Data Structures Homework 4 Debugging Report - Zachary Ward

Overall development environment
Operating System: Windows 10 Home
Compiler: WSL/g++
Computer hardware: 16.0 GB Ram, Intel(R) Core(™) i7-7700HQ CPU @ 2.80GHz, x64-based processor

INITIAL WARNINGS:
Upon compiling, a series of warnings are produced:

```
main.cpp: In function ‘int oohtxl()’:
main.cpp:214:50: warning: for increment expression has no effect [-Wunused-value]
   for(uint wzufh = 0; wzufh < lhzzn.size(); wzufh+1) {
                                                  ^
main.cpp: In function ‘int ym_ven(int, int)’:
main.cpp:120:1: warning: control reaches end of non-void function [-Wreturn-type]
 }
 ^
main.cpp:110:20: warning: ‘lyon’ is used uninitialized in this function [-Wuninitialized]
   float ldeb = modf(sqrt(ufqv), lyon);
                    ^
main.cpp: In function ‘bool fxaur(int, char**, char*&, int&)’:
main.cpp:64:31: warning: ‘w_fwh’ may be used uninitialized in this function [-Wmaybe-uninitialized]
   char* yadsy = new char[w_fwh];
```

For the first warning, wzufh is simply not being incremented correctly. Instead of being reassigned a value that is one greater than its current value, there expression “wzufh + 1” is present with no reassignment taking place. The fix for this is simple: change “wzufh+1” to “wzufh+=1.”This change allows for reassignment to take place and in turn allowing incrementation to occur.

For the second warning, a return statement is not being reached in the function; this is because the condition in the if-statement which contains the return statement is invalid. Instead of “==”, the boolean operator, being in the if statement, the assignment operator “=” is present. A simple fix would be to add an extra “=” sign, allowing the boolean expression in the if-statement to correctly execute. However, there is another issue. When there is no solution, -1 is supposed to be returned but never is. This is a simple fix: by adding the statement “return -1” at the end, this warning is gone.

For the third warning, ‘lyon’ is not being initialized because it is declared as a double* and passed in as an argument into the modf function. The modf function works by passing a value into its second parameter (which would be lyon), however a value cannot be passed into the variable if it is declared as a pointer which isn’t even pointing to anything. The ideal fix would be to instead declare lyon as a double and to pass &lyon into the modf function instead of just lyon. This would allow for the integral part of whatever is passed into the modf function to be correctly stored into the double variable lyon.

For the final warning, w_fwh is initialized after an array is created of size w_fwh. This needs to be fixed, and can be fixed easily be simply initializing w_fwh prior to the creation of the array. I implemented the fix by moving the lines of code initializing w_fwh above the creation of the char array in the “fxaur” function.

FIRST ERROR:

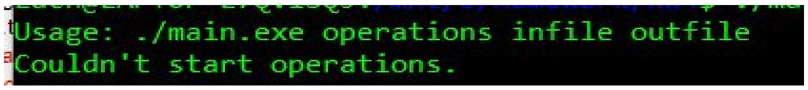
Upon running decrypt.exe with the command line arguments" --arithmetic-operations encrypted_message.txt secret_message_output.txt," the following error message is presented:

```
main.exe: main.cpp:439: int xewag(): Assertion `gomh(xqis,lqfdzq,aelex,5,lqfdzq) == 5' failed.
Aborted (core dumped)
```

We can decipher that the error is not necessarily just occuring in the xewag() function, but more specifically, could be occurring within the usage of the gomh() function in the xewag() function. Using gdb, I stepped through the program, starting at the top of the xewag() function and stepping into the gomh() function once i hit the assertion statement. I noticed that the division would always return an integer, so i casted the numerator variable to a float in the division. Stepping through the xewag() function, I suspected that the variables were initialized to incorrect values. I confirmed this suspicion by printing out the values of the variables and comparing them to the comments stating what the values of the variables should be. After manipulating the math behind how the variables were initialized, i managed to correct this error in the xewag() function.

