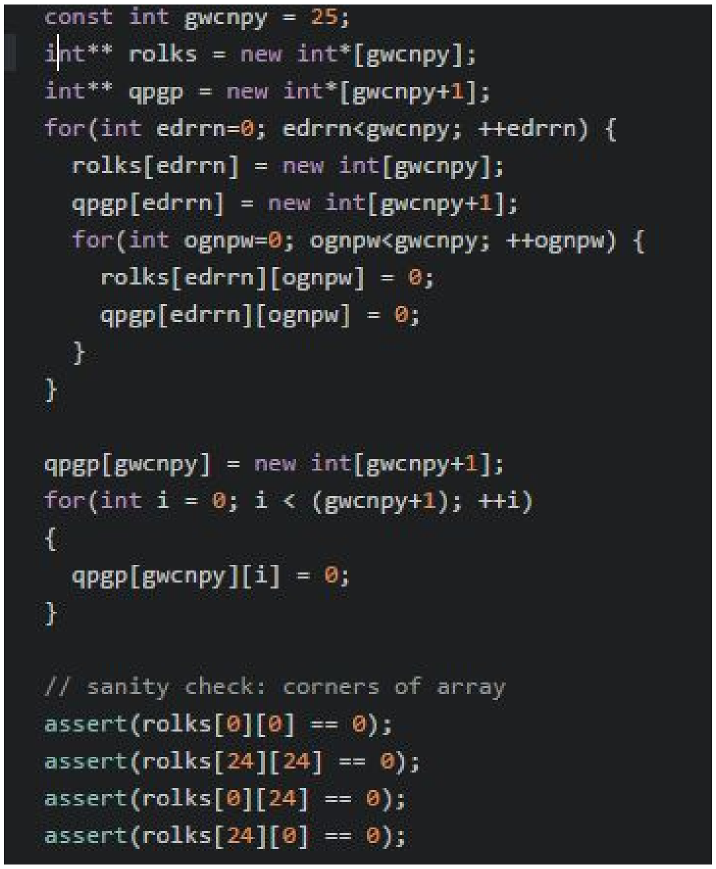
While debugging the code I took a very systematic approach; I tackled each operation to completion before continuing to the next. Arithmetic and file operations had very trivial resolutions while array and list operations were much more involved. Therefore I majority of the bugs I'll be discussing will be on arrays and lists.

One of the more difficult arrays of bugs I had to tackle was within the "--array-operations" aspect of the homework. When compiling, I got an initial run error of "Segmentation fault (Core Dumped". I started by running gdb, setting breakpoints at the for loops as well as set watch points for the incremental values to assess the actual values I was getting vs the expected values. What I noticed was that the for loops were going to completion but near the ending I'd receive a segmentation fault. The issue was that it was trying to assign 0 to an index that was not created yet. Instead of keeping the rolks[edrrn][ognpw + 1], I took the +1 out and changed it to the other array, qpgp. Then wrote another for look where it created the edge case memory for where the final value was assigned. My fix was a good one because made it very simple to just add in the corner case. I learned the functionality of "watch" in gdb. It made tracking values easier and more consistent than writing by hand (which I did initially and I did make minor errors). Another thing I initially tried was debugging without really understanding the code. One thing I took away from this code was don't trust the names of variables. Instead, ignore them and understand pure functionality. This was achieved by actually reading comments and then using gdb to backtrace and find better approximations of where segmentation faults occured.

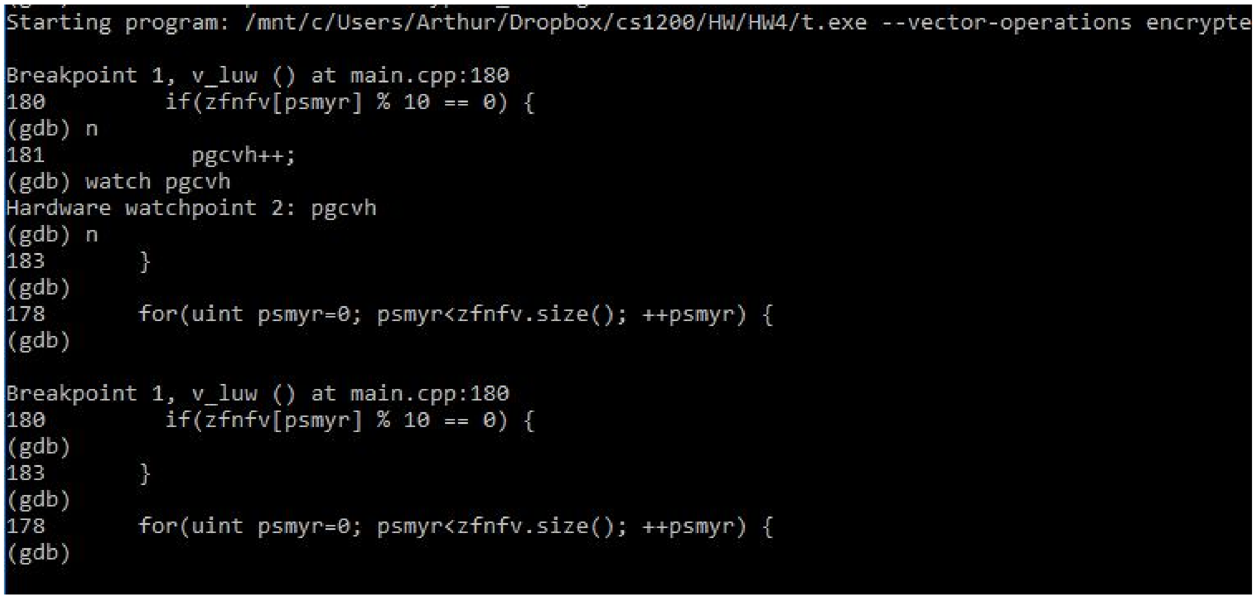


```
const int gwcnpy = 25;
int** rolks = new int*[gwcnpy];
int** qpgp = new int*[gwcnpy+1];
for(int edrrn=0; edrrn<gwcnpy; ++edrrn) {
  rolks[edrrn] = new int[gwcnpy];
  qpgp[edrrn] = new int[gwcnpy+1];
  for(int ognpw=0; ognpw<gwcnpy; ++ognpw) {
    rolks[edrrn][ognpw] = 0;
    qpgp[edrrn][ognpw] = 0;
  }
}

qpgp[gwcnpy] = new int[gwcnpy+1];
for(int i = 0; i < (gwcnpy+1); ++i)
{
  qpgp[gwcnpy][i] = 0;
}

// sanity check: corners of array
assert(rolks[0][0] == 0);
assert(rolks[24][24] == 0);
assert(rolks[0][24] == 0);
assert(rolks[24][0] == 0);
```

