Tianxin Zhou
zhout3@rpi.edu
02/24/2017

Bug report

For data structure hw4

1. Mismatched value for variables in arithmetic-operations.

Overall development environment:

MacOS Sierra 10.12.3, G++ version: 4.2.1.

Command that produce the problem:

./a.out --arithmetic-operations encrypted_message.txt output.txt.

Exact error message:

Multidivide #1: 0 (expected 5).

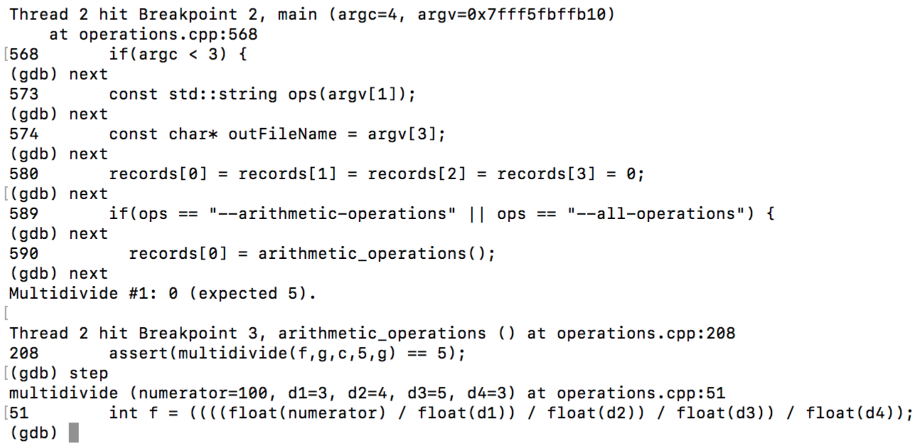
Assertion failed: (multidivide(f,g,c,5,g) == 5), function arithmetic_operations, file operations.cpp, line 208.   Abort trap: 6

Debugging process:

This error produced when I compiled the program with arithmetic-operations. Firstly, I read the compile error information: Assertion failed: (multidivide(f,g,c,5,g) == 5), which means multidivide(f,g,c,5,g) is not 5, and the position is on line 208 in operations.cpp. Then I got two assumption: The multidivide function is bugged or the arguments are mismatched. Therefore I used debugger: gdb to see what the exactly problem is(showing the steps in photos):

I have set the breakpoint at function: multidivide by gdb: b multidivide, then run the program by

r --arithmetic-operations encrypted_message.txt output.txt.



```
Thread 2 hit Breakpoint 2, main (argc=4, argv=0x7fff5fbffb10)
    at operations.cpp:568
568       if(argc < 3) {
(gdb) next
573       const std::string ops(argv[1]);
(gdb) next
574       const char* outFileName = argv[3];
(gdb) next
580       records[0] = records[1] = records[2] = records[3] = 0;
(gdb) next
589       if(ops == "--arithmetic-operations" || ops == "--all-operations") {
(gdb) next
590         records[0] = arithmetic_operations();
(gdb) next
Multidivide #1: 0 (expected 5).

Thread 2 hit Breakpoint 3, arithmetic_operations () at operations.cpp:208
208       assert(multidivide(f,g,c,5,g) == 5);
(gdb) step
multidivide (numerator=100, d1=3, d2=4, d3=5, d4=3) at operations.cpp:51
51        int f = ((((float(numerator) / float(d1)) / float(d2)) / float(d3)) / float(d4));
(gdb) 
```

Then I found a wired thing in the debugger: d1 = g = 3, but in line 195: int g = e - (b/c) + d + 20 = -1.

Thus I can draw the conclusion: some values of variables are wrong.

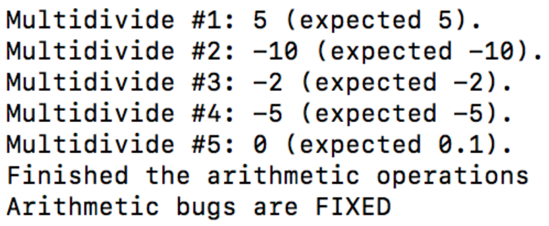
Bug fix:

Original code: int e = b - 3*a + 5*c;

New replacement code: int e = b - 3*a + 5*c -4 ;

The reason it can fix the bug: the change of e will correct g, then make the multidivide(f,g,c,5,g) == 5.

