Chemeketa CS Dev Environment
Building and Testing Code

Contents

Requirements

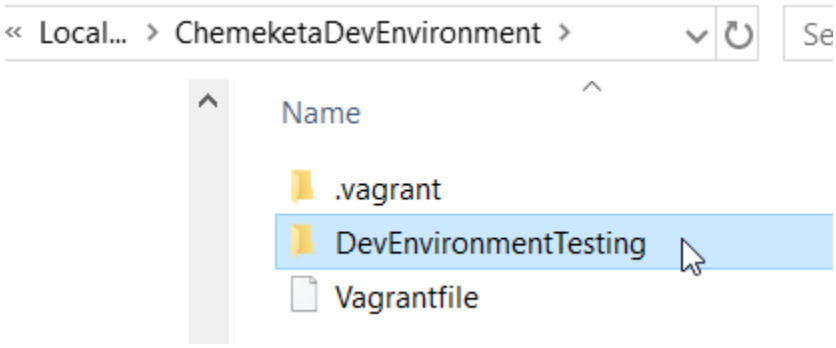
It is assumed that you have the **ChemeketaDevEnvironment** virtual machine set up and running. Unless stated otherwise, all commands will be happening in the guest (linux) operating system.

It makes use of the project **DevEnvironmentTesting** from the CS162 code repository:
https://github.com/ascholerChemeketa/cs162Code

Building And Running Code

To run code in the guest OS (linux) you must build a new version of the program. You cannot run a windows or mac program in the Linux OS.

1. On your host OS, make sure your project folder is in the ChemeketaDevEnvironment folder that is shared with the virtual machine. Here I am ready to work on **CourseProject**:



   You may have the project open in QTCreator on your host operating system to edit files while you build and run in the guest operating system. You can even build and run a version of your program in the host OS so you can debug it while you are also building and running the program in Linux.

2. In the guest OS, navigate to that folder and list the files (something like "**cd /vagrant**", "**cd DevEnvironmentTesting**" and then "**ls -la**"). Make sure you see the cpp files you wish to compile.

   Then make a folder to hold the Linux version of the program:

mkdir linux
You should see a folder called "linux" appear inside your project folder.

Then do:
g++ -std=c++11 -g -o linux/MyProgram.exe *.cpp
Compile (g++) using C++11 (-std=c++11), with debugging (-g), call the program MyProgram.exe and place it in the directory "linux"(-o linux/MyProgram.exe), using all the .cpp files in this directory (.cpp)*

This command will build all of your .cpp files into the program MyProgram.exe and place it in the linux folder. You will only be able to run it from the guest OS. If there are compile errors, stop and fix them (you can edit files in your host OS).

If you do not want to compile all the .cpp files, list them out instead of using *.cpp:
g++ -std=c++11 -g -o linux/MyProgram.exe main.cpp Location.cpp
Same as before, but compile the files main.cpp and Location.cpp

3. To run your program, type:
   linux/MyProgram.exe
   "Run MyProgram.exe that is in the linux folder"

   This will run the program using the current directory (the project folder, not the linux folder) as the working directory. Any files that are read from by your program should be in that folder.

   Your program should run and show you the output.
4. Hint: up/down arrows cycle through recent commands - you can use that to rebuild the code after changing it instead of retyping the command.

Using DrMemory

1. First build your program as described above. Then ask DrMemory to execute your code:
   drmemory -- linux/MyProgram.exe

2. Drmemory will spit out a report like this after your program is done. Do not worry about any ERRORS IGNORED, focus on any that appear in the top summary under ERRORS FOUND. Here there is 1 real error:

```
vagrant@ubuntu-16:/vagrant/DevEnvironmentTesting$ drmemory -- linux/MyProgram.exe
~~Dr.M~~ Dr. Memory version 2.0.0
~~Dr.M~~
~~Dr.M~~ Error #1: LEAK 40 direct bytes 0x000000000061b000-0x000000000061b028 + 0 indirect bytes
~~Dr.M~~ # 0 replace_operator_new_array              [/drmemory_package/common/alloc_replace.c:2929]
~~Dr.M~~ # 1 makeLocationList                       [/vagrant/DevEnvironmentTesting/main.cpp:9]
~~Dr.M~~ # 2 main                                   [/vagrant/DevEnvironmentTesting/main.cpp:16]
~~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,     40 byte(s) of leak(s)
~~Dr.M~~       0 unique,     0 total,      0 byte(s) of possible leak(s)
~~Dr.M~~ ERRORS IGNORED:
~~Dr.M~~      15 unique,    33 total,  89032 byte(s) of still-reachable allocation(s)
~~Dr.M~~          (re-run with "-show_reachable" for details)
~~Dr.M~~ Details: /tmp/Dr. Memory/DrMemory-MyProgram.exe.19091.000/results.txt
vagrant@ubuntu-16:/vagrant/DevEnvironmentTesting$
```

