CSCI 102: Fundamentals of Computation
Spring 2021 - Final Exam
5/10/21, 4:30 PM – 6:30 PM
(1 hour 40 to answer, 20 to scan/upload submit)

<No Need to Fill this out due to Gradescope>

Name:____Solutions__________________________

Student ID: ____

Email: _____________________@usc.edu

Lecture section:

Redekopp
M/W 10 a.m. / 11 a.m.

Ques	Your score	Max score	Recommended Time
1		10	10 min.
2		10	20 min.
3		10	20 min.
4		10	20 min.
5		10	30 min.
Total		**50**	**100 min +20 min upload**

Questions 2-5 contain a link/URL to a skeleton file in which you can write/edit the provided skeleton code.

We prefer you upload the finished .cpp files but you may handwrite/annotate the pages below and submit a scanned PDF for each problem.

1. (10 pts.) Multiple Choice / Short Answer

Enter answers directly on Gradescope.

1.1. **True / [✓ False]**: If a single `int` (e.g. `int x`) is passed to a function (e.g. `void f1(int x)`) and modified in the function, that modification is visible back in the calling function (e.g. `main()`).

1.2. **[✓ True] / False**: If an `int` array (e.g. `int y[10]`) is passed to a function (e.g. `void f2(int y[])`) and modified in the function, those modifications are visible back in the calling function (e.g. `main()`).

1.3. **[✓ True] / False**: The if statement: `if( 102 ) { /* code */ }` will always be true and execute the given code.

1.4. **True / [✓ False]**: The string of characters "abc" requires 3 bytes total when stored in memory or a character array.

1.5. **[✓ True] / False**: Given `string s` and the command: `cin >> s;` if the user types `hello world`, then `s` will only contain `hello`.

1.6. **[✓ True] / False**: Given `int y[10],` the last legal index is 9.

1.7. **True / [✓ False]**: Given `int y[10],` the contents of all 10 integers can be printed with the command: `cout << y << endl;`

1.8. **True / [✓ False]**: The following for loop results in an infinite loop.

```
for(int i=10; i != 1; i-=2) { i--; cout << i << endl; }
```

1.9. **[✓ True] / False**: The following for loop results in an infinite loop.

```
for(int i=0; i < 10; i+1) { cout << i << endl; }
```

1.10. Assuming x is a **string**, enter any value of x that will cause the following condition to evaluate to **true**.

```
if( x.size() > 1 && !(x < "a" || x >= "ac") && x[0] != x[1] )
```

a?*, where ? = any character < 'c' (e.g. '1-9','A-Z','b') but not 'a' and * is any string

2. **(Coding – 10 pts.)** Write a program that compares the **volume** of **3** shapes chosen from the options: **sphere, cone, and cylinder**. As a reminder, the equations for the volume of these shapes are:
   - $Sphere\ Volume = {}^{4}/_{3}\,\pi r^3$, where **r** is the radius of the sphere
   - $Cone\ Volume = {}^{1}/_{3}\,\pi r^2 h$, where **r** is the radius of the cone and **h** is the height
   - $Cylinder\ Volume = \pi r^2 h$, where **r** is the radius of the cylinder and **h** is the height

The user will input a letter: s (for sphere), o (for cone), or y (for cylinder) followed by the pertinent dimensions. For a **sphere** the user will only enter a radius value after s. However, for **cone** and **cylinder** the user will enter a **radius value first** followed by a value for the **height** of the cone or cylinder.

The user will enter all the information about the first shape followed by information for the second shape. You may assume the first shape will always be different than the second shape. The program should then output which shape is larger (e.g. sphere is bigger, cone is bigger, or cylinder is bigger) or the word Tie if both volumes are the same.

The user will enter all the information about the first shape followed by information for the second shape. You may assume the first shape will always be different than the second shape. The program should then output which shape is larger (e.g. sphere is bigger, cone is bigger, or cylinder is bigger) or the word Tie if both volumes are the same.