By the way, I checked other variables in the function, and corrected them(h,p,r), then I got the result:



```
Multidivide #1: 5 (expected 5).
Multidivide #2: -10 (expected -10).
Multidivide #3: -2 (expected -2).
Multidivide #4: -5 (expected -5).
Multidivide #5: 0 (expected 0.1).
Finished the arithmetic operations
Arithmetic bugs are FIXED
```

2. Wrong variable types in while loop from vector_operations.

Overall development environment:

MacOS Sierra 10.12.3, G++ version: 4.2.1.

Command that produce the problem:

./a.out --all-operations encrypted_message.txt output.txt.

Exact error message:

operations.cpp:532:27: warning: comparison of unsigned expression >= 0 is

always true [-Wtautological-compare]

for(uint i=counter-1; i >= 0; --i)

Debugging process:

Since this bug is interesting and tricky, so I picked it to my bug report. When I first read this error message, I was really confused, since how can a loop argument be always true? Then I remembered the Professor has told us on the lecture that the unsigned int means no sign, and can not be negative. **Then the loop will never stop. So I should change the unsigned int to int.**

Bug fix:

Original code: for(uint i=counter-1; i >= 0; --i)

Replacement code: for(int i=counter-1; i >= 0; —i)

The reason it can fix the code: Since normal int can be negative, the the for loop will stop when i < 0.

Result: The warning message disappeared.

3. Lack of logical implement for pythagoras.

Overall development environment:

MacOS Sierra 10.12.3, G++ version: 4.2.1.

Command that produce the problem:

./a.out --array-operations encrypted_message.txt output.txt.

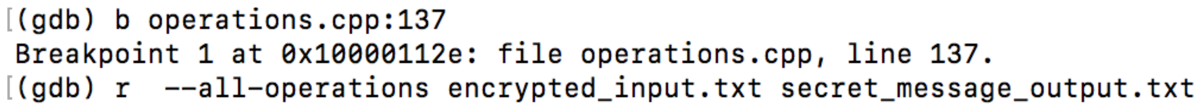
Exact error message:

Assertion failed: (array[1][2] == -1), function array_operations, file operations.cpp, line 137.

Abort trap: 6

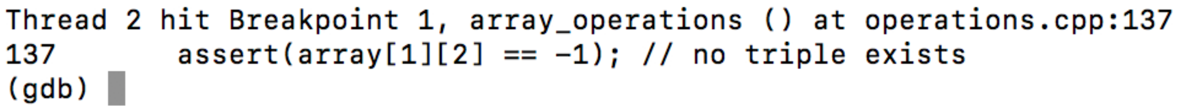
Debugging process:

When I took first glance on the error, I supposed the array must arranged wrongly. Then I used debugger: gdb to discover the values in the array. Following are photos to show the steps on debugger:

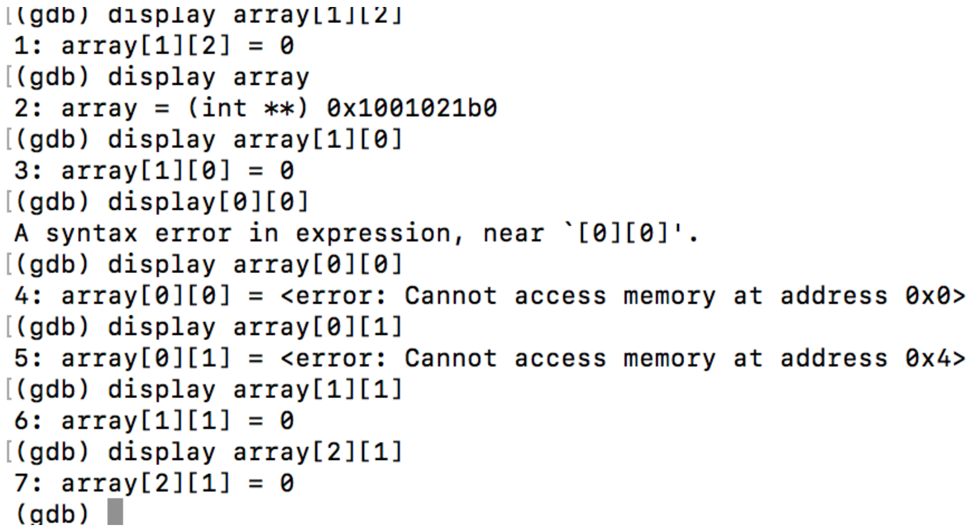


```
[(gdb) b operations.cpp:137
 Breakpoint 1 at 0x10000112e: file operations.cpp, line 137.
[(gdb) r  --all-operations encrypted_input.txt secret_message_output.txt
```

