CSCI 102: Fundamentals of Computation
Fall 2020 - Final Exam
11/21/20, 11 AM – 1 PM
(1 hour 40 to answer, 20 to scan/upload submit)

Name:_____Solution________________

Student ID: ____

Email: ___________________@usc.edu

Lecture section:

Redekopp
M/W 1 p.m. / 2 p.m.

Ques	Your score	Max score	Recommended Time
1		8	10 min.
2		7	10 min.
3		8	25 min.
4		10	20 min.
5		8	15 min.
6		9	20 min.
Total		**50**	**100 min +20 min upload**

Questions 3-6 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.

1. (8 pts.) Multiple Choice

Enter answers directly on Gradescope.

1.1. **True / [✓ False]**: When variables are declared (e.g. `int x;`), they are automatically initialized to 0.

1.2. Single variables (non-arrays) are passed by (**[✓ value] / reference**).

1.3. **True / [✓ False]**: If an `else if` statement is used, it must be followed by an `else` statement.

1.4. **True / [✓ False]**: A string of characters stored in a character array (e.g. `char str[10];`) must be terminated with the `'\n'` character.

1.5. **[✓ True] / False**: Two C++ string types (e.g. `string s1, s2`) can be compared using normal comparison operators (e.g. <, ==, >, etc.)

1.6. **[✓ True] / False**: A `break` statement is used to immediately quit or jump out of the inner-most loop.

1.7. **True / [✓ False]**: Given a function with signature: `void area(int wid, int len);` the following cout statement will compile: `cout << area(8,5) << endl;`

1.8. Assuming x is an integer, what value(s) of x will cause the following condition to evaluate to **true**. `if( !(x > 7 || x <= 4) && x != 6 )`

a) x can be 4, 5, or 7
b) x can be 4 5, 6, or 7
[✓ **c) x can be 5 or 7**]
d) x can be any value outside of the range 4 through 8

2. **Tracing (7 pts.):** For 2.1 – 2.6, show what the program will output when run.. You must trace the behavior manually. For 2.7, provide an input string that the user can type as input that will cause yes to be printed by the following line. Answers may vary (any correct answer will be accepted).
   You may NOT compile nor run this program on your own system or some website

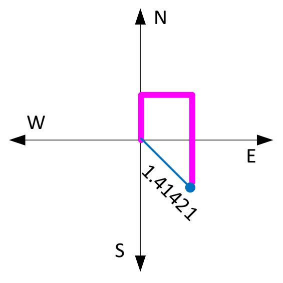
   Enter all 7 answers in the single textbox on Gradescope. Number each line: 2.1, 2.2, etc. followed by the answer to that question.

```
/* You may NOT use a compiler for this problem. */
#include <iostream>
#include <algorithm>
#include <string>
using namespace std;

string comp(string s);
int main()
{
    int x[] = { 4, 3, 1, 4, 9, 0};
    cout << x[1] << endl;                        // 2.1 Sol: 3
    cout << x[ x[1] ] << endl;                   // 2.2 Sol: 4
    cout << x[ min( max(2, x[3]), 3) ] << endl; // 2.3 Sol: 4

    string y = "ONLINE", z = "Pass";
    cout << y.size() << endl;                    // 2.4 Sol: 6
    cout << y.substr(y.size()-4) << endl;        // 2.5 Sol: "LINE"
    cout << z.substr(0,1) + y.substr(3) << endl;// 2.6 Sol: "PINE"

    // Provide an input that could be typed by the user
    // that will cause 'yes' to be printed
    cin >> y;                                    // 2.7
Answers may var: Use the ASCII table if needed.  Might be "b", "b1", "bA" or start
with"az" plus 1 or more additional characters (eg. "b", "aza", "azz",
"azaaaaaaaa", …)
    cout << comp(y) << endl;
    return 0;
}
string comp(string s)
{    if(s > "az" && s < "ba" )  {
        return "yes";
    }
    return "no";
}
```

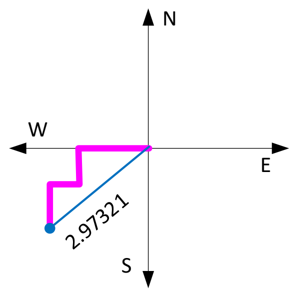
3. Coding (8 pts.) Consider a Cartesian (2D) plane where a traveler starts at the origin (x,y=0,0). A traveler will input a series of choices from the options {**n, s, e, w**} representing the direction (**north, south, east, west**, respectively) and the **distance** (which can contain decimals) they will travel in that direction. Stop when the user types: **q** or **Q** rather than one of the directions: **n, s, e, w**. At that time you should **output the straight-line distance from origin (x,y=0,0) to the travelers current location**. Complete the program to correctly implement this program. (You may assume only valid letters: **n, s, e, w, q**, or **Q** are entered). See the example graphics below.
Recall that distance can be found using Pythagorean's theorem for a right triangle with sides a and b, and hypotenuse, c: $c^2 = a^2 + b^2$

