Data Structure Homework 4
Bug Report

Zhengneng Chen, chenz11@rpi.edu, 661409204

February 24, 2018

Overall development environment

Windows Subsystem Linux with Ubuntu 16.04.3 LTS.
gcc version 5.4.0 20160609 (Ubuntu 5.4.0-6ubuntu1 16.04.9)
Intel(R) Core(TM) i7-7700K CPU @ 4.20GHz
Installed memory (RAM): 16.0 GB

BUG #1: Arithmatic error

1. Reproduction

For a slice of code in function $wieh()$, we have

Listing 1: A few lines of function wieh()

```
int mnuwcf = 10;
int blgrn = 46;
int vfpdky = 4;
int ssmxiz = blgrn - 3*mnuwcf + 4*vfpdky; //  32
```

If we change the equation of variable $ssmxiz$ to

```
int ssmxiz = blgrn - 3*mnuwcf + 5*vfpdky; //  32
```

as the same as original code, we would reproduce this error that value of $ssmxiz$ is not 32 as suggested. Any numbers other than 3 and 4 in equation would make $ssmxiz$'s value to be incorrect. The error is recognized when using

```
--arithmetic-operations encrypted_message.txt secret_message_output.txt
```

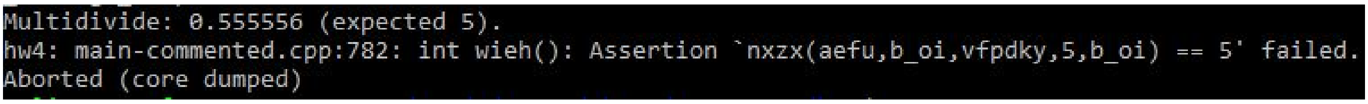
as arguments that encrypted_message.txt is the input file.

2. Erroneous behavior

At assertion below, compiler would abort the program due to assertion failure

```
assert(nxzx(aefu,b_oi,vfpdky,5,b_oi) == 5)
```

3. Exact text of error messege and exact output



```
Multidivide: 0.555556 (expected 5).
hw4: main-commented.cpp:782: int wieh(): Assertion `nxzx(aefu,b_oi,vfpdky,5,b_oi) == 5' failed.
Aborted (core dumped)
```

We expect the result return from function $nxzx()$ is 5 but in fact we have 0.555556.

4. Debugger info

Since the value returned from $nxzx()$ is wrong, I set a breakpoint there to check the correctness of its parameters and found that the value of b_oi is 3 instead of suggested -1. Then another breakpoint is set where b_oi is initialized and we look up the locals variables' values to check what affect the assignment of b_oi. Here we can see that the problem is

$$ssmxiz = blgrn - 3 \cdot mnuwcf + 5 \cdot vfpdky = 46 - 3 \cdot 10 + 5 \cdot 4 = 46 - 30 + 20 = 36 \neq 32$$

The correct calculation is suppose to be

$$ssmxiz = blgrn - 3 \cdot mnuwcf + 4 \cdot vfpdky = 46 - 3 \cdot 10 + 4 \cdot 4 = 46 - 30 + 16 = 32$$

which is coherent with the number suggested in comment. Hence, the problem is the error of arithmatic when initializing $ssmxiz$ which affect the initialization of b_oi that determine the correctness of return value of function $nxzx()$.

BUG #2: Logic error in assert

1. Reproduction

For last few lines in function $nvti()$, there is a assertion saying that

Listing 2: Last assertion in function nvti()

```
vxseib.read(arrf, xcrdft);
assert(vxseib.gcount() == xcrdft);
```

Error emerges if we change $==$ sign to $!=$ as originally shown in buggy version of code

Listing 3: Last assertion in function nvti()

```
assert(vxseib.gcount() != xcrdft);
```

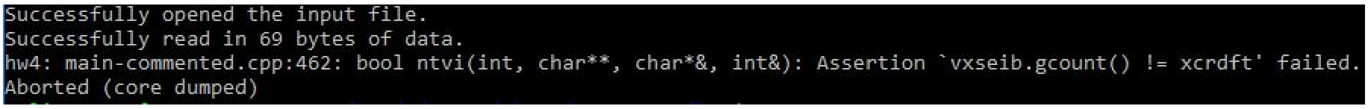
with arguments

```
−−file −operations encrypted_message.txt secret_message_output.txt
```

2. Erroneous behavior

Program is aborted due to failure of assertion.

3. Error message and expect output



```
Successfully opened the input file.
Successfully read in 69 bytes of data.
hw4: main-commented.cpp:462: bool ntvi(int, char**, char*&, int&): Assertion `vxseib.gcount() != xcrdft' failed.
Aborted (core dumped)
```

4. Debugger usage and error analysis

Since this is a assert failure, I set breakpoint on that line and check the value of variable $xcrdft$ and the value returned from the calling of $vxseib.gcount()$. They appear to be the same value but assertion said that they are not equal. I was confused for a long time because it didn't seems reasonable that assertion itself has a logic error. By looking up into manual of function $gcount()$, I realized that value which $gcount()$ would returned is determined when calling $read()$ function. So I looked up the manual of function $read()$ in terminal. The manual says that $vxseib$'s gcount is set by the second argument passing into $read()$ call. That is, gcount are suppose to be the same value as $xcrdft$. So there is indeed an error in assertion. Although it looks weird at first, I understand that codes are wrote by person and if there is a bug in code, it could be in anywhere without except. When we debug for a program, we should not have a white list in mind for the place where a bug might located at.