I have omitted the part of outputs.



```
Thread 2 hit Breakpoint 1, array_operations () at operations.cpp:137
137         assert(array[1][2] == -1); // no triple exists
(gdb) █
```



```
[(gdb) display array[1][2]
 1: array[1][2] = 0
[(gdb) display array
 2: array = (int **) 0x1001021b0
[(gdb) display array[1][0]
 3: array[1][0] = 0
[(gdb) display[0][0]
 A syntax error in expression, near `[0][0]'.
[(gdb) display array[0][0]
 4: array[0][0] = <error: Cannot access memory at address 0x0>
[(gdb) display array[0][1]
 5: array[0][1] = <error: Cannot access memory at address 0x4>
[(gdb) display array[1][1]
 6: array[1][1] = 0
[(gdb) display array[2][1]
 7: array[2][1] = 0
 (gdb) █
```

In many cases that have no triple exists stored 0 instead of -1 , so the problem is must in the function Pythagoras. I reviewed the function Pythagoras, and found that the function only considered the situation that the triple exists. **Therefore there is a serious logical error that the situation of no triples is ignored thus array[1][2] != -1.**

Bug fix:

Original code: no implementation for the situation of no triples.

Replacement code: Added the return -1 in the end of the function.

The reason it can fix the bug: the situation was implemented.

Result: the assertion failed disappeared.

4. Memory not assigned in pythagoras.

Overall development environment:

MacOS Sierra 10.12.3, G++ version: 4.2.1.

Command that produce the problem:

./a.out --array-operations encrypted_message.txt output.txt.

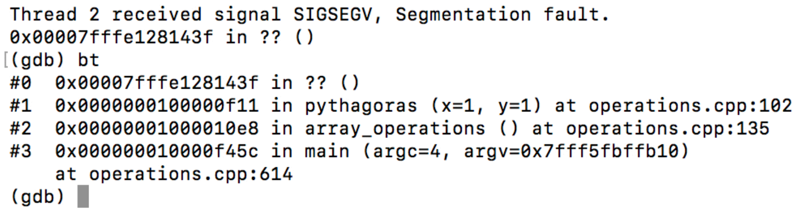
Exact error message:

Segmentation fault.

Debugging process:

After I have corrected the bug #3, the assertion failed was gone, but another error: segmentation fault occurred, which surprised me a lot. For the segmentation fault, I have no idea from the compilation message, since there was no hint for the position of the the error in code. So I found the error position by gdb, following the photo contained debugging steps:

After I run the code in gdb by r --array-operations encrypted_message.txt output.txt.



```
Thread 2 received signal SIGSEGV, Segmentation fault.
0x00007fffe128143f in ?? ()
[(gdb) bt
#0  0x00007fffe128143f in ?? ()
#1  0x0000000100000f11 in pythagoras (x=1, y=1) at operations.cpp:102
#2  0x00000001000010e8 in array_operations () at operations.cpp:135
#3  0x000000010000f45c in main (argc=4, argv=0x7fff5fbffb10)
    at operations.cpp:614
(gdb) 
```

Then I got the real position of the error: still in the Pythagoras function. I intended to use drmemory checking the memory bug, but luckily I found the bug by my eyes! Because we didn't allocate a true memory for placeholder, then the function modf() has no idea where to put the value. I have two ways to figure it out: changing the placeholder to int and passing the value by reference to modf() or assigned a new double on heap for the variable. **Since if we need to return the placeholder to prevent producing memory leak or changing too much on the original code, I choose to pass the value by reference for placeholder.**

Bug fix:

Original code: double* placeholder; modf(sqrt(sumsquares), placeholder);

Replacement code: double placeholder; modf(sqrt(sumsquares), &placeholder);

The reason it can fix the bug: pass by reference will not point to the garbage memory.

Result: the segmentation fault disappeared.

