing code* so that the program will print the output `y=15`? **Important**: your code must call the `triple()` function with appropriate arguments.

Question 10. [5 points] What output is printed by the following function (which begins on the left and continues on the right)? Note there are `printf` statements in both main and the function.

```
#include <stdio.h>

void mystery(int x[], int size);

int main(void) {
  int y[2] = { 4, 5 };
  mystery(y,2);
  printf("y[0]=%i\n", y[0]);
  printf("y[1]=%i\n", y[1]);
  return 0;
}
```

```
void mystery(int x[], int size) {
  for (int i=0; i < size; i++) {
    x[i] = x[i] * 2;
    printf("x[%i]=%i\n", i, x[i]);
  }
}
```

Question 11. [5 points] What output is printed by the following program (which begins on the left and continues on the right)?

```
#include <stdio.h>
#define SIZE 5
int arr_mod(int arr[], int size);

int main(void) {
  int nums[SIZE] = {3, 6, 8, 4, 7};
  int n = arr_mod(nums, SIZE);
  printf("n=%i\n", n);
  return 0;
}
```

```
int arr_mod(int arr[], int size) {
  int n = 1;
  for (int i = 1; i < size; i++) {
    if (arr[i] < arr[i - 1]) {
      n++;
    } else {
      n = 1;
    }
  }
  return n;
}
```

Question 12. [5 points] What value is returned by the following function (which begins on the left and continues on the right)? Briefly explain.

```
#include <stdio.h>

int doStuff(int x,int y,int z);

int main(void){
  int r=14, s=1, t=7;
  int ans;
  ans = doStuff(r-5,2*s,t+7);
  printf("The result is:
         %i\n",ans);
  return 0;
}
```

```
int doStuff(int x, int y, int z)
{
  int r;
  r = x;
  if (r < y) {
    r = y;
  }
  if (r < z) {
    r = z;
  }
  return r;
}
```

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/~dbabcock/spring2016/cs101/zipfile
unzip zipfile
cd CS101_Exam3
```

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., `question13.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 13, run the following command in the terminal:

```
make question13.exe
```

Change the number as appropriate for the other questions (e.g., `question14.exe`).

Running: To run the program for Question 13, run the following command in the terminal:

```
./question13.exe
```

Change the number as appropriate for the other questions (e.g., `question14.exe`).

To submit: In Cygwin Terminal, run the command

```
make submit
```

Enter your Marmoset username and password when prompted.

Good luck!

Question 13. [20 points] Complete the program in `question13.cpp` as follows.

Code is provided in the program's `main` function to prompt the user to enter a number of cents. It also has output statements printing the number of whole dollars and the remaining change. **DO NOT MODIFY** these parts of the program in any way.

Your task is to add function prototypes and defintions for `get_whole_dollars()` and `get_change()`, and corresponding calls in `main()`. Code should only be added in the indicated locations.

Part I

Add function prototypes for the `get_whole_dollars()` and `get_change()` functions above `main()`.

- `get_whole_dollars()` function takes a single `int` parameter and returns an `int`.
- `get_change()` function takes a single `int` parameter and returns an `int`.

DO NOT write the function definitions at this point.

Part II

Add function calls in `main()` to call the `get_whole_dollars()` and `get_change()` functions passing appropriate arguments.

- Store the result of the `get_whole_dollars()` function into the variable `d`.
- Store the result of the `get_change()` function into the variable `c`.

The `scanf` and `printf` statements are already provided for you.

Part III

Add function definitions below `main()` for the `get_whole_dollars()` and `get_change()` functions.

- `get_whole_dollars()` takes a single `int` parameter representing the number of cents, and should compute and return the number of whole dollars.
- `get_change()` takes a single `int` parameter representing the number of cents, and should compute and return the number of cents left after taking away the whole dollars

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

```
Number of cents: 437
Whole dollars=4
Change=37
```

Hints:

- You **DO NOT** need to get any user input or print any output as that code has been provide.
- The `/` and `%` operators will be useful in performing the computations.
- Don't forget to include a `return` statement at the end of each function.

Question 14. [15 points] Complete the program in `question14.cpp` as follows.

The program's `main()` function (provided) reads the `double` values for the center coordinates and the radii for two circles, calls the `does_intersect()` function, and prints out whether or not the circles intersect. **DO NOT MODIFY** `main()` in any way.

Your tasks are to add function definitions for the `compute_distance()` and `does_intersect()` functions. Function prototypes are provided at the top of the code.

- `compute_distance()` takes four `double` parameters representing the `x` and `y` coordinates of two points. The function should compute the distance between the points using the formula $d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$ and return this distance as a `double` value. This function **should not** print any output.
- `does_intersect()` takes six `double` parameters representing the `x` and `y` coordinates of the center, and the radius `r` for each of the two circles. This function should compute the distance between the centers **using the** `compute_distance()` **function** and print the result as shown in the example output. It should then compute the sum of the radii and print the result as shown in the example output. Finally the function should determine if the two circles intersect and return 1 if they intersect and 0 otherwise. Two circles intersect if the distance between their centers is less than the sum of their radii.

`does_intersect()` **MUST** call the `compute_distance()` function in order to receive full credit for this problem.

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

```
Enter x, y, and r values for circle 1: 1.5 2.5 3.0
Enter x, y, and r values for circle 2: -1.0 2.0 4.25
The distance between the centers is 2.55
The sum of the radii is 7.25
The circles intersect
```

Another example run (user input in **bold**):

```
Enter x, y, and r values for circle 1: 0.75 1.2 1.0
Enter x, y, and r values for circle 2: -2.5 -4.0 2.1
The distance between the centers is 6.13
The sum of the radii is 3.10
The circles do not intersect
```

Hints:

- You **DO NOT** need to get any user input as that code has been provide.
- Function prototypes for the `compute_distance()` and `does_intersect()` functions are provided. Each function takes a single `int` parameter, and returns an `int` value.
- To compute the square root use the `sqrt()` function.
- Don't forget to include a `return` statement at the end of each function.

Question 15. [15 points] Complete the program in `question15.cpp` as follows. The program's `main` function (provided) reads an integer specifying how many `int` values will be entered (up to 10), reads that many `int` values, and stores them in an array. It should then call the `neg_sum` function, passing the array and the number of values as parameters, and print the sum returned by the function.

Your tasks are as follows:

Add a function definition for the `neg_sum` function. Its return type is `int`, and it takes two parameters, the first being an array of `int` values, and the second being an `int` specifying how many values the array has. Refer to the provided function declaration (prototype).

The `neg_sum` function should return the sum of the *negative* values in the array. It should ignore any non-negative values, and should return 0 in the case that the array does not contain any negative values.

Add the function call to `neg_sum` in `main()`, where shown, and pass the array and the array size to it. Capture the return value from `neg_sum` in the `sum` variable.

$\Longrightarrow$ **Important**: The *only* modification to `main` is the single line call to `neg_sum` mentioned above. $\Longleftarrow$

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

```
How many values? 6
Enter values: -5 -4 1 3 2 -3
Sum of negative values is -12
```

Hints:

- The function prototype (also called a function declaration) has been provided. It specifies the return type, name, and parameters of a function, but does not have a body.
- A function definition looks just like a function prototype, but does specify a body (a block of statements inside curly braces.)
- In the body of the `neg_sum` function, don't forget to initialize the variable that will accumulate the sum of the negative values, and to return it as the result of the function