

Name: Jiahong Ji

RCS: [jjj2@rpi.edu](mailto:jjj2@rpi.edu)

RIN: 66184836

Couse and section number: CSCI 1200-05

## Debugging feedback essay

Generally, this sample program is a disaster. The variables name inside the function are just some random typed things. Probably the reason why this program work is that the programmer first write all the program in a normal way. Then, he or she replace whole variable names inside the program into some random-typed things. Things the names of those functions and variables literally indicate nothings, reading this program is extremely difficult. However, the good news is that our TAs or professors left some comments inside the program, which can help us significantly. Those comments give us the directions and told us which way should we go to. Without the comments, we probably could not make it.

My operations system is Windows 10, and the compiler I am using is called Atom. When I was trying to debug, I realize that I good thing about the Atoms is that it can split one program into two windows and edit on either of them will change the program itself at the same time. This is extremely useful when we are trying to read a function inside a function. With that function, we do not have to go backward and forward several times and forget where we are. The program I use to compile the program is WSL, which probably is most of the student's choice. Also, the debugger tool I am using for debugging is gdb. Before I did this homework, I am not familiar with it because the first and only time I use it is in the Lab3. Nonetheless, after this assignment, I become more familiar with it since it is a powerful tool which enable us to tracing the program and those variables inside the program. If you don't know what cause the program core dumped or what is going on with those variables, gdb is a smart choice. The below examples is a few example of how I debugging though out this program.

```
~Dr.M~n~
~Dr.M~n~ Error #1: LEAK 70 bytes
~Dr.M~n~ # 0 replace_operator_new_array      [/drrmemory_package/common/alloc_replace.c:2929]
~Dr.M~n~ # 1 vjxol                             [main.cpp:610]
~Dr.M~n~ # 2 main                             [main.cpp:716]
~Dr.M~n~
```

The first I want to talk about for this program is the memory leaking issue. In the PDF, it said that this is not memory leak in the main program. However, if we actually run it, there are a few memory leaking issue. To, fix it, we need to go though the program, look at which line of this program cause the leaking happened. After I doing it, I find out that the file buffer array is a new\*, which suppose to be deleted since it is the thing in the heap. So, I add delete []file\_buffer, in line 758, which solve the memory leaking issue.

Glancing though the pdf, I knew that I should run the arithmetic operations first. So, I just typed accordingly, and the first bug pop up, which is below

```
225D.exe: main.cpp:225: int aomko(): Assertion `!xqu(aqmtlg,dvyftr,onmikl,5,dvyftr) == 5' failed.
```

```
Aborted (core dumped)
```

After I checking it, I find out this bug is happened in the Lxqu function Line 225. The failed assertion caused core dumped. So, I go check those variables value and change them to the value that we want.

Also, in the variables like Zavt int / int, there is a calculating error. I just fixed them all accordingly

Also in the last case, the printed out result didn't match {Multidivide: 0 (expected 0.1).} which should be 0.1.

So, I go into the function and check them. And I realize that all the input in lxqu function are int. To make to make it match the result. We should turn all the variable into float in the lxqu function to make it precise.

Keep going, the following bug showed up.

```
D.exe: main.cpp:251: int aomko(): Assertion `!_fhjf(pwa_, zavt)' failed.
```

```
Aborted (core dumped)
```

So, I go to line 251 and looking at the function. I find out the Variable type are different. So, I changed the data type of pwa\_ and zavt type to float. After that. The first case was passed. Whole the case 1 test are passed. The illustrations above are a few examples of how I fixing the bug in the athematic operations.

File operations

Type accordingly, it said.

```
Usage: ./D.exe operations infile outfile
```

```
Couldn't start operations.
```

Track it to line 582, it said we should not take four arguments. However, in the pdf, it said that we suppose to take four arguments as the input. So, I just change the assert accordingly.

Continue run it, it return That file could not be opened!

Go to the line 591, we find out that we need to change the if statement to enable us run the program since its origin means that if the condition is false, we could not run it.

In the next, it said that

D.exe: main.cpp:616: bool vjxol(int, char\*\*, char\*&, int&): Assertion `r\_igv.gcount() != cvbjc' failed.

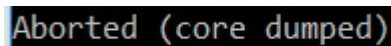
This means that the assert should be equal to make it right.

After that, we fixed all the operation bugs.

Then, is the array operation. The line 511 and 512 are using the index of -1 for the array txhz, which could potentially cause some problem happened. So, we changed those index to the size of the array to make the operation success.

Also, the rarq function is the most tough. It use some functions which we never seen before. So, a good way to know what is going on is go to google and search for those function to get to know what is actually going on among those code. After, we checked it on the C++ reference, we realize that those variables types should be consistent to make those operation works.

Continue tracking it, we go to the array operation.



```
Aborted (core dumped)
```

Continue tracking it, we realize that the line 501 cause **core dumped**. Also, by looking at the code, all the index in it start from 1, which usually suppose to start from 0. So, we also change it accordingly.

```
// sanity check: corners of array
assert(txhz[1][1] == 0);
assert(txhz[1][-1] == 0);
assert(txhz[-1][1] == 0);
assert(txhz[-1][-1] == 0);

// store outlagaraga numbers in txh
```

In line 508,509, they are using -1 as the index of the array. By searching it on google, I find out that -1 is not a valid index label in C++. So, I change it accordingly.

```
(core dumped)
```

After that, we go to the Vector operation. If we run the vector operation directly, **core dumped** happened directly and nothing will be print out. So, we use gdb to help us track the program and kind of let us know what is actually going on. By using it to track the program, we realize that the core dumped happened in the function uemufi. So, I looking at it carefully and realize that the are several index out of range error and logic error in this function. Like what I did before, I fixed all the bugs over here accordingly.

Keep going, we find out that the **assert at line 73,74 failed**. By checking the local variable, we find out that all the elements in the vectors has not been changed yet. As the result, we to add a reference sign in the uemufi function to make the origin content in the vector changed.

Also, in line 97, it said the **assert for titrg == 4 failed**. By checking the number of value of titrg over here, we realize that titrg is a super big number over, which suppose to be four over here. By simply staring at the code, we realize that the reason why this happened is because of that the integral has not been initialize yet. So, it go to some ridiculous number.

```
(core dumped)
```

Later, in line 158, **the output variables are creepy and cause some core dumped** issue happened at the end. By using backtracing in the gdb feature, we realize the for above are causing some trouble. So, we just fixed them accordingly. I saw that they are just storing the ordered number into to the array but not the content of the vector that we want to store. So, we changed the content of it store it into an new array.

Later, the Assert in line 359 failed, which cause the core dumped happened again. This remind me that the offgq array is not right because the begin of this array should be an capitalized A. So, I trace it back to the place where I create it. The comments remind me that I should created an array which A~Z is in the front. So, change it accordingly, and the assertion passed.

After I passed that, the weirdest thing happened. Every assert in it has passed, but the weird huge number is not correct. So, I glance though every list that are used to create the number and make sure everything is created correctly and accordingly. Since, this is called logic error or algorithm even the

powerful tool like gdb or Dr memory cannot help me get through it. Suddenly, I realize that when we erase something from an array, we need to -- the iterator to make it work. This is what the previous lab told me. So, I also change the function accordingly, and everything worked just fine!

The above illustrations are a few examples of how I debugged for this complex program. Besides the techniques of debugging, I also learned that when we are writing our own code, comments and proper variables and function names are extremely important since it can reduce the workload of other code-readers.