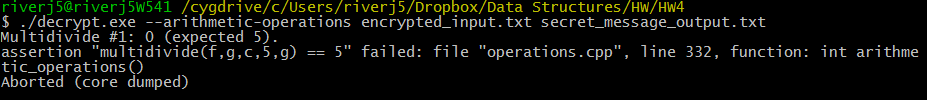
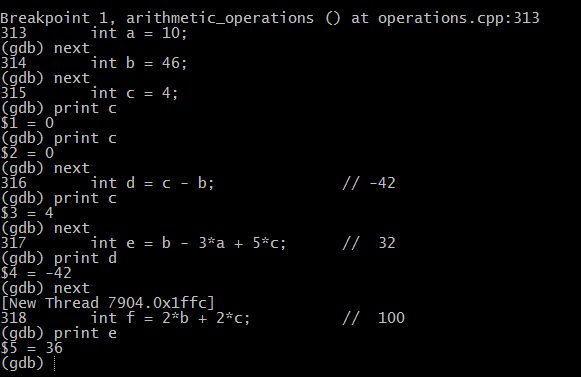
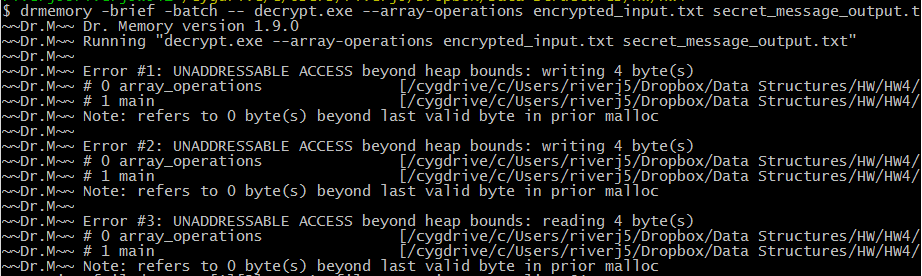
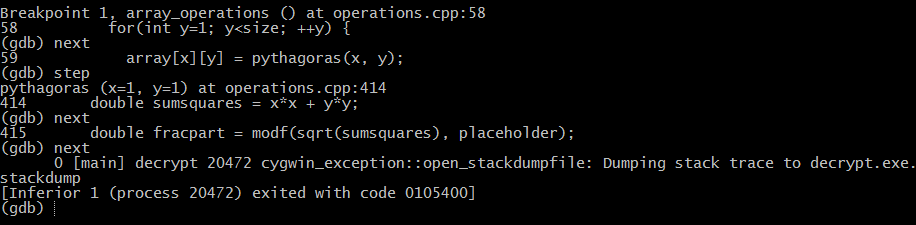
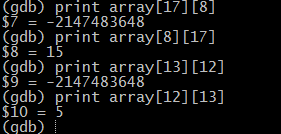
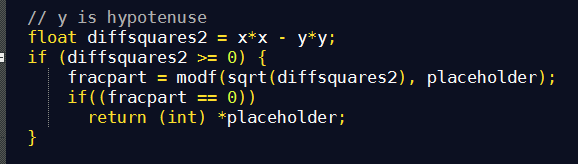
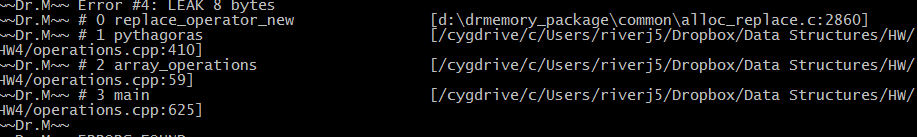
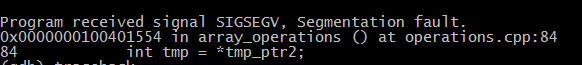
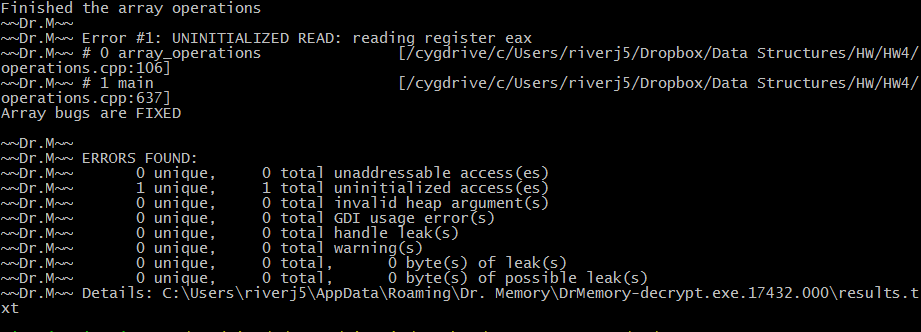
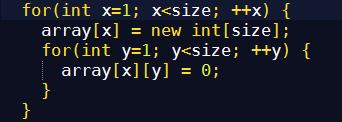
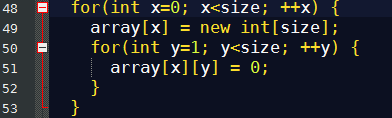
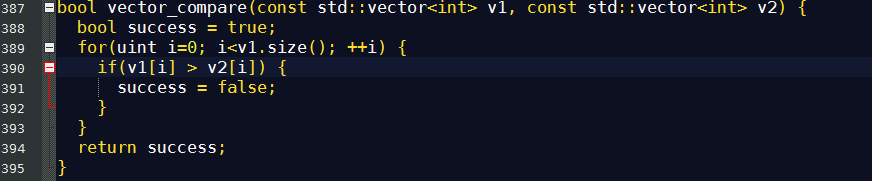
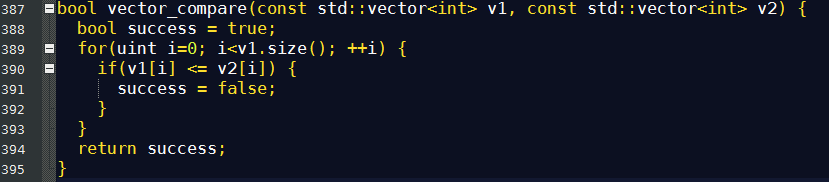
Arithmetic\_operations