Example 1	A cylinder with r=1 and h=0.5 vs. a sphere with r=1	**Example 2**	A cone with radius 1 and height 4 and a sphere with radius 1

Input	`y 1 0.5`
	`s 1`
		Input	`o 1 4`
			`s 1`

Output	`sphere is bigger`	**Output**	`Tie`

The skeleton file on the next page can be downloaded with the link: volume.cpp

This problem will be shorter and simpler by using functions. Two recommended options are prototyped below. **You DO NOT have to use functions and can change/delete them**. Use them if you believe it makes your work faster and simpler, or do not use them if you prefer.

```
#include <iostream>
#include <cmath>
using namespace std;
// Given the specified shape's input letter/code (e.g. 's', 'o', 'y'),
// get's the radius and/or height input using cin and returns the volume
double getVolume(char shape);

// Prints 1 of the 3 output messages based on the input argument which is the
// letter/code of the largest volume shape (e.g. 's', 'o', 'y').
// Does not handle the case of a tie which must be handled outside the function.
void printBiggerShape(char shape);
```

```
int main()
{   double pi = M_PI; // you can use this variable where you need PI
    char s1, s2;
    cin >> s1;
    double a1 = getVolume(s1);
    cin >> s2;
    double a2 = getVolume(s2);
    if(a1 > a2){
        printBiggerShape(s1);
    }
    else if( a2 > a1) {
        printBiggerShape(s2);
    }
    else {
        cout << "Tie" << endl;
    }
    return 0;
}
double getVolume(/* Choose parameters */char shape)
{
    double PI = M_PI, r, h, a;
    if(shape == 's'){
        cin >> r;
        a = 4*PI*r*r*r/3;
    }
    else if (shape == 'o'){
        cin >> r >> h;
        a = PI*r*r*h/3;
    }
    else if(shape == 'y'){
        cin >> r >> h;
        a = PI*r*r*h;
    }
    return a;
}
void printBiggerShape(char shape)
{
    if(shape == 's') {
        cout << "Sphere is bigger" << endl;
    }
    else if(shape == 'o') {
        cout << "Cone is bigger" << endl;
    }
    else {
        cout << "Cylinder is bigger" << endl;
    }
}
```

3. **Debugging / Coding (10 pts.)**: Billy Bruin was writing a program to find the maximum length sequence of consecutive equal values in an array and list the **start** and **end** index of that sequence. As an example, given the array: `3 7 7 7 7 5 5 1 2 2 2`, the maximum length sequence of consecutive equal values are the **7s** which start at index **1** and end at index **4.** (Note: we want to show the **indexes** of the start and end of the sequence.) However, Billy was struggling to get his code working. While there are many ways to implement this, you must help fix any bugs in Billy's code and complete the blanks that Billy was unable to fill in. But other than that, you may not change the current implementation.

   His general idea was to use a function to read in the array and update the length in a variable, **n**. Then he wanted to scan through the array and call another function that takes a "starting" location, scans through the sequentially next values (after the starting location) stopping as soon as it reaches a value that differs from the starting element. It would then return the size of that sequence which `main()` could use appropriately.

   Look through the written code and fix any issues with the prototype, definition, and/or call of the `getInput` function. Also, fill in the blanks with appropriate code to make the program work correctly. You may **NOT** alter any other given code (other than the code related to `getInput` and the given blanks).

   Requirements
   - The input will be at most 50 integers and will end with **-1** which should NOT be stored in the array
   - The program must output the start and end **index** of the longest/max consecutive sequence of values.