The skeleton file can be downloaded with the link: nwse.cpp. You may use a compiler.

```
#include <iostream>
#include <cmath>
using namespace std;
int main()
{   char dir;
    double dist;
    // You may declare more variables
   double amt, x = 0, y = 0;

    cin >> dir;
    while(dir != 'q' && dir != 'Q')
    {
        cin >> amt;
        if(dir == 'n'){
            y += amt;
        }
        else if(dir == 'w'){
            x -= amt;
        }
        else if(dir == 's'){
            y -= amt;
        }
        else if(dir == 'e'){
            x += amt;
        }

        cin >> dir;
    }
    dist = sqrt(x*x + y*y);
    // Output straight line distance
    cout << dist << endl;
    return 0; }
```



```
Input:
n 1
e 1
s 2
q

Output:
1.41421
```



```
Input:
w 1.5
s 1
w 0.5
s 1.2
q

Output:
2.97321
```

4. **Using Functions (10 pts.)**.

4.1. Consider the 2 functions provided below: **sr**, **sp.** Examine the code in **test_sr** and show the contents of the **vals** array and **data** array after **test_sr()** calls each function. Write your answer in the text box below. Show the values of the vals array on the first line and then the data array on the second line (separate each value by a space)

```
void sr(int dat[], int s, int e, int v, int d)
{
    for(int i=s; i != e; i++){
        dat[i] = v;
        v += d;
    }
}
void sp(int d[], int n)
{
    for(int i=0; i < n-1; i+=2) {
        int t = d[i];
        d[i] = d[i+1];
        d[i+1] = t;
    }
}
void test_sr()
{
    int vals[8] = { 10, 11, 12, 13, 14, 15, 16, 17};
    sr(vals, 2, 6, 9, -1);
    // show all 8 values in the vals array
    int data[8] = { 10, 11, 12, 13, 14, 15, 16, 17};
    sp(data,4);
    // show all 8 values in the data array
}
```

Fill in your answer in the Gradescope textbox (or below if submitting a PDF). **NO compiler may be used.**

```
vals[]: 10 11  9  8  7  6 16 17
data[]: 11 10 13 12 14 15 16 17
```

4.2. On the next page, complete the main() by **ONLY** making (possibly multiple) calls to **sr**, **sp** (from this page) and the **pr** function on the next page. **main**() should produce the output shown below.

Desired output:

```
 0  1  2  3  4 |  5  6  7  8  9 | 19 18 17 16 15 | 14 13 12 11 10 |
 0  1  2  3  4  5  6  7  8  9 # 18 19 16 17 14 15 12 13 10 11 #
```

You may only add function calls to main(). You may **NOT** add **loops, if, else if or else statements**. You may **NOT declare other variables, add other couts, or even add other assignment statements**. Add function calls only!

The skeleton file can be downloaded with the link: funcid.cpp. You may NOT use a compiler.

Desired output (repeated again for your benefit):

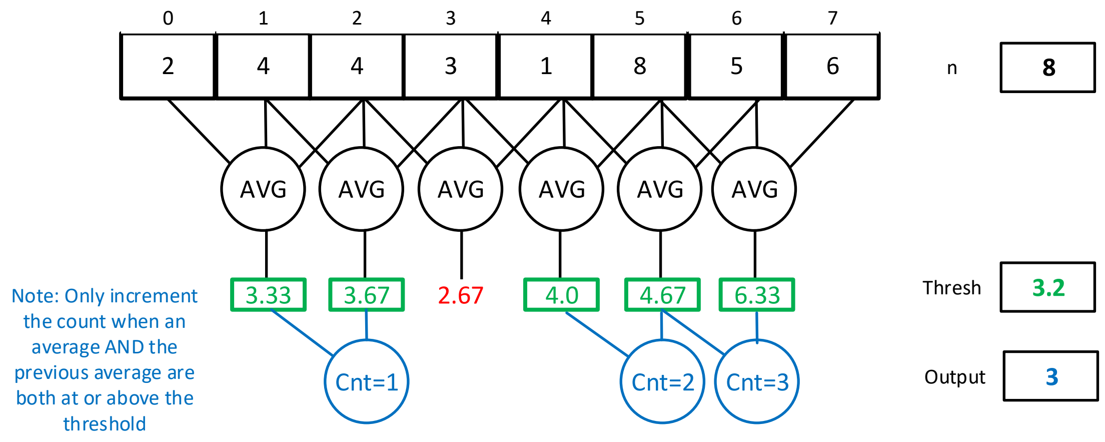
```
 0  1  2  3  4 |  5  6  7  8  9 | 19 18 17 16 15 | 14 13 12 11 10 |
 0  1  2  3  4  5  6  7  8  9 # 18 19 16 17 14 15 12 13 10 11 #
```

