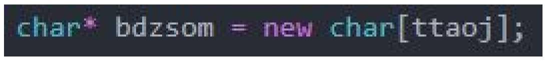
I used gdb to debug and doctor memory to find memory leak in the program on a laptop with windows 10 system on it.

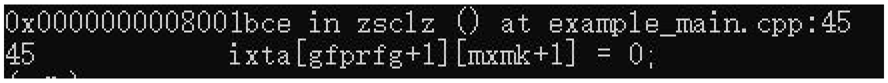
File:



```
char* bdzsom = new char[ttaoj];
```

There is an interesting "bug" or memory leak that needs to be fixed in the file part. As it shows, we have created a char array on the heap to store all information from the input text file. If we compile it, everything works and program can decrypt successfully. However, when I runned drmemory, it showed that this variable has caused memory leak. Since this variable is stored on the heap, it won't destruct itself when it goes out of scope, which means we have to delete it somewhere else in the program. I initially tried to delete this variable right after we finished reading through input file. I compiled and runned it, this time it showed that decryption failed. At this point, I realized that I cannot delete this variable inside the bool function since we will use it later, if we delete it now, we will cause the program to run incorrectly since the variable will now hold an incorrect value. I have found two other places where we have used this variable: later used in the part we are not suppose to edit and in the main function. Looking back at file function, the only place we can correctly delete this pointer is after it was used again (used in the part we are not supposed to edit anything, after that is the end of file function). Basically at this point, I have to find places outside file function to delete this pointer. However, the only other place I can find this variable is in main function. I was initially scared when I saw the comments "PROBABLY contain NO bugs". So I went back to main function, tried to delete this memory after we called bool function. When delete statement was added, all operations works as expected and no more struggles for me. This "bug" is especially tricky because most of time I have spent on fixing this bug was used to decide whether to delete it in main function or not.

Array:

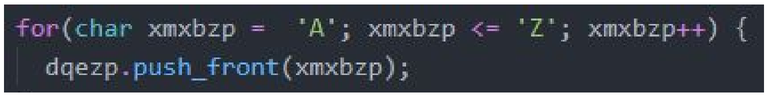


```
0x0000000008001bce in zsclz () at example_main.cpp:45
45              ixta[gfprfg+1][mxmk+1] = 0;
```

Before any edits had happened in array part, console showed segmental fault as its only error message if I tried to compile at this time. Entering into gdb and using backtrace call, it showed that function crashed at line 45 when program was trying to add int 0 into next row and next column. It looks completely legit at first glance, all it does is giving initial values to the vector. After I took a look at the entire for loop again, I found out that we are creating rows one by one, which means when we are adding elements into current row, that will be the last row in the vector, next row has not been created since the for loop has not reached there yet. Since we will get to every single rows and columns anyways and giving all spots integer 0 (argument of for loop will change as next paragraph describes), there is not point to add another line to change vector[current_row+1] [current_column+1], so I delete this line, and that totally fix the problem.

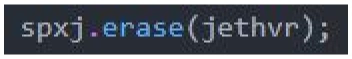
There were other bugs I have found in the for loop. First of all, there was a second, empty 2D array which has never been used elsewhere in the function. Secondly, since we need to cover every spots in the array and filled it with integer 0, the first argument in the for loop condition should start from int 0 and second argument should end at 24. If we use int 1 as the starting spot, the first position of each row will be filled in with random values which is not what we want to do. I have also found that in array part, the programmer made same mistakes at other places when he is trying to access the first and last position of the array (or four corners of a 2D array), they could be properly accessed by using subscript [0] and [array.size() - 1].

List:



```
for(char xmxbzp =  'A'; xmxbzp <= 'Z'; xmxbzp++) {
  dqezp.push_front(xmxbzp);
```

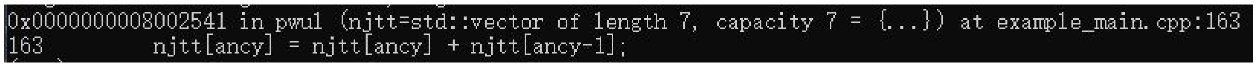
First running list part, console showed core dumped, by using gdb, I have found that char list did not pass assert test. I assumed that the program didn't add characters into this list as what we expect it to do. Using print inside gdb, I saw every elements now in the list, I found the uppercase letters are not in the correct order. Requirement of this for loop is that 'A' comes before 'Z' in the vector. If I am using push_front function, I would have to add 'Z' first, 'A' last, so basically I switch starting point and the ending point of this for loop, and then change increment to decrement.



```
spxj.erase(jethvr);
```