BUG #3: Error of accessing array with operator[]

1. Reproduction

Let's see 4 lines of assertions in function $e\_lf()$

Listing 4: Assertions in function e_lf() to check corners of array

```
assert(txhz[0][0] == 0);
assert(txhz[0][lyon-1] == 0);
assert(txhz[lyon - 1][0] == 0);
assert(txhz[lyon - 1][lyon-1] == 0);
```

To reproducing errors, we access array with

```
assert(txhz[-1][1] == 0);
assert(txhz[-1][-1] == 0);
```

with arguments

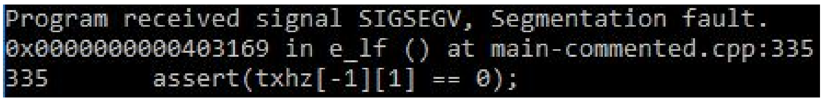
```
--array-operations encrypted_message.txt secret_message_output.txt
```

2. Erroneous behavior

Segmentation fault.

3. Error message and expect output

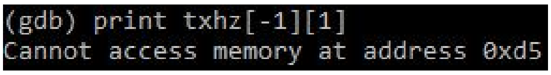
Since terminal does not give any useful information but a segmentation fault, I used gdb to debug it and have a error message: I expect it to show nothing but pass these asserts. However, there is a segmentation fault instead.



```
Program received signal SIGSEGV, Segmentation fault.
0x0000000000403169 in e_lf () at main-commented.cpp:335
335       assert(txhz[-1][1] == 0);
```

4. Debugger info

If I print $txhz[-1][1]$ in gdb, I was told that I cannot access the memory of $txhz[-1][1]$. So there must be a problem at the value we used in [] to access array. The problem is we cannot use -1 as the reference of last element of an array in C++.



```
(gdb) print txhz[-1][1]
Cannot access memory at address 0xd5
```

BUG #4: Error of passing function's parameter by reference/value

1. Reproduction

Let's have a look on the declaration of function $rnh\_()$

Listing 5: Declaration of function rnh_()

```
int rnh_(std::vector<int>& zimwqb);
```

If we remove the $\&$ from this line and implementation part

```
int rnh_(std::vector<int> zimwqb);
```

we would reproduce this error with arguments

```
--vector-operations encrypted_message.txt secret_message_output.txt
```

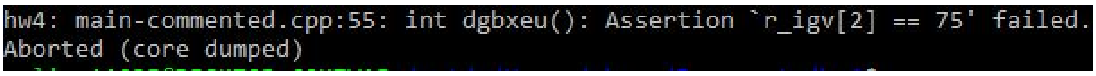
2. Erroneous behavior

Program is aborted due to failure of assertion.

```
assert(r_igv[2] == 75);
```

3. Error message and expect output

We expect the value of $r\_igv[2]$ to be 75 but it is not so that the assertion failed



```
hw4: main-commented.cpp:55: int dgbxeu(): Assertion `r_igv[2] == 75' failed.
Aborted (core dumped)
```

4. Debugger info

Because the program didn't failed at the assertion for variable $edrrn$, we know that the return value of function $rnh\_()$ is correct. By setting a break point inside of function $rnh\_()$, we can see that the array is correctly summed up. However, if a vector passed into $rnh\_()$ is processed correctly but we find error in caller function, we can tell that vectors are not changed because it is not passed by reference. Looking at the declaration of function $rnh_()$, our guess is conformed that vector is passed by value into function $rnh\_()$. So all we need to do is add $\&$.

BUG #5: Error of order of list when calling push_front()

1. Reproduction

In function $vyxc()$, we add chars to list $vixu$ with

Listing 6: Pushing capitalized char to list

```
for(char gaxfgp = 'Z'; gaxfgp >= 'A'; gaxfgp--) {
  vixu.push_front(gaxfgp);
}
```

To reproduce this error, suppose we add capitalized char to list with

```
for(char gaxfgp = 'A'; gaxfgp >= 'Z'; gaxfgp++) {
  vixu.push_front(gaxfgp);
}
```

Then we would fail the assertion that the beginning of char list is $'A'$ with arguments

```
--list-operations encrypted_message.txt secret_message_output.txt
```

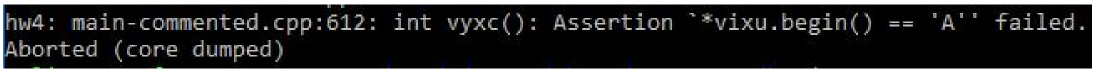
2. Erroneous behavior

Program is aborted due to failure of assertion.

```
assert(*vixu.begin() == 'A');
```

3. Error message and expect output

We expect the beginning of lsit $vixu$ to be $'A'$ but we got



```
hw4: main-commented.cpp:612: int vyxc(): Assertion `*vixu.begin() == 'A'' failed.
Aborted (core dumped)
```

4. Debugger info

By setting break point on the line of failed assertion and printing the beginning of list, we found that the beginning of list is $'Z'$. Then the capitalized chars must be in reverse order. Hence we look back to the place where capitalized chars are assigned and found that chars are pushed to front from $'A'$ to $'Z'$. As a result, $'Z'$ is the last one to be pushed to front so that Z is the beginning of list that we had a list with reverse order of desired order. To fix it, we suppose to push to front from $'Z'$ to $'A'$.