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

/* assume sr() and sp() are provided as shown on the previous page */

void pr(int d[], int n, int w, char c)
{   // reminder: setw(2) ensures all numbers printed take 2 spaces for pretty output
    for(int i=0; i < n; i++){
        if(i % w == w-1){
            cout << setw(2) << d[i]  << " " << c << " ";
        }
        else {
            cout << setw(2) << d[i]  << ' ';
        }
    }
    cout << endl;
}

int main()
{
    int vals[20];
     /* your code here - only calls to the above functions */
    sr(vals, 0, 10, 0, 1);  // set first half
    sr(vals, 10, 20, 19, -1);  // set second half

    pr(vals, 20, 5, '|');  // print values

    // next 2 lines can be in either order
    sp(vals, 10);  // swap first 10
    sp(vals, 20);  // swap all 20 (so first 10 get swapped back, but 2nd 10 swap)

    pr(vals, 20, 10, '#');  // print values
    return 0;
}
```

5. **(8 pts.) Debugging**. Billy Bruin had to write a program that would scan through the interior elements (i.e. ignoring the first and last element) of an integer array, take the average of each element with its neighbors on the left and right, and then count if the *previous* and *current* average were both at or above a threshold. The user will start by entering the size of an integer array, **n** (max 100) elements, and then a threshold, **thresh**, (as a double). They will then enter the **n integers** for the array at which point the program should perform the desired computation illustrated in the diagram below.

   Find and correct the errors in the code below (next page or in the linked skeleton file). You may only **CHANGE** a line, **NOT ADD** lines of code (there is no need). You may **NOT alter the approach**, but must correct the approach given. Simply cross out the line or part of the line and write the correct code next to it (if hand annotating) or update the line with the correct code (if editing the .cpp skeleton file online).

   There are 8 lines of code that need to be changed.
   You may compile and run this code to help in debugging if you like. Be careful not to spend too much time chasing bugs!



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

   Feel free to use any debugging methods you like to find the errors (you can run and compile this code)

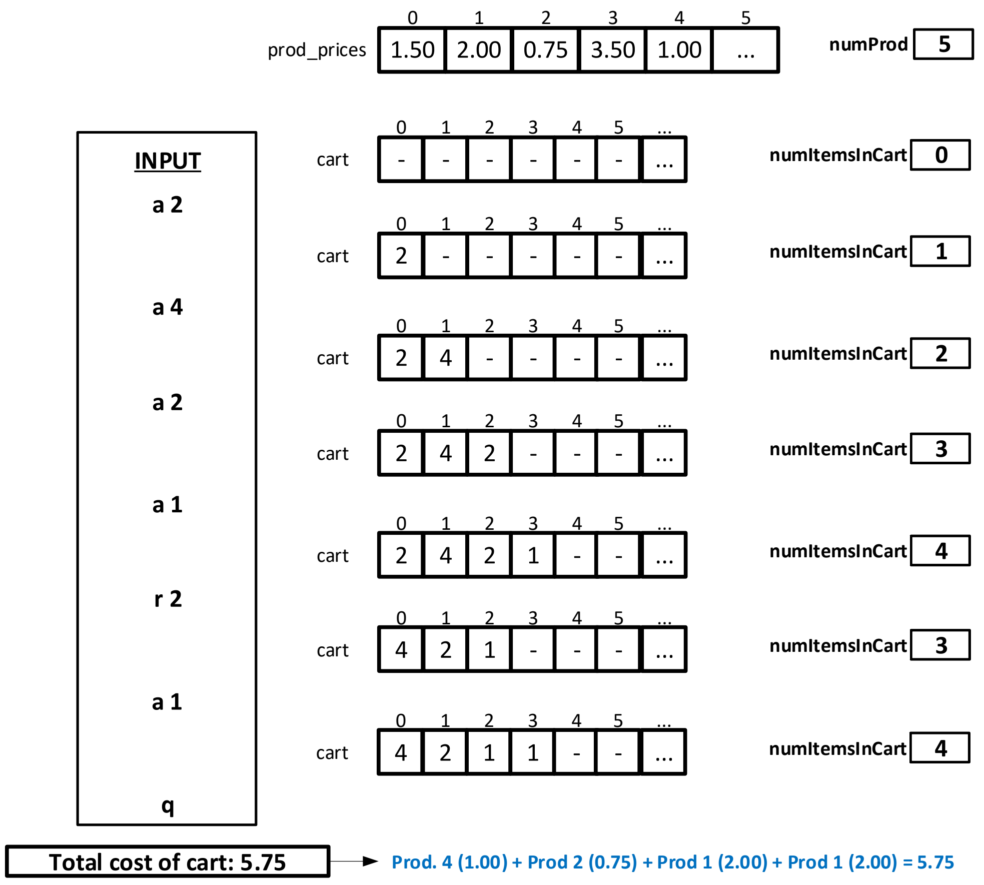
```
#include <iostream>
using namespace std;