3. If you want to capture all the DrMemory output to a file, you can do so with:
   drmemory -- linux/MyProgram.exe 2> errors.txt
   That says "redirect error output (stream 2) to the file errors.txt". You can then open that file and scroll around to read it. To redirect both your program and drmemory's output to the file do:
   drmemory -- linux/MyProgram.exe &> errors.txt

Continues…

Leak Example

The code provided in the sample has a leak:

```
vagrant@ubuntu-16:/vagrant/DevEnvironmentTesting$ drmemory -- linux/MyProgram.exe
~~Dr.M~~ Dr. Memory version 2.0.0
~~Dr.M~~
~~Dr.M~~ Error #1: LEAK 40 direct bytes 0x000000000061b000-0x000000000061b028 + 0 indirect bytes
~~Dr.M~~ # 0 replace_operator_new_array              [/drmemory_package/common/alloc_replace.c:2929]
~~Dr.M~~ # 1 makeLocationList                        [/vagrant/DevEnvironmentTesting/main.cpp:9]
~~Dr.M~~ # 2 main                                    [/vagrant/DevEnvironmentTesting/main.cpp:16]
~~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,     40 byte(s) of leak(s)
~~Dr.M~~       0 unique,     0 total,      0 byte(s) of possible leak(s)
~~Dr.M~~ ERRORS IGNORED:
~~Dr.M~~      15 unique,    33 total,  89032 byte(s) of still-reachable allocation(s)
~~Dr.M~~         (re-run with "-show_reachable" for details)
~~Dr.M~~ Details: /tmp/Dr. Memory/DrMemory-MyProgram.exe.19091.000/results.txt
vagrant@ubuntu-16:/vagrant/DevEnvironmentTesting$
```

At the bottom we are told there is 1 leak that make up 40 bytes. Above that, we are told where the leaks actually are. Error #1 is a leak of 40 bytes. Below each error is the call stack at the time of the error.

- #0 tells us the memory was allocated by the new operator (#0)… we don't care about that - we are looking for an error in our code.
- #1 is the function that called new. That was the function makeLocationList from our main.cpp, line 9. The :9 after main.cpp indicates line number 9.
- #2 says that makeLocationList was called from the main function line 16.

Note: The DrMemory showed me where the memory that was leaked was created. It does not tell us where it was leaked. In this case, t can't know where we intended to free that memory - figuring that out is our job.

DrMemory tells you there is an error and gives you the symptoms, it does not tell you what the cure is.

1. Delete the array at the end of main to get rid of the leak:

```
13  int main()
14  {
15      //Get an array of 5 locations
16      Location* locationList = makeLocationList(5);
17      delete [] locationList;
18  }
```

2. Make sure to save the files in QTCreator, then rebuild the program in Linux and rerun DrMemory. You should get a clean bill of health:

```
~~Dr.M~~ Dr. Memory version 2.0.0
~~Dr.M~~
~~Dr.M~~ NO 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~~       0 unique,     0 total,      0 byte(s) of leak(s)
~~Dr.M~~       0 unique,     0 total,      0 byte(s) of possible leak(s)
~~Dr.M~~ ERRORS IGNORED:
~~Dr.M~~      15 unique,    33 total,  89032 byte(s) of still-reachable allocation(s)
~~Dr.M~~         (re-run with "-show_reachable" for details)
~~Dr.M~~ Details: /tmp/Dr. Memory/DrMemory-MyProgram.exe.34690.000/results.txt
vagrant@ubuntu-16:/vagrant/DevEnvironmentTesting$
```

Out of Bounds Example

1. This time, add a line of code to main that goes past the end of the array. Something like this that accesses index 5 in an array of size 5:

```
13  int main()
14  {
15      //Get an array of 5 locations
16      Location* locationList = makeLocationList(5);
17      cout << locationList[5].getX();
18      delete [] locationList;
19  }
```

2. Rebuild the application to include the changes, and then rerun in DrMemory:

```
~~Dr.M~~ Dr. Memory version 2.0.0
~~Dr.M~~
~~Dr.M~~ Error #1: UNADDRESSABLE ACCESS beyond heap bounds: reading 0x000000000061c028-0x000000000061c02c 4 byte(s)
~~Dr.M~~ # 0 Location::getX              [/vagrant/DevEnvironmentTesting/Location.cpp:17]
~~Dr.M~~ # 1 main                        [/vagrant/DevEnvironmentTesting/main.cpp:17]
~~Dr.M~~ Note: @0:00:00.885 in thread 34706
~~Dr.M~~ Note: refers to 0 byte(s) beyond last valid byte in prior malloc
~~Dr.M~~ Note: prev lower malloc:  0x000000000061c000-0x000000000061c028
~~Dr.M~~ Note: instruction: mov    (%rax) -> %eax
0~~Dr.M~~
~~Dr.M~~ ERRORS FOUND:
~~Dr.M~~       1 unique,     1 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~~       0 unique,     0 total,      0 byte(s) of leak(s)
~~Dr.M~~       0 unique,     0 total,      0 byte(s) of possible leak(s)
~~Dr.M~~ ERRORS IGNORED:
~~Dr.M~~      16 unique,    34 total,  90056 byte(s) of still-reachable allocation(s)
~~Dr.M~~          (re-run with "-show_reachable" for details)
~~Dr.M~~ Details: /tmp/Dr. Memory/DrMemory-MyProgram.exe.34706.000/results.txt
vagrant@ubuntu-16:/vagrant/DevEnvironmentTesting$
```