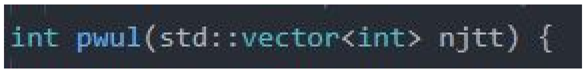
Moreover, the programmer didn't use iterator correctly after the call of erase function in std::list. For example, as the image shows above, programmer is trying to use erase function to delete an element from a std::list. Different than vector, we are using iterators to trace every elements in a list, in this case, it won't work if we just delete an element and make no changes to current iterators. In the given file, such mistake causes the program to crash. By using gdb and printing out elements in the list after the erase function, console shows list ends at wherever we call erase function and spots after that are all invalid. In other words, after we delete an element, previous iterator is not pointing at the new position to connect next element. In order to fix it, we have to find the previous iterator, change it to point to one iterator after the element we want to delete, then call erase function. When using iterators to make changes on a list type, it is important to keep all pointers in the list up-to-date, pointing at the right position.

Vector:



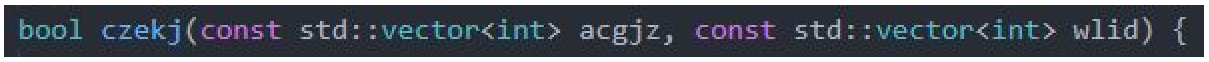
```
0x0000000008002541 in pwul (njtt=std::vector of length 7, capacity 7 = {...}) at example_main.cpp:163
163         njtt[ancy] = njtt[ancy] + njtt[ancy-1];
```

When I runned the program, it didn't pass assert tests. In gdb, the error message shows as above. We are adding each elements in the vector with previous value. I was initially confused on how this line went wrong when I tried to run it. But when I tried to print out the value of the vector, the program immediately crashed, didn't even this algorithm at all. From that I realized that the function crashed at the very beginning. So I checked the first position of the vector, then I realized that function is trying to do vector[current-1] at position 0. However, there is no previous value at first position, so I changed the starting position of for loop from 0 to 1 to avoid first position in the vector.



```
int pwul(std::vector<int> njtt) {
```

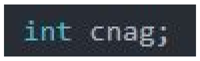
After this bug has been fixed, it still didn't pass assert tests. At this time, the only thing could go wrong is the function that add each elements in a vector with its previous values. I used gdb to set up two breakpoints, one before we first call this function, one after this function has runned once. Then I print the value of vector at those two breakpoints. Surprisingly, two vectors have completely same values, which means this function never actually change anything inside original vector. Looking at this function again, I found that we are passing in vectors by values. This doesn't change anything inside the original vector after the function is done. When we pass something into a function by values, the function would create a copy of it and make changes on the copy one, but no changes will be made on original vector. In order to fix it, I change the parameter into passing by reference when we declare the function prototype and implementing this function.



```
bool czekj(const std::vector<int> acgjz, const std::vector<int> wlid) {
```

Moreover, the function above compares the size of two vectors, return true if every elements in the first vector is bigger than the second, return false otherwise. This function is implemented correctly, and program successfully run with these codes. However, after I used doctor memory to test memory leak, I found this function contains memory leak (2 unaddressable accesses). Since this is my first time encounter unaddressable accesses, I was initially really confused why this function would cause such problem since all lines inside function contain no bugs. Looking back to main vector function, I found out that when we are calling this bool function, sometimes we are comparing two vectors with different size. Logically, the function should directly return false as what comments require me to do, but the bool function didn't even cover this special case. If we want to compare two vectors of different size, program will still use for loop to go through and compare each element of two vectors. It will be wrong when either vector reaches its last spot but the other vector still have couple elements left. In order to handle this case, I have added an if statement to deal with two vectors with different size.

Overall:

There are lots of minor bugs in this program. In a function, there is a variable being used for two different calculations. I was first really confused when my output is slightly off but all codes look good. Later on, using print in gdb, I realize this variable was first used to hold numbers of values in a vector that is a multiple of 10, then used again to hold numbers of values that is divisible by 3. These two numbers should be stored in two different variables so that it could be distinguished and number won't be wrong. Moreover, there is a problem related to declared variable without assigning any values. 

```
int cnag;
```

From the image, we see that this integer is declared, and it was later used in for loop increment. We want to avoid such case because if we declare a variable without giving any values, it might be assigned with value 0 as we want, but it might be other random values, we never know. In order for it to perform as we want to, we should assign an initial value, in this cause, this variable should be initially assigned to 0.

After this homework, I have learned how to fix memory leak under any conditions, especially when it has unaddressable access. I have also practiced how to utilize pointers and iterators inside lists since they are both new, and more practice with arrays and vectors which I am already used to. I found this homework helpful because it forces me to use to gdb to debug many times when the only message I see in the console is segmental fault, but now after this homework I completely understand and learn how to use gdb to debug.