Jerald Jacobs | jacobj7 | 661541559

OPERATIONS.CPP

Bugs found from simply reading through the code:

1 [Line 39] The multidivide function would cause an error because it would always return an int instead of a float which will not give us the value we want. If you operate on two ints, it cannot become a float, so we must cast the numerator as a float to get a float answer.

```
float multidivide(int numerator, int d1, int d2, int d3, int d4) {
  float f = ((((numerator / d1) / d2) / d3) / d4);
  return f;
```

2 [Line 110] Error checking on command line arguments should be checking if argc != 2, not if it does equal 2. Having the if condition be argc == 2 would mean that it would always print an error when we put the right number or arguments. Change to if(argc != 2).

3 [Line 121] To check if a file can be opened correctly, we should check if(infile). If it cannot be opened then we should check if(!infile). The if statement is checking if(infile) which does not do what we want because it prints out the error if that file DOES exist, we only want it to print that error if it cannot open file. Change to if(!infile).

4 [Line 226] The second nested for loop should not be iterating the same ptr that is in the first loop or else the first array wont be properly iterated through. The second loop should be iterating the through the second array (not the first) so we change it to increment tmp_ptr2 (refers to the inside array) instead of tmp_ptr (outside array).

```
for(int x = 1; x <= size; ++x, ++tmp_ptr) {
    int* tmp_ptr2 = *tmp_ptr;
    for(int y = 1; y <= size; ++y, ++tmp_ptr) {
      int tmp = *tmp_ptr2;
```

5 [Line 269] The function is supposed to return false if any element in v1 is SMALLER than an element in v2. The if statement condition is a logic error because it does the opposite of what we want it to do. We don't want it to set success = false unless v1[i] is smaller than v2[i]. To fix this, we simply change the > sign to a <= sign.

```
for(uint i=0; i<v1.size(); ++i) {
    if(v1[i] > v2[i]) {
      success = false;
```

6 [Line 282] This loop is supposed to add values 1-10 to v2, but instead it adds the values 0-9 (logic error). The easy fix to this bug is to simply push back i+1 to the vector instead of just i.

7 [Line 368] The if statement is a logic error because we don't want to see if i is divisible by 3, we want to see if the VALUE at position i in the all vector is divisible by 3. To fix this we just change the if statement to check if(all[i] % 3 == 0).

```
for(uint i = 0; i < all.size(); i+1) {
    if(i % 3 == 0) {
      // std::cout << all[i] << " is divisible by 3" << std::endl;
      counter++;
      threes.push_back(i);
```

8 [Line 371] This a very similar problem as before (logic error). We want to add the value at position i to the new vector, not i itself. So again, we just change i to all[i] when we push it back.

9 [Line 172] The if statements need to change to see if the fracpart == 0 instead of writing an expression fracpart = 0. The if statements need a bool comparison, not an expression. Fixed by checking if(fracpart == 0).

10 [Line 178] Same exact problem as the previous error.

Bugs found from reading the terminal warnings:

11 [Line 181] The function is supposed to return -1 if it cannot find the third Pythagorean triple and it does not, so we need to add it at the very end so that this function always returns something. It would have given us an error if it failed to find a third triple (assigning nothing to a L value).

12 [Line 367] Must have been a typo. Having the loop do (i+1) at the end of each "iteration" doesn't actually modify i (it's just a value), so we would get an infinite loop. To properly iterate through a loop, you should increment i by using ++i or i++. Fixed by changing it to i++.

13 [Line 378] Since i is an unsigned int, the condition of the for loop will never yield true because you can't decrement an unsigned int from 0. The solution to this bug is to change the data type of i to a regular int, not an unsigned int.

```
for(uint i=counter-1; i >= 0; --i) {
    std::cout << "threes[" << i << "] = " << threes[i] << std::endl;
```

Bugs found from checking through the assert fails:

14 [Line 199] It took me awhile to find this one. I kept getting the assert fail array[1][2] equal -1 and I thought the problem was in my pythagoras function but after I put a std::cout at the beginning of the function, I realized that the program didn't even make it to the function. I inserted multiple std::couts throughout the code until I realized that it never made it into the loop. Then I noticed that the for loop condition was wrong, it should be x<size, not x>=size. It never got into the first iteration of the loop because the condition returned false.

```
for(int x=1; x>=size; ++x) {
    for(int y=1; y>=size; ++y) {
      array[x][y] = pythagoras(x, y);
```

15 [Line 200] Same exact problem as the previous error.

16 [Line 176] Assert showed us a test case in which x can be greater than y. This will give us an error if we try to take the square root of the difference in squares, so we must wrap it in an absolute value function. Change to abs(y*y - x*x).

```
float diffsquares = y*y - x*x;
  fracpart = modf(sqrt(diffsquares), placeholder);
  if((fracpart = 0))
    return (int) *placeholder;
```

17 [Line 256] Assert showed us that the original vector did not change when we called vector_sum(). This is because it made a copy of the vector when we used it as the parameter. To actually modify the vector, we must pass the vector by reference. Change parameter to &inVec.

```
int vector_sum(std::vector<int> inVec) {
  for(uint i=0; i<=inVec.size(); ++i) {
    inVec[i] = inVec[i] + inVec[i-1];
```

18 [Line 310] Noticed that counter did not always equal 4 when it should have. Counter needs to be initialized to equal 0. To fix this, change it to int counter = 0;

Bugs found from segmentation faults / bus errors:

19 [Line 166] The pythagoras function used a pointer to placeholder, but it makes more sense to just allocate memory for it on the stack as its own double. The rest of the function uses it as being a double instead of a pointer to a double, so we don't need to deference it. Also, in place of the pointer to the double, we reference the double instead for the modf function. Change all instances of "*placeholder" to "placeholder" and change all instances of "placeholder" to "&placeholder". It fixes the bus error: 10.