   The skeleton file can be downloaded with the link: maxconsec.cpp
   You may need to right-click and choose Save As or Save Target As to download the `.cpp` file or open it in your browser and save the file.

```
#include <iostream>
using namespace std;

// Find issue(s) related to this function including its prototype, definition, and call
//  in main() of this function. Change this prototype line, if needed
int getInputData(int vals[], int n);

// Do not change this line
int nextRun(int vals[], int n, int start);
int main()
{
    int values[50];
    int n=0, step, max_s = 0, max_len = 0;

    // Fix any issues with this call (update this one line if needed)
    n = getInputData(values, n);

    // Correct code
    for(int s = 0; s < n; s += step ){
        step = nextRun(values, n, s);

        if( _step > max_len__ ){
            max_len = step;

            __max_s = s;______________________
        }
    }
    // Find and fix the error in this line
    cout << "Max start: " << max_s << " to end: " << max_s + max_len -1 << endl;
    return 0;
}
// Fix the issue with this function (update any lines necessary)
int getInputData(int vals[], int n)
{
    int temp;
    n = 0;
    cin >> temp;
    while(temp != -1){
        vals[n] = temp;
        n++;
        cin >> temp;
    }
    return n;
}
// returns the number of consecutive equal values beginning at 'start'
int nextRun(int vals[], int n, int start)
{
    if(start >= n){ return 0; }
    int end = start+1;
    while(end < n){
        if(vals[start] != vals[end]){

            __break;_________
        }

        __end++;_____________
    }
    return end-start;
}
```

4. **Tracing / Functions (10 pts.)**: An **anagram** is a rearrangement of a given set of letters to form a different word. For example, $\boxed{\textbf{StoP}}$ and $\boxed{\textbf{Tops}}$ are anagrams (ignoring case). The program below should take in **two** strings and determine if they are anagrams (**ignoring upper or lower case**). You are given the code below which provides a set of functions and a skeleton for `main()`. Fill in the blanks in `main()` to arrive at a correct program by **ONLY** calling the appropriate functions, performing assignment, and possibly using simple arithmetic operations. You **may NOT alter any other code** in `main()`.

   You may not need ALL the functions defined in the code; use only the ones you need and you may call a function more than once. You should read/analyze the code in the various functions to determine what they do and then consider how they can be used to determine if the two input strings are anagrams (ignoring case). The generally strategy is to iterate through the letters of one word, attempting to find and "cross off" the corresponding letter in the other. If only "crossed off" letters remain at the end, we know the two inputs were anagrams.

   To help you understand the functions, we will ask some tracing questions about the functions that you need to answer for partial credit. By answering those questions, you will better understand how all the functions work and can decide how to use them to achieve the overall task. **The questions are at the top of the skeleton code**. **Enter your answers in the comments of the linked .cpp skeleton file or directly on the PDF page below**.

   The skeleton file can be downloaded with the link: anagram.cpp

   You may need to right-click and choose Save As or Save Target As to download the `.cpp` file or open it in your browser and save the file.

```
// ------------ ANSWER THE FOLLOWING QUESTIONS ----------------------
// Suppose you are given two strings: string s1 = "abcabc", s2 = "zzz";
// (1 pt.) What string is returned by a call to f1(s1,'c');  ___ab.abc_____?
// (1 pt.) What integer is returned by a call to f2(s2, 'z', true); ___3_______?
```

```
#include <iostream>
#include <string>
#include <cctype>
using namespace std;

string f1(string a, char c);
int f2(string g, char c, bool m);
string f3(string x);
void f4(string x);

// Functions shown in 2 columns to be visually more compact and readable
 string f1(string a, char c)                   int f2(string g, char c, bool m)
 {                                             {
     int i = 0;                                    int t = 0;
     while( i != a.size()){                        if(m){
         if(a[i] == c){                                for(int i=0; i < g.size(); i++){
             a[i] = '.';                                   if(g[i] == c){
             break; // break immediately                       t++;
         }                                         }   }   }
         i++;                                      else {
     }                                                 for(int i=0; i < g.size(); i++){
     return a;                                             if(g[i] != c){
 }                                                             t++;
                                                   }   }   }
                                                   return t;
                                               }


 string f3(string x)                           void f4(string x)
 {                                             {
     for(int i=0; i < x.size(); i++){              for(int i=0; i < x.size(); i++){
         x[i] = toupper(x[i]);                         x = x.substr(i) + 'A';
     }                                             }
     return x;                                 }
 }


// Complete main() on the  next page
// COMPLETE MAIN BELOW
int main()
{
    string letters, word;
    int n1, n2;
    cin >> letters >> word;

    // Remove the /* */ comments and fill in the blanks below
    // add 1 or more lines of code immediately below that may include
    // function calls, assignment, and/or arithmetic

     ___letters = f3(letters);________ ;

     ____word = f3(word);____________________________ ;
    for(int i=0; i < word.size(); i++){
        // Remove the /* */ comments and fill in the blank below
        // add 1 line of code immediately below that may include
        // function calls, assignment, and/or arithmetic

         ___letters = f1(letters, word[i]);_______ ;
    }
    // Remove the /* */ comments and fill in the blanks below
    // complete the two lines of code immediately below that may include
    // function calls, assignment, and arithmetic

    n1 =  ___f2(letters, '.', true);_________ ;

    n2 =  ___ f2(letters, '.', false);_________ ;
    if(n1 == word.size() && n2 == 0){
        cout << "Anagram!" << endl;
    }
    else {
        cout << "Not an anagram!" << endl;
    }
    return 0;
}
```