Notice the asserts at the end. Without formally using a debugger, I found that index's of [-1] (that were initially written) were not correct (C++ is not that smart). To fix this I figured out the bounds of the 2D dynamic array and did that for every edge case which was a success.



```
Starting program: /mnt/c/Users/Arthur/Dropbox/cs1200/HW/HW4/t.exe --vector-operations encrypte

Breakpoint 1, v_luw () at main.cpp:180
180         if(zfnfv[psmyr] % 10 == 0) {
(gdb) n
181           pgcvh++;
(gdb) watch pgcvh
Hardware watchpoint 2: pgcvh
(gdb) n
183       }
(gdb)
178       for(uint psmyr=0; psmyr<zfnfv.size(); ++psmyr) {
(gdb)

Breakpoint 1, v_luw () at main.cpp:180
180         if(zfnfv[psmyr] % 10 == 0) {
(gdb)
183       }
(gdb)
178       for(uint psmyr=0; psmyr<zfnfv.size(); ++psmyr) {
(gdb)
```

Around line 472 (pythagorean function), there were also more minor errors. For just safety, I figured that setting the double value, tinbfb that held the integer part of modf, to 0 so it doesn't accidentally store a garbage value. Forward, when calling modf, the second argument was the variable to store the value (hypotenuse). In order to pass the value into the variable it must be passed by reference (&). This was a part of the previous homework, Matrix, so it stood out immediately. Also, say the arguments were (4,3) instead of (3,4). In this case, in order to fix it, you must call abs(int x) on the (x^2 -y^2) part of figuring out side lengths of the triangle so square rooting the difference didn't result in an imaginary number. Lastly in this function there was a semantic error in the return of the function. This issue was simply resolved by changing "return 0" to "return -1". The way I fixed these were the most efficient because abs is a simple fix instead of writing another case to swap the values around. Furthermore based on the assert I found the semantic error "return 0" should have been return -1.

In "--list-operations", line ~536, there were errors that were, although difficult to find, quick fixes. Namely, adding a decrement for eouij in the if statement. This is necessary because when removing elements from the list you must also lower the count in the list so the loop does not go out of bounds. The issue was found because I kept getting out of indexing issues (segmentation fault). What I learned from debugging this is there is a difference between post incrementing vs pre incrementing. To achieve debugging this I traced the code using pen and paper and finally realized there was an issue within the incrementing.

When it came to memory debugging, I found that in my 2D dynamic array configurations that there were uses of the **new** keyword in order to create the dynamic memory. Without memory debugging I realized that at some point I need to delete just as much. Hence I wrote a loop to write the looped memory allocation and a delete to handle the 2D pointer. Not much debugging was necessary but I found that from the previous homework that my eye always catches the **new** operator as well as corresponding **delete []** operators.

Another problem with the code was a pass by copy vs pass by reference. This error was in a function prototype in line 29. When looking at the function declaration I found that the comments included the detail to pass the value stored into the parameter. This immediately signifies that pass by reference is necessary. I initially noticed the problem when the function was called to edit vectors in lines 167-169 and nothing was changing so I was getting assert issues. In order to adjust it I changed the function prototype parameter to pass by reference and adjusted the actual definition of the function lower. Reading comments in this code is a godsend to figure out issues without completely breaking the overall objective of the code is what I learned from specific examples like this.

One issue that took a lot of moving around was type casting in the “--arithmetic-operations”. The one in lines 121-126. The comment specified a multi-division operation to a number. In order to fix this I nested a bunch of floats to correspond to every step of the division. I found this error when there were assert issues. To solve it I kept moving around the float casts and making sure that parentheses were properly corresponding to each cast. The lesson from this was that parentheses matter and following parenthese will make debugging far easier.

Another minor issue that I encountered is opening the infile outfile in the “--file-operations”. Although this was a small error it made me believe that I actually had more code wrong than I really did. In order to fix this I followed the link to make sure that that I was opening the file properly. My issue was that there was the condition “!=” when it should have been ==. The original way it was written basically means that if it opens correctly then throw an error. This was the simplest fix because I changed one character and didn’t add any more.

In total, doing this homework actually taught me a lot about debugging especially since I couldn’t go off of variable names. Debugging through gdb and through hand written tracing really helped. Furthermore this gave me a more thorough grasp of debugging methods and really how these concepts all work.