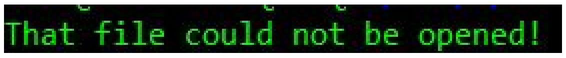
SECOND ERROR:

Upon using the "--file-operations" command line argument with all other command line arguments kept constant, we are pleasantly greeted by a few minor errors before we are presented with the major one.



```
Usage: ./main.exe operations infile outfile
Couldn't start operations.
```

With the correct usage, we are given this message. To fix this error, I used gdb to step through bool fxaur() until I found where the logic error was. This message was being produced whenever argc==4, but since we want argc to be equal to 4, i changed this statement to argc!=4 in the if statement. This fix solved the error only to lead me to another:



```
That file could not be opened!
```

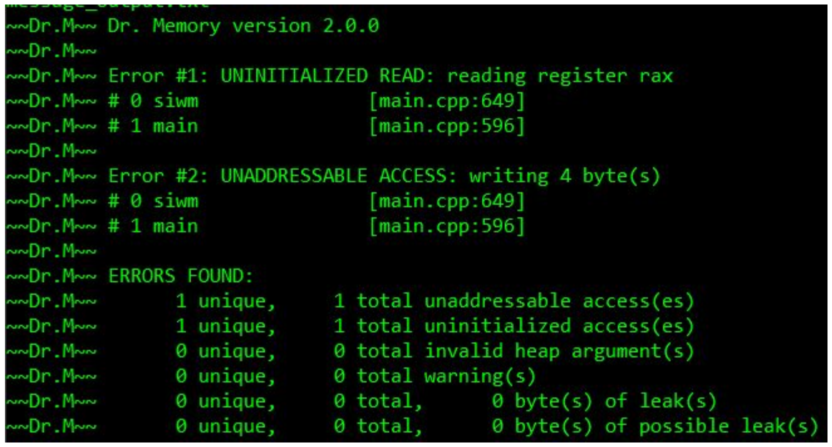
The program still was terminating for some reason. I did not need to use gdb however to notice that this was because of a similar logic error of not including a not(!) operator where one should be present. Fixing this bug led me to the real error:

```
Successfully opened the input file.
Successfully read in 77 bytes of data.
main.exe: main.cpp:80: bool fxaur(int, char**, char*&, int&): Assertion `nciy.gcount() != w_fwh' failed.
Aborted (core dumped)
```

Stepping through bool fxaur() with gdb, I decided it would be a good idea to print out the values of nciy.gcount() and w_fwh and noticed that they were equal. I observed that this would always be the case because w_fwh also represents the size of the array of bytes which is created in the function, and so i changed the != in the assertion statement to ==, making the assertion pass every time and allowing the function to work properly.

THIRD ERROR:

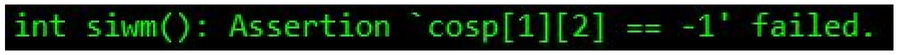
Upon using the "--array-operations" command line argument, we are greeted by a segmentation fault, leading me to believe that there could be errors in either the siwm() or jitf() functions, which are called when we perform our array operations.



```
~~Dr.M~~ Dr. Memory version 2.0.0
~~Dr.M~~
~~Dr.M~~ Error #1: UNINITIALIZED READ: reading register rax
~~Dr.M~~ # 0 siwm                [main.cpp:649]
~~Dr.M~~ # 1 main                [main.cpp:596]
~~Dr.M~~
~~Dr.M~~ Error #2: UNADDRESSABLE ACCESS: writing 4 byte(s)
~~Dr.M~~ # 0 siwm                [main.cpp:649]
~~Dr.M~~ # 1 main                [main.cpp:596]
~~Dr.M~~
~~Dr.M~~ ERRORS FOUND:
~~Dr.M~~       1 unique,     1 total unaddressable access(es)
~~Dr.M~~       1 unique,     1 total uninitialized access(es)
~~Dr.M~~       0 unique,     0 total invalid heap argument(s)
~~Dr.M~~       0 unique,     0 total warning(s)
~~Dr.M~~       0 unique,     0 total,      0 byte(s) of leak(s)
~~Dr.M~~       0 unique,     0 total,      0 byte(s) of possible leak(s)
```

After running the code through Dr. Memory, I was pointed towards an unaddressable access error and an uninitialized access error in the siwm() function. After stepping through the siwm() function with gdb and printing out the values of the iterators in the for loops, I noticed that the arguments of the for loop needed to be changed so that points outside of the array would not be called, resulting in an out of bounds error. More specifically, the <= signs in the second argument of the for loops needed to be changed to < signs: this error is ubiquitous throughout the siwm() function. I also removed line 649: cosp[itph+1][dqezp+1] = 0; because it served no purpose, unnecessarily reinitializing some values and going out of bounds in the process.