5. Logical error in for-loop for list-operations.

Overall development environment:

MacOS Sierra 10.12.3, G++ version: 4.2.1.

Command that produce the problem:

./a.out --list-operations encrypted_message.txt v.txt

Exact error message:

elderberry quart nectarine orange zwetschge pomegranate durian grape yellow squash fig iodine strawberry tangerine jujube lemon mango cherry uglyfruit apple watermelon kiwi

-1268210875 letters did not ever appear in the fruit names.

Assertion failed: (*l1.begin() == 'A'), function list_operations, file operations.cpp, line 404.

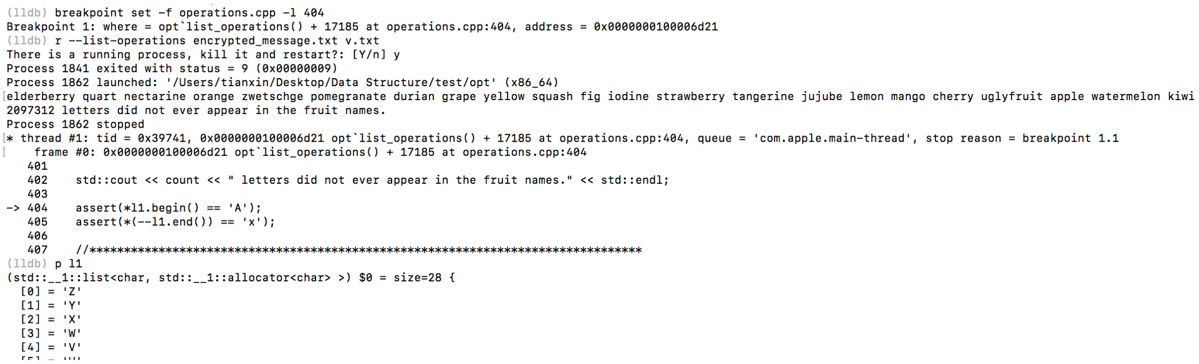
Abort trap: 6

Debugging process:

Obviously, the list l1 must arranged in wrong way, then the first element of the list is not A.

For this bug, I used lldb instead of gdb because lldb is better in displaying list or vector.

Following is the photo for debugging steps:



```
(lldb) breakpoint set -f operations.cpp -l 404
Breakpoint 1: where = opt`list_operations() + 17185 at operations.cpp:404, address = 0x0000000100006d21
(lldb) r --list-operations encrypted_message.txt v.txt
There is a running process, kill it and restart?: [Y/n] y
Process 1841 exited with status = 9 (0x00000009)
Process 1862 launched: '/Users/tianxin/Desktop/Data Structure/test/opt' (x86_64)
[elderberry quart nectarine orange zwetschge pomegranate durian grape yellow squash fig iodine strawberry tangerine jujube lemon mango cherry uglyfruit apple watermelon kiwi
2097312 letters did not ever appear in the fruit names.
Process 1862 stopped
[* thread #1: tid = 0x39741, 0x0000000100006d21 opt`list_operations() + 17185 at operations.cpp:404, queue = 'com.apple.main-thread', stop reason = breakpoint 1.1
[    frame #0: 0x0000000100006d21 opt`list_operations() + 17185 at operations.cpp:404
   401
   402     std::cout << count << " letters did not ever appear in the fruit names." << std::endl;
   403
-> 404     assert(*l1.begin() == 'A');
   405     assert(*(--l1.end()) == 'x');
   406
   407     //*******************************************************************************
(lldb) p l1
(std::__1::list<char, std::__1::allocator<char> >) $0 = size=28 {
  [0] = 'Z'
  [1] = 'Y'
  [2] = 'X'
  [3] = 'W'
  [4] = 'V'
  [5] = 'U'
```

Through the result from lldb, we can easily figure out that the order of uppercase words is wrong. Then i reviewed the for loop to create the list l1, and found an **apparent error in the loop: the order for the uppercase is opposite since the push_front is difference from push_back.**

Bug fix:

Original code: for(char c = 'A'; c <= 'Z'; c++) {
l1.push_front(c);}

Replacement code: for(char c = 'Z'; c >= 'A'; c--) {
l1.push_front(c);}

The reason it can fix the bug: I have corrected the for loop order then we can get a correct sequence in list.

Result: the assertion fail gone.

