CS 300 In Class Valgrind Lab

Download the Valgrind Workspace. Untar the workspace and open that workspace in Eclipse.

/home/CS300Public/2019/ValgrindWorkspace.tar.gz

The project **ValgrindLinkedListLab** lab contains two C files, mainArrays.c and mainList.c, a Makefile and a data file, numbers.txt. You must find and fix memory errors in mainArrays.c and mainList.c.

Use Valgrind to profile the code and see where some errors come from.

1) Run mainArrays from inside Eclipse. The first data mainArrays prints to the screen are the sizes of various primitive data types. This data is necessary for later parts of this lab.
Note: Very often if you have memory errors you will get no output on the console inside Eclipse.

2) Run main from the command line.

```
cd ValgrindWorkspace/ValgrindLinkedListLab
bin/mainArrays
```

3) Run mainArrays through valgrind at the command line

```
valgrind  -v --leak-check=yes --track-origins=yes --leak-check=full --show-
leak-kinds=all bin/mainArrays
```

4) Run mainArrays through valgrind in Eclipse
Right Click mainArrays | Profileing Tools | Profile with Valgrind

5) Fix errors! Iterate! Memory errors may mask other memory errors!