While this took care of all of the memory errors, we are now presented with an assertion failure:

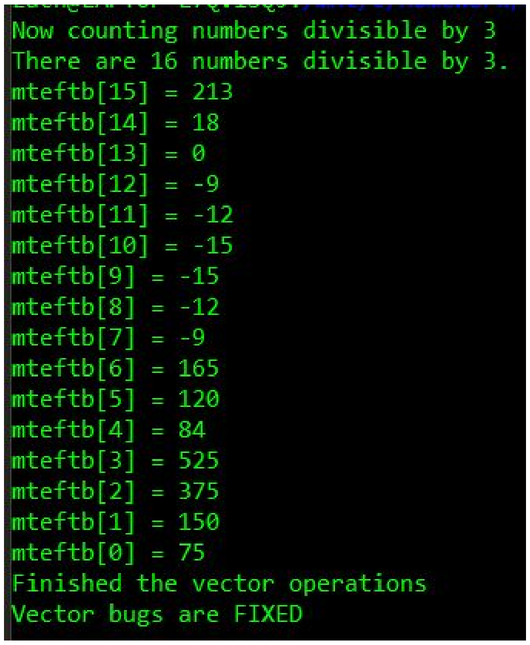


```
int siwm(): Assertion `cosp[1][2] == -1' failed.
```

The function responsible for resetting all of the values of the cosp array is the ym_ven() function which will return the third number in a pythagorean triple (only if the arguments you passed are part of a triple in the first place). I noticed a mathematical error: absolute value needed to be involved in a line which it previously was not. When checking whether subtracting the squares of the passed in arguments and taking the square root has no remainder, the possible negative result of the subtraction was not accounted for. Putting absolute value around “sfekx*sfekx - m__dqh*m__dqh” seemed to fix this because now square roots of negative values weren’t being taken.

FOURTH ERROR:

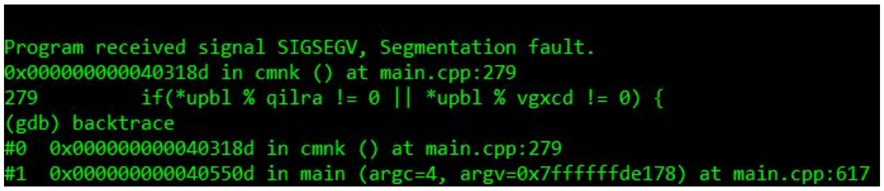
Upon using the” --vector-operations” command line argument, we are yet again greeted by a segmentation fault, and the errors were just as obvious. After fixing all segmentation faults and assertion failures, we are presented with the hard part of debugging the vector operations: figuring out why the program is telling us that the “Vector bugs are NOT fixed.” After looking at int main(), I deduced that this message is produced only when the oohtxl() function is returning the incorrect value. Stepping through oohtxl() with gdb, I noticed firstly that when checking for numbers divisible by three in the lhzzn vector, the loop iterator wzufh was being compared with mod 3 instead of lhzzn[wzufh], resulting in the wrong value of zvat, the variable being returned, as it is incremented every time a number divisible by 3 is found. Fixing this mistake in the code however did not lead to a successful run. The next step was checking whether the integers tbox and ubowha had the correct values. They were initially set up to be the sum of all elements in the hfzvzc and enk_ vectors, respectively. After stepping and re-stepping through the oohtxl() function for hours, I found no errors, checking whether vectors were being initialized with the right values and whether certain integer variables had the correct values. I came to realize that the assertions would have probably told me if anything was set up incorrectly: that is what they are there for, after all. When I came around to the loop which printed out mteftb, the vector that had all elements in lhzzn divisible by 3, I noticed that mteftb.size() and zavt, the variable used to keep track of how many number are divisble by 3, had different values. The problem was now clear. Previously, zavt was used to represent something else, and now that it was going to represent the number of elements in lhzzn divisble by 3, its value had to be reset to 0. After adding this fix, the function oohtxl() ran flawlessly.



```
Now counting numbers divisible by 3
There are 16 numbers divisible by 3.
mteftb[15] = 213
mteftb[14] = 18
mteftb[13] = 0
mteftb[12] = -9
mteftb[11] = -12
mteftb[10] = -15
mteftb[9] = -15
mteftb[8] = -12
mteftb[7] = -9
mteftb[6] = 165
mteftb[5] = 120
mteftb[4] = 84
mteftb[3] = 525
mteftb[2] = 375
mteftb[1] = 150
mteftb[0] = 75
Finished the vector operations
Vector bugs are FIXED
```