Error #1 is a an UNADDRESSABLE ACCESS - we used a memory address we should not have. The call stack says:

- #0 tells us that getX function used bad memory. But it was not it's fault
- #1 tells us getX was called from main line 17.

The heart of the problem here was the bad array index on line 17 of main. It is critical to examine the whole call stack to identify the root cause of the error.

DrMemory tells you there is an error and gives you the symptoms, it does not tell you what the cure is.

Use of Freed Memory Example

1. Modify makeLocationList to delete the memory before returning a pointer:

```
 8  Location* makeLocationList(int size) {
 9      Location* temp = new Location[size];
10      delete [] temp;
11      return temp;
12  }
```

   Make main look like:

```
16  int main()
17  {
18      //Get an array of 5 locations
19      Location* locationList = makeLocationList(5);
20      cout << locationList[2].getX();
21      delete [] locationList;
22  }
```

2. Build and test with DrMemory:

```
~~Dr.M~~
~~Dr.M~~ Error #1: UNADDRESSABLE ACCESS of freed memory: reading 0x000000000061c010-0x000000000061c014 4 byte(s)
~~Dr.M~~ # 0 Location::getX              [/vagrant/DevEnvironmentTesting/Location.cpp:17]
~~Dr.M~~ # 1 main                        [/vagrant/DevEnvironmentTesting/main.cpp:20]
~~Dr.M~~ Note: @0:00:00.880 in thread 34743
~~Dr.M~~ Note: 0x000000000061c010-0x000000000061c014 overlaps memory 0x000000000061c000-0x000000000061c028 that was
 freed here:
~~Dr.M~~ Note: # 0 replace_operator_delete_array              [/drmemory_package/common/alloc_replace.c:2999]
~~Dr.M~~ Note: # 1 makeLocationList                           [/vagrant/DevEnvironmentTesting/main.cpp:10]
~~Dr.M~~ Note: # 2 main                                       [/vagrant/DevEnvironmentTesting/main.cpp:19]
~~Dr.M~~ Note: instruction: mov    (%rax) -> %eax
~~Dr.M~~
~~Dr.M~~ Error #2: INVALID HEAP ARGUMENT to free 0x000000000061c000
~~Dr.M~~ # 0 replace_operator_delete_array             [/drmemory_package/common/alloc_replace.c:2999]
~~Dr.M~~ # 1 main                                      [/vagrant/DevEnvironmentTesting/main.cpp:21]
~~Dr.M~~ Note: @0:00:00.900 in thread 34743
~~Dr.M~~ Note: next higher malloc: 0x000000000061c050-0x000000000061c450
~~Dr.M~~ Note: memory was previously freed here:
~~Dr.M~~ Note: # 0 replace_operator_delete_array              [/drmemory_package/common/alloc_replace.c:2999]
~~Dr.M~~ Note: # 1 makeLocationList                           [/vagrant/DevEnvironmentTesting/main.cpp:10]
~~Dr.M~~ Note: # 2 main                                       [/vagrant/DevEnvironmentTesting/main.cpp:19]
0~~Dr.M~~
~~Dr.M~~ ERRORS FOUND:
~~Dr.M~~       1 unique,     1 total unaddressable access(es)
~~Dr.M~~       0 unique,     0 total uninitialized access(es)
~~Dr.M~~       1 unique,     1 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)
```

Error #1 is a an UNADDRESSABLE ACCESS - we used a memory address that was already freed. The call stack says:

- #0 tells us that getX function used bad memory. But it was not it's fault
- #1 tells us getX was called from main line 20. This isn't the problem area either.
- The Note under that tell us where the memory was freed:
  - Note #0 tells us in operator delete… that isn't our concern
  - Note #1 tells us delete was called from makeLocationList - that is where our problem is.

Error #2 tells us that main line 21 tries to free memory that was already freed. That is just another symptom of the line of code that deletes the array temp points to before it is returned in makeLocationList.

DrMemory tells you there is an error and gives you the symptoms, it does not tell you what the cure is.