BUG #6: Memory leak

1. Reproduction

We notice that in function $nvti()$, we are going to read a file. To help us read in file, we set up a integer variable $xctdft$ to measure the size of file and a char pointer $arrf$ at the size of $xctdft$ as read-in buffer.

```
vxseib.seekg(0, vxseib.end);
int xcrdft = vxseib.tellg();
vxseib.seekg(0, vxseib.beg);

char* arrf = new char[xcrdft];
```

To reproduce this leak, we initialize $xcrdft$ without assign it with any value and use it to initialize $arrf$. And then call $tellg()$ function to assign value for $xcrdft$:

```
int xcrdft;

char* arrf = new char[xcrdft];

vxseib.seekg(0, vxseib.end);
xcrdft = vxseib.tellg();
vxseib.seekg(0, vxseib.beg);
```

with arguments

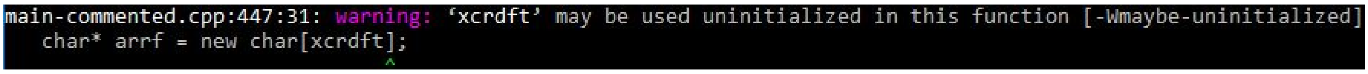
```
--file-operations encrypted_message.txt secret_message_output.txt
```

or

```
--all-operations encrypted_message.txt secret_message_output.txt
```

2. Erroneous behavior

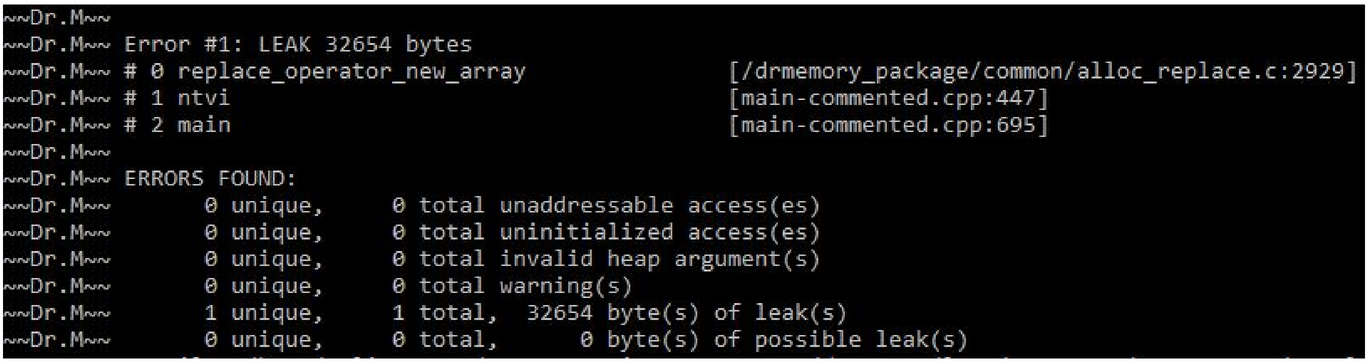
Compiler would give us a warning since we use a uninitialized variable. But this warning does not affect the execution of program and its correctness.



```
main-commented.cpp:447:31: warning: ‘xcrdft’ may be used uninitialized in this function [-Wmaybe-uninitialized]
   char* arrf = new char[xcrdft];
                               ^
```

3. Error message

This leak does not affect the execution of program. We can only visualize it in Dr.Memory. Error message from Dr.Memory shows that



```
~~Dr.M~~
~~Dr.M~~ Error #1: LEAK 32654 bytes
~~Dr.M~~ # 0 replace_operator_new_array               [/drmemory_package/common/alloc_replace.c:2929]
~~Dr.M~~ # 1 ntvi                                     [main-commented.cpp:447]
~~Dr.M~~ # 2 main                                     [main-commented.cpp:695]
~~Dr.M~~
~~Dr.M~~ ERRORS FOUND:
~~Dr.M~~       0 unique,     0 total unaddressable access(es)
~~Dr.M~~       0 unique,     0 total uninitialized access(es)
~~Dr.M~~       0 unique,     0 total invalid heap argument(s)
~~Dr.M~~       0 unique,     0 total warning(s)
~~Dr.M~~       1 unique,     1 total,  32654 byte(s) of leak(s)
~~Dr.M~~       0 unique,     0 total,      0 byte(s) of possible leak(s)
```