5. **Coding / Arrays (10 pts.):** In this program, the user will input between **2 to 100 `x,y` points** on a Cartesian plane and your task is to determine if **at least one point** is at most a distance/radius, **`r`**, to **ALL** others (where **`r`** is chosen and input by the user). Your program should:

   1. First read in a **distance threshold**, **`r`** (assume **r >= 0**) from the keyboard which could contain a decimal.
   2. Read in an **integer, `n`**, (assume **2 <= n <= 100**) from the keyboard
   3. Proceed to read in **`n`** pairs of numbers (could be decimals) representing the **`x`** and **`y`** coordinates of a point on the Cartesian plane (the **`x`** value is entered first followed by the **`y`** value and is separated by whitespace).
   4. Then determine **if there exists at least one x,y point that is at most a distance, r, to all the other points**.
      - If such a point exists, output a message containing that **`x,y`** point. (If many such points exists, you can output **ANY** single one of them...your choice which one).
      - If no such point exists, output a message indicating that.

You should store each x value in an **array of x-coordinates** and y value in an **array of y-coordinates**. Once entered, your code should check if at least 1 of the points has a distance of **`r`** or less to all others. Recall that the formula for the distance between two points: **(x0,y0)** and **(x1,y1)** is:

$$\boldsymbol{distance = \sqrt{(x0 - x1)^2 + (y0 - y1)^2}}$$

Note: It may be helpful to write a function to compute the distance between two points:

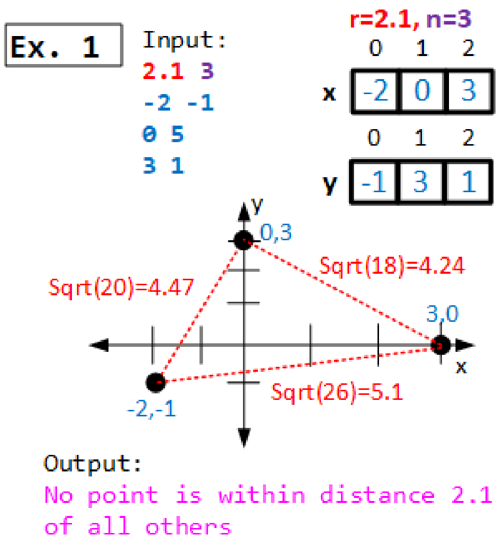
```
   double dist(double x0, double y0, double x1, double y1);
```

Example 1: For example, consider the example below with **`r=2.1`** and **3 points**: **`(-2,-1)`**, **`(0,5)`**, and **`(3,1)`**:

```
2.1  3
-2  -1
0   5
3   1
```