void avg3(int x, int y, int z);  // **Error: return type should be double

int main()
{
    int in[100];
    bool flag = false;
    double thresh;
    int n;
    int cnt;             // **Error: should be init to 0
    cin >> n >> thresh; // read in size of array and threshold
    // now read in values into the array
    for(int i=0; i < n; i++){
        cin >> in[n];   // **Error:  should be in[i]
    }

    // now loop through inputs to take average of neighbors
    // and find consecutive averages at or above threshold
    for(int i=0; i < n; i++)  // **Error: for(i=1; i < n-1; i++)
    {
        double result = 0;
        avg3(in[i-1], in[i], in[i+1]);  // ** Error: result = avg3(in[i-1], in[i], in[i+1]);

        // if result is lower than the threshold, reset the flag
        if(result < thresh){
            flag = false;
        }
        if(result >= thresh && flag == false){
            // since flag is false, the previous value must be under the threshold
            flag = true;
        }
        if(result >= thresh && flag == true){   // ** Error: should be 'else if'
            // since flag is true, previous value must have been above/equal to threshold
            cnt++;
        }
    }
    cout << "Final count: " << cnt << endl;
    return 0;
}
void avg3(int x, int y, int z)              // **Error: return type should be double
{
    double result = (x + y + z) / 3;        // **Error: should divide by 3.0 (or cast to double)
    return;                                 // **Error: return result;
}
```

6. **Coding (9 pts).** Complete the following program for a simple online shopping application where users can add and remove products from their cart and then compute the total cost of the cart by **filling in the missing parts of the code**.

The program will start by asking the user how many products (i.e. **numProd**) they wish to enter and then read in **numProd** prices (doubles) into the array: **prod_prices**. Then the user will use commands: **a** to add an item or **r** to remove an item from the cart **followed by the product ID** (i.e. the index of the product's price in the prod_prices array) until they enter **q** to quit. For example, if the user enters **a 2** then the program should add product 2 (which costs 0.75) to the cart array. When the user types in the command **q**, you should quit the program and output the total cost of the items currently in the cart. You may assume product ID's entered by the user exist (i.e. are between 0 to numProd-1). If the user tries to remove a product item, remove only the 1st occurrence of that item (not all occurrences). Also, if the item to remove doesn't exist, the program should output an error message. Below is a sample execution of the program.

The program below has some missing elements. Complete the blanks (add code) to make the program function as desired. You may only fill in the blanks shown and cannot add or alter other code. A sample execution is shown below.

The skeleton file on the next page can be downloaded with the link: cart.cpp. You may use a compiler.



```
#include <iostream>
using namespace std;
int main()
{
    double prod_prices[100]; // product prices
    int cart[100];           // products in cart
    int numItemsInCart = 0;  // number of items in cart
    int numProd;             // number of products
    cin >> numProd;

    for(int i=0; i < numProd; i++){  // Read in the product prices array
        cin >> prod_prices[i];
    }

    char cmd = '-';  int prod;
    cin >> cmd;
    while( ___cmd != 'q'____ ) {
        cin >> prod;
        if(cmd == _ 'a'______ ) {
            cart[numItemsInCart] = prod;
            numItemsInCart += ___1__________;
        }
        _else if__ (cmd == __'r'______) {

            bool itemRemoved = ___false__;
            for(int i = 0; i < numItemsInCart; i++) {
                if(cart[i] == ___pro.d______ ){
                    for(int k=i; k < numItemsInCart-1; k++) {
                        // Shift items up one slot to remove item i

                        _cart[k] = cart[k+1]__;
                    }
                    numItemsInCart += _-1____;

                    itemRemoved = __true____;
                    break;  // break from i loop
                }
            }
            if(itemRemoved == false) { cout << "Item was not in cart" << endl; }
        }
        cin >> cmd;
    }
    double cartCost = 0.0;
    for(int i=0; i < numItemsInCart; i++){
        int p = __cart[i]__;

        cartCost += ___prod_prices_[ p ];
    }
    cout << "Total cost of cart: " << cartCost << endl;
    return 0;
}
```