* 
  + assertion "multidivide(f,g,c,5,g) == 5" failed, instead giving me 0
  + 
    - Using print, we see that e is supposed to be 32 but it is 36
    - Went to line in code that initialized e, change it from https://i.gyazo.com/19afe3b8d4f102b5639d869f166b9d66.png to https://i.gyazo.com/2e5b1a29ac554dce276c145c00be74de.png
    - Fixing e fixed g, b/c was dependent on on e
* Same with assert multidivide(840, d, m, p, 1) == -10
  + Started with fixing h (which was supposed be 3 but was instead 2) using same debugging method.
    - Before change: 320 int h = (f/c)/a
    - After change: 320 int h = int(ceil((float(f)/float(c))/float(a)));
      * https://i.gyazo.com/83d6e355854a6e6e67c1e90aff46a971.pngThen gave me this
      * It says d3 = 0, put d3=p which is supposed to be -1.
        + Changed from https://i.gyazo.com/7b57610328978281f11fb58ee5e99f3d.png
        + to https://i.gyazo.com/a967a37b5badedbc711e3f295860f90b.png

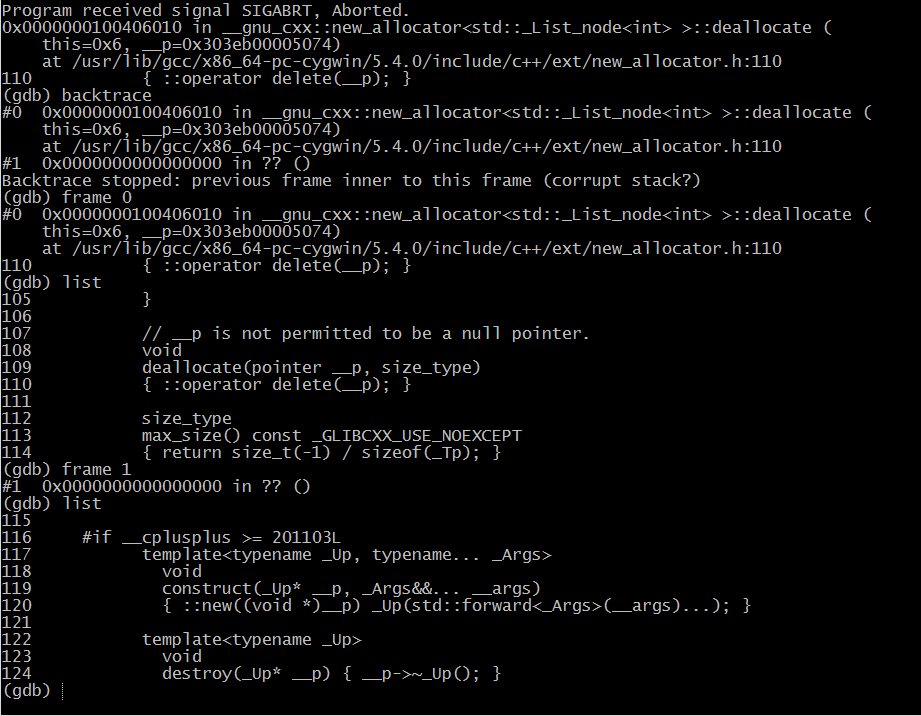
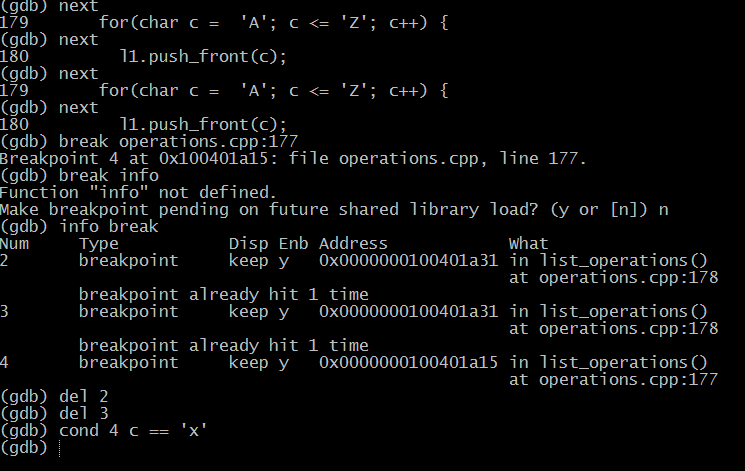
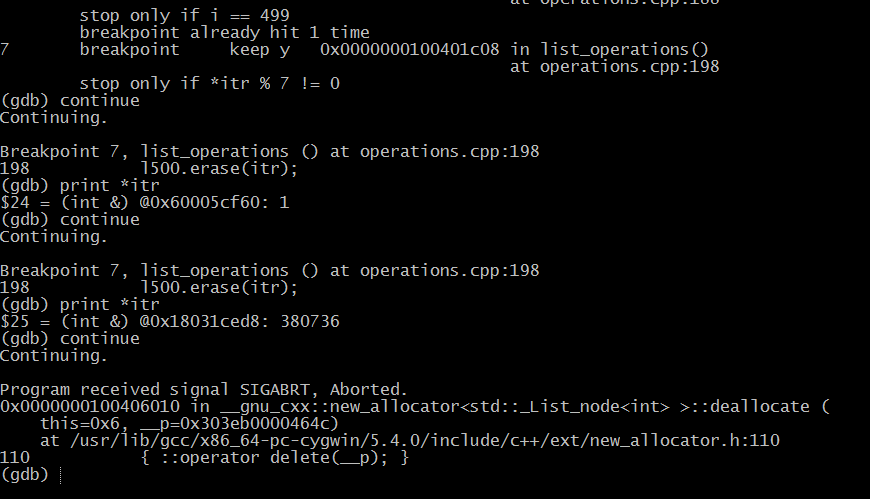
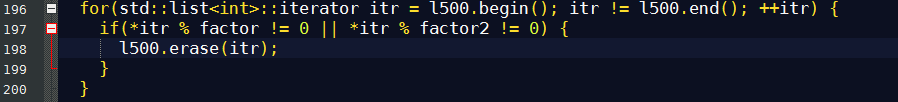
Array\_operations