FIFTH ERROR:

Upon using the “--list-operations” command line argument, we are greeted by an assertion failure. This was easy to fix simply by flipping a >= to a <= in the first for loop initializing the first list. I noticed something very interesting, however. When looking at the loop which pushes all capital letters to the front of the list in the beginning of the cmnk() function, a segmentation fault was being created whenever all of the capital letters were being what seemed to be correctly pushed to the front of the list. This segmentation fault was particularly confusing; I had no idea where it was coming from. Running the function through gdb and using the backtrace command, I was enlightened.

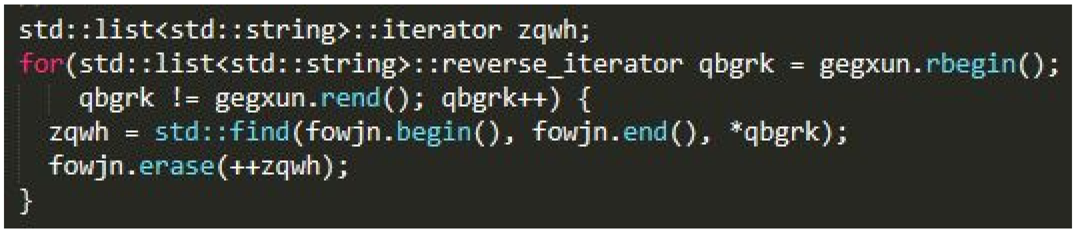


```
Program received signal SIGSEGV, Segmentation fault.
0x000000000040318d in cmnk () at main.cpp:279
279         if(*upbl % qilra != 0 || *upbl % vgxcd != 0) {
(gdb) backtrace
#0  0x000000000040318d in cmnk () at main.cpp:279
#1  0x000000000040550d in main (argc=4, argv=0x7fffffffde178) at main.cpp:617
```

It became clear that whenever a spot in the list was erased, the iterator also had to be reset or else a spot out of bounds would be called during the next iteration. Therefore, I added the statement “upbl = nixv.begin()” at the end of the if statement and the segmentation fault was gone. The final error was:

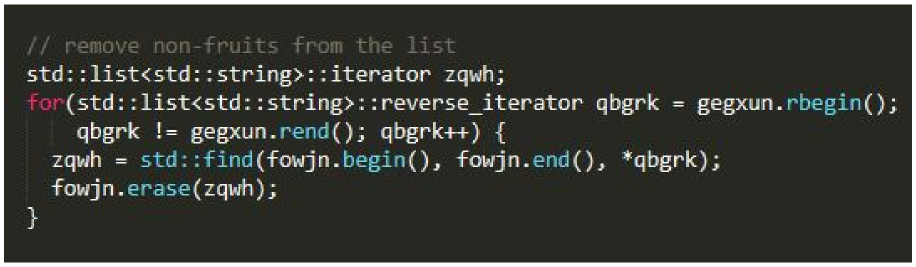
main.exe: main.cpp:353: int cmnk(): Assertion `*zghc_v.begin() == 'A'' failed.
Aborted (core dumped)

Using gdb, I stepped through the cmnk() function and noticed the block of code:



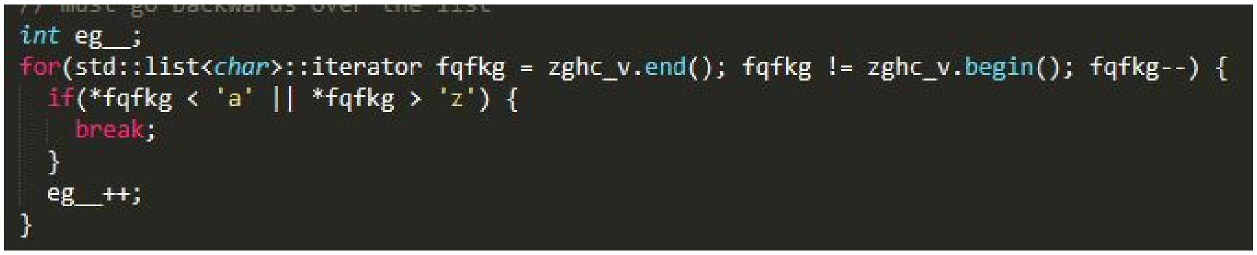
```
std::list<std::string>::iterator zqwh;
for(std::list<std::string>::reverse_iterator qbgrk = gegxun.rbegin();
    qbgrk != gegxun.rend(); qbgrk++) {
  zqwh = std::find(fowjn.begin(), fowjn.end(), *qbgrk);
  fowjn.erase(++zqwh);
}
```

Immediately when i reached the erase statement in gdb, it became clear to me that it was zqwh itself that needed to be erased, not ++zqwh. The statement ++zqwh is responsible for incrementing zqwh but does not refer to zqwh itself, therefore it does not make sense to return ++zqwh and would make much more sense to return zqwh.
THE FIX:



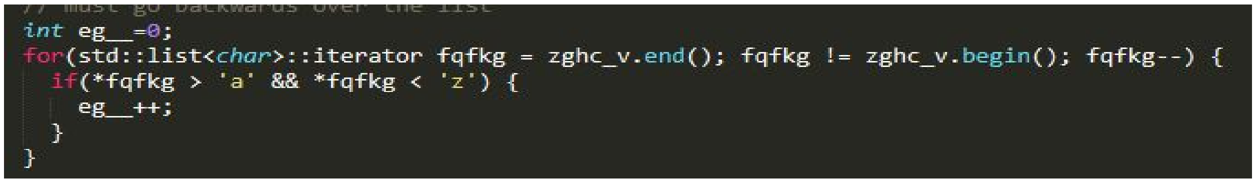
```
// remove non-fruits from the list
std::list<std::string>::iterator zqwh;
for(std::list<std::string>::reverse_iterator qbgrk = gegxun.rbegin();
    qbgrk != gegxun.rend(); qbgrk++) {
  zqwh = std::find(fowjn.begin(), fowjn.end(), *qbgrk);
  fowjn.erase(zqwh);
}
```

Although this one fix did not fix the assertion error, it brought me one step closer to the fix as the correct non-fruits were now being successfully removed from the list. To finally solve my problem, I stepped through the function in gdb until I got to the following block:



```
// must go backwards over the list
int eg__;
for(std::list<char>::iterator fqfkg = zghc_v.end(); fqfkg != zghc_v.begin(); fqfkg--) {
  if(*fqfkg < 'a' || *fqfkg > 'z') {
    break;
  }
  eg__++;
}
```

Whenever I printed out the value of eg__, I got garbage. I realized that this was because eg__ was not initialized to 0, and so the first part of my fix was to set eg__ to 0. Stepping through this loop, i realized that fqfkg was progressing from end to beginning just fine, leading me to believe that the logic inside the loop was horribly wrong. This loop is supposed to count how many lowercase letters are in fqfkg, and this loop was currently incrementing eg__ whenever an uppercase letter was hit until a lowercase letter was hit(then it breaks). To fix this, I first got rid of the break statement, switched the < and > signs, changed the || to and && and incremented eg__++ within this if statement, like so:



```
// must go backwards over the list
int eg__=0;
for(std::list<char>::iterator fqfkg = zghc_v.end(); fqfkg != zghc_v.begin(); fqfkg--) {
  if(*fqfkg > 'a' && *fqfkg < 'z') {
    eg__++;
  }
}
```

The reason this fix works is because the for loop is now first checking if the current character is lowercase (correctly, being within ‘a’ and ‘z’), and then proceeds to increment eg__++. With this fix, the program now runs flawlessly and all errors are gone!