6. Wrong argument type for vector_sum.

Overall development environment:

MacOS Sierra 10.12.3, G++ version: 4.2.1.

Command that produce the problem:

./a.out --vector-operations encrypted_message.txt v.txt

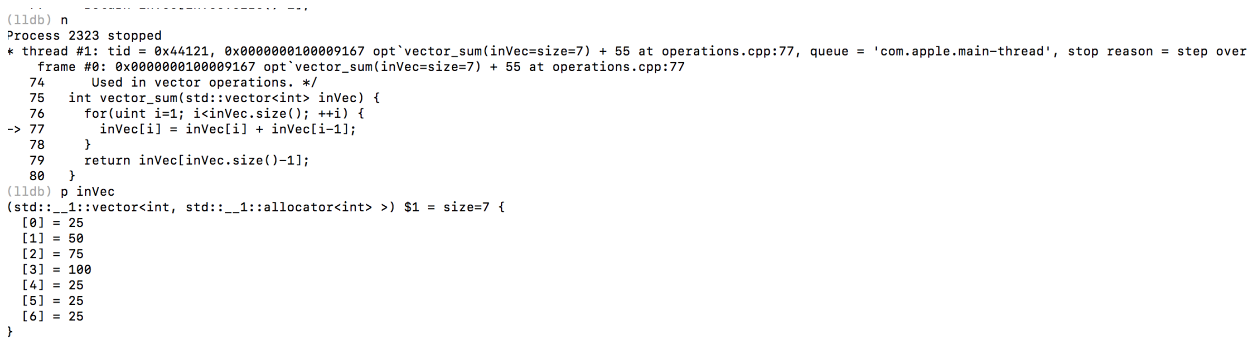
Exact error message:

Assertion failed: (v1[2] == 75), function vector_operations, file operations.cpp,   line 458.

Abort trap: 6

Debugging process:

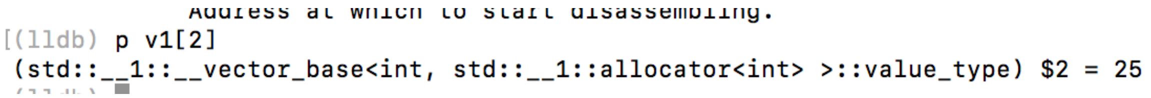
When I fist saw this error, I assumed the element in original v1 vector or the function vector_sum is wrong. Then I used lldb to printed out original v1 and the private variable invec in vector_sum. The original v1 is totally right from its simple initialize process. So I set a breakpoint on vector_sum and stepped in to the function:



```
(lldb) n
Process 2323 stopped
* thread #1: tid = 0x44121, 0x0000000100009167 opt`vector_sum(inVec=size=7) + 55 at operations.cpp:77, queue = 'com.apple.main-thread', stop reason = step over
    frame #0: 0x0000000100009167 opt`vector_sum(inVec=size=7) + 55 at operations.cpp:77
   74        Used in vector operations. */
   75      int vector_sum(std::vector<int> inVec) {
   76        for(uint i=1; i<inVec.size(); ++i) {
-> 77          inVec[i] = inVec[i] + inVec[i-1];
   78        }
   79        return inVec[inVec.size()-1];
   80      }
(lldb) p inVec
(std::__1::vector<int, std::__1::allocator<int> >) $1 = size=7 {
  [0] = 25
  [1] = 50
  [2] = 75
  [3] = 100
  [4] = 25
  [5] = 25
  [6] = 25
}
```

The invec[2] == 75, the function looks right in the implementation. Therefore, I have checked the v1[2] to find the truth behind the bug.

I set a breakpoint just before the assertion, and displayed the v1[2]:



```
                  Address at which to start disassembling.
[(lldb) p v1[2]
(std::__1::__vector_base<int, std::__1::allocator<int> >::value_type) $2 = 25
```

Spuriously, the v1[2] != invc[2], so the function didn't change the value in v1[2]. **I checked the argument type for sum_vector and the argument was not pass-by-reference.**

Bug fix:

Original code: int vector_sum(std::vector<int> inVec)

Replacement code: int vector_sum(std::vector<int> &inVec)

The reason it can fix the bug: When the argument is pass-by-reference, the function can truly change the value in the vector v1, then the bug fixed..

Result: the assertion fail gone.