Because there is no point within distance, `r=2.1`, of all others, the program would output:

```
No point is within distance 2.1 of all others
```



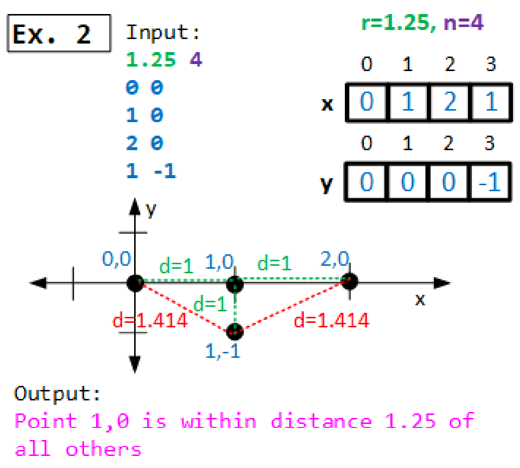
```
Input:
2.1 3
-2 -1
0 5
3 1
```

	0	1	2
x	-2	0	3
y	-1	3	1

```
Output:
No point is within distance 2.1
of all others
```

Example 2: However, given the next example:

```
1.25  4
0 0
1 0
2 0
1 -1
```



```
Ex. 2
Input:
1.25 4
0 0
1 0
2 0
1 -1
```

```
Output:
Point 1,0 is within distance 1.25 of
all others
```

Only point **(1,0)** is within **1.25** of all others (it is a distance of exactly 1 to all three other points). The output should indicate success and list the point.

```
Point 1,0 is within distance 1.25 of all others
```

Example 3: If we assume the same inputs as Example 2 (above) but now **r** is entered as **1.5** (as shown below), then point **(1,0)** and **(1,-1)** are within a distance of **1.5** of all others. You could choose which one to output:

```
Point 1,0 is within distance 1.5 of all others
```

or

```
Point 1,-1 is within distance 1.5 of all others
```

Requirements & Assumptions

- The input representing **n** (number of points) is guaranteed to be **2 or more**.
- The input representing **r** will be positive.
- Inputs can be **double**s and not just **int**s.
- Any output of a **double** can be printed with arbitrary precision (you don't need a fixed number of decimal places for your outputs).
- **You can output any point that satisfies the criteria**. You **need NOT find ALL** points that satisfy the criteria.
- Your output message should be the appropriate message from the two shown below, where **r** is replaced with the actual distance input by the user, and **x,y** is replaced with the satisfying **x,y** coordinate values (as shown in the examples above).

```
Point x,y is within distance r of all others
```

or

```
No point is within distance r of all others
```

The skeleton file on the next page can be downloaded with the link: within.cpp

```
#include <iostream>
#include <cmath>  // cmath provides for sqrt()
using namespace std;
// Add any prototypes here


int main()
{   double x[100], y[100];  //  arrays to hold x,y points
    double r;               //  distance/radius threshold
    int n;                  //  number of x,y points
    cin >> r >> n;          //  get distance threshold and number of points
    for(int i=0; i < n; i++) {
        cin >> x[i] >> y[i];  // read in each x,y point 1 at a time
    }
    // Declare more variables and write your code below.
    int center = -1;
    for(int i = 0; i < n; i++){
        bool within = true;
        for(int k = 0; k < n; k++) {
            if( dist(x[i], y[i], x[k], y[k]) > r){
                within = false;
                break;
            }
        }
        if(within){
            center = i;
        }
    }
    if(center != -1){
        cout << "Point " << x[center] << "," << y[center] << " is within distance " << r
<< " of all others" << endl;
    }
    else {
        cout << "No point is within distance " << r << " of all others" << endl;
    }

    return 0;
}
// Define any functions below
double dist(double x0, double y0, double x1, double y1)
{
    return sqrt((x0-x1)*(x0-x1) + (y0-y1)*(y0-y1));
}
```