* 
  + Line 48: for(int x=1; x<=size; ++x) {
  + Line 49: array[x] = new int[size];
    - Will assign one over the max range as new int[size]
      * Change line 48: for(intx = 0; x<size; ++x) {
  + Line 50: for(int y=1; y<=size; ++y) {
  + Line 51: array[x][y] = 0;
    - Will go over max range
      * Change line 50: for(int y=1; y<size; ++y) {
* https://i.gyazo.com/7c9964ab983e5e1cf979cbed231e019b.png
  + Go to where values are assigned (line 58 – 62)
    - for(int x=1; x>=size; ++x) {
    - for(int y=1; y>=size; ++y) {
      * X and y usually wont start off as bigger than size and if they are, it will be an infinite loop.
        + Change the >= to < for both cases.
    - Made Pythagoras function return -1 at the end, or else no value on array will ever be -1.
* 
  + Dr.memory was giving me uninitialized access errors and large amount of leaks
  + Program crashes right when modf is used.
    - I made place holder allocate space for double, so then it can be used to store modf value.
    - Also changed if ((fracpart = 0)) to if((fracpart == 0)) b/c I noticed it
* 
  + Numbers that should give the same answers are dependent on order. This is the case only for those that include the hypotenuse as one of the inputs to the Pythagoras function call.
    - Added a case for when y is hypotenuse right after the x is hypotenuse case: 
    - Debugger than said that assertion “array[1][2] == -1” failed even though it was working before, which means now its going into code that I just wrote instead of skipping to return -1. I did the calculations with my calculator with x = 1 and y = 2. Saw that I was trying to take square root of negative number. So then I added the if (diffsquares1 >=0) right before we use modf for the x is hypotenuse case and (diffsquares2>=0) for the y is hypotenuse case.
* 
  + In Pythagoras, instead of returning (int) \*placeholder, I assigned a variable to this value, and deleted placeholder, and then returned the value so no memory leaks coming from Pythagoras. Also put a delete statement before return -1.
* 
  + Verified that tmp\_ptr had correct values
  + Figured out that row and columns of index 0 were ignored in the rest of code, so they weren’t all assigned values. So tmp was assigned \*tmp\_ptr which may not have been a valid value. When pointer tmp\_ptr and tmp\_ptr2 were first initialized, they were pointing to index 0 of row/columns. So after they are initialized, I incremented the pointers (lines 80 – 84) so they would start pointing at index 1.
* 
  + One more error at line 106, something that we can’t edit.
  + https://i.gyazo.com/7a65da329006d0e720711effb5f8ce85.png
    - Array[0] is trying to read something that was never initialized. So I changed to 
    - This initializes every spot in the array, even if we’re not going to use array[0]

Vector Operations

* V1sum == 175 assertion failed. Went into vector\_sum function and returned inVec[inVec.size()-1] instead of inVec[inVec.size()] because that index would be out of range.
* Also made for loop start with i = 1 to get rid of segmentation fault because if I = 0, then it would try to access index -1 of inVec
  + It worked
* Assertion v1[2] == 75 failed
  + Put a break point right after calls to the function vector\_sum that should have changed. Print the values of the array in the gdb. Saw that they were all 25, which means that function wasn’t changing the values.
    - Made the function argument a pass by reference (also had to be done when it was first initialized.
* Assertion v2sum == 55 failed, means that v2 was initialized wrong b/c vector\_sum was already verified to have worked.
  + Changed https://i.gyazo.com/77e248e10c938567c307d607942341b7.png to https://i.gyazo.com/a2e39d69a1f4235ec833c7818e71921f.png
* Assertion counter == 4 failed.
  + Set a breakpoint right before assertion
    - Printed v2 values to make sure they were correct
      * They were, which means counter was the problem. Printed counter was -13203, which made no sense following logic of loop. Which means it was uninitialized memory. I set counter = 0 when it was first initialized so it can increment upon the right number.
* Assertion\_compare(v1, v4) failed
  + I went into compare function and saw faulty logic.
    - Changed  to 
* https://i.gyazo.com/c2fe042eb0b61cd55dc84f9d030385d3.png
  + Watched counter. Saw that counter is changed within the loop, which means that counter wasn’t reset to 0 before it was reused. Set counter to 0 before loop.

List Operations

* 
  + In this case backtrace leads me to files that aren’t my own
  + I decide to check whether the lists were initially made correctly
  + I go into loop and keep hitting next, but its taking me too long. So I learn how to set a conditional break: 
  + 
    - Continue with conditional break method and come across 196 loop problems. First of all, it’s erasing the wrong values, not the ones that are a multiple of 7 or 11. Also, it ends up on a number that is not part of the list, probably because it’s accessing memory that it shouldn’t be due to the fact that after using erase, the iterator is invalid. It starts to skip some values in the list.
    - Changed  to 