20 [Line 190] I would keep getting a seg fault (pinpointed via couts) at line 227. I realized that the tmp_ptrs were starting at position 0 in the array and not 1. And at line 190, we can see that we don't actually assign any values at position 0 in both arrays. To fix this we have to have both x start at 0 instead of 1. Also it sould iterate while x<size (not x<=size). Out of bounds access.

```
const int size = 25;
  int** array = new int*[size];
  for(int x=1; x<=size; ++x) {
    array[x] = new int[size];
    for(int y=1; y<=size; ++y) {
      array[x][y] = 0;
```

21 [Line 192] Same exact problem as previous error, except that we do this for y.

22 [Line 257] I kept getting a segmentation fault when we called vector_sum(), so I checked it out and it's obvious to see that i must start at 1 and not 0 since the function accesses position i-1. The condition also must be < and not <= or else it will access a position that is one over the actual vector. These cause out of bounds access errors. To fix change to i=1 and i<inVec.size().

```
int vector_sum(std::vector<int> inVec) {
  for(uint i=0; i<=inVec.size(); ++i) {
    inVec[i] = inVec[i] + inVec[i-1];
  }
  return inVec[inVec.size()];
```

23 [Line 260] Similar problem with out of bounds access error. The function is trying to return the element that is one position past its last element. The last position is not at inVec.size(). Fix this by changing it to inVec.size()-1.

Bugs found by outputting the data to the iostream:

24 [Line 50] In arithmetic_operations, int e is actually equal to 36 instead of 32, so it was edited to equal 32 by changing the equation to 4*c instead of 5*c.

```
int a = 10;
int b = 46;
int c = 4;
int d = c - b;              // -42
int e = b - 3*a + 5*c;      //  32
int f = 2*b + 2*c;          //  100
int g = e - (b/c) + d + 20; // -1
int h = (f/c) / a;          //  3
int m = (d / h) / 7;        // -2
int n = g + m;              // -3
int p = (f / e) - h;        // -1
int q = f + 2*d;            // 16
int r = g + m + p + n;      // -8
float s = a / f;            //  0.1
```

25 [Line 52] After fixing e, g also showed to be incorrect. It did not equal 3 because it does not round up two ints (25/10). To fix this, the value should be multiplied by 1.0 to make it 2.5 (instead of 2) and then round(2.5).

26 [Line 56] After fixing g, p was also incorrect, so it was corrected to equal -1.

27 [Line 58] Same deal with r. r was off by 1 so we need to correct it.

28 [Line 59] Float s was equal to 0 and not 0.1 since operating two ints returns an int. To fix this, we just cast one of them as a float.

29 [Line 365] I noticed that when it printed the threes vector, it would actually print four random values outside of the actual threes vector. I realized that this is because a) the print loop used started at counter-1 and b) the counter was not reset to 0 before counting how many threes there are. Easy fix to this is to reset counter = 0.

TRIANGLES.CPP + TRIANGLES.H

Bugs found from simply reading through the code:

30 [Line 77 .cpp] The length_between() function asks for the parameters in the order x1,y1,x2,y2, but all of the calls to this function ask for the parameters in the order x1,x2,y1,y2. This would create a logic error because the length between the points calculated would not be the correct value. Fix this by changing it to the proper order.

```
double length_between(int x1, int y1, int x2, int y2) {
  return sqrt((x2-x1)*(x2-x1) + (y2-y1)*(y2-y1));
```

31 [Line 56 .cpp] This if statement only tests if sides a and c are close enough. This will result in a logic error because it would fail the isosceles condition if a and b are close enough or b and c are close enough. To fix this you have to check all 3 sides with each other, not just a and c.

```
else if(close_enough(a,c)) {
    return "Isoceles";
```

Bugs from checking through the assert fails:

32 [Line 57 .cpp] The second triangle never seemed to pass the "Isosceles" test, so I checked out the get_type() function and noticed that they misspelled "Isosceles". To fix it, spell it properly.

33 [Line 28 .cpp] It kept failing the get_area() tests and then I realized that the return statement did not match the actual Heron Formula (logic error). It's missing the sqrt so to fix this simply sqrt the final answer.

Bugs found by outputting the data to the iostream:

34 [Line 26 .h] I used the Triangle::print() function to print the triangles points after the triangle was constructed and I noticed that the coordinate points did not match what appeared in the operations.cpp file, so I double checked the constructor again and noticed a mismatch in the triangles.h file. y1 and y3 were swapped. To fix this swap them back.

```
Triangle(int x1_, int y1_, int x2_, int y2_, int x3_, int y3_,
      std::string name_) :
 x1(x1_), y1(y3_), x2(x2_), y2(y2_), x3(x3_), y3(y1_), name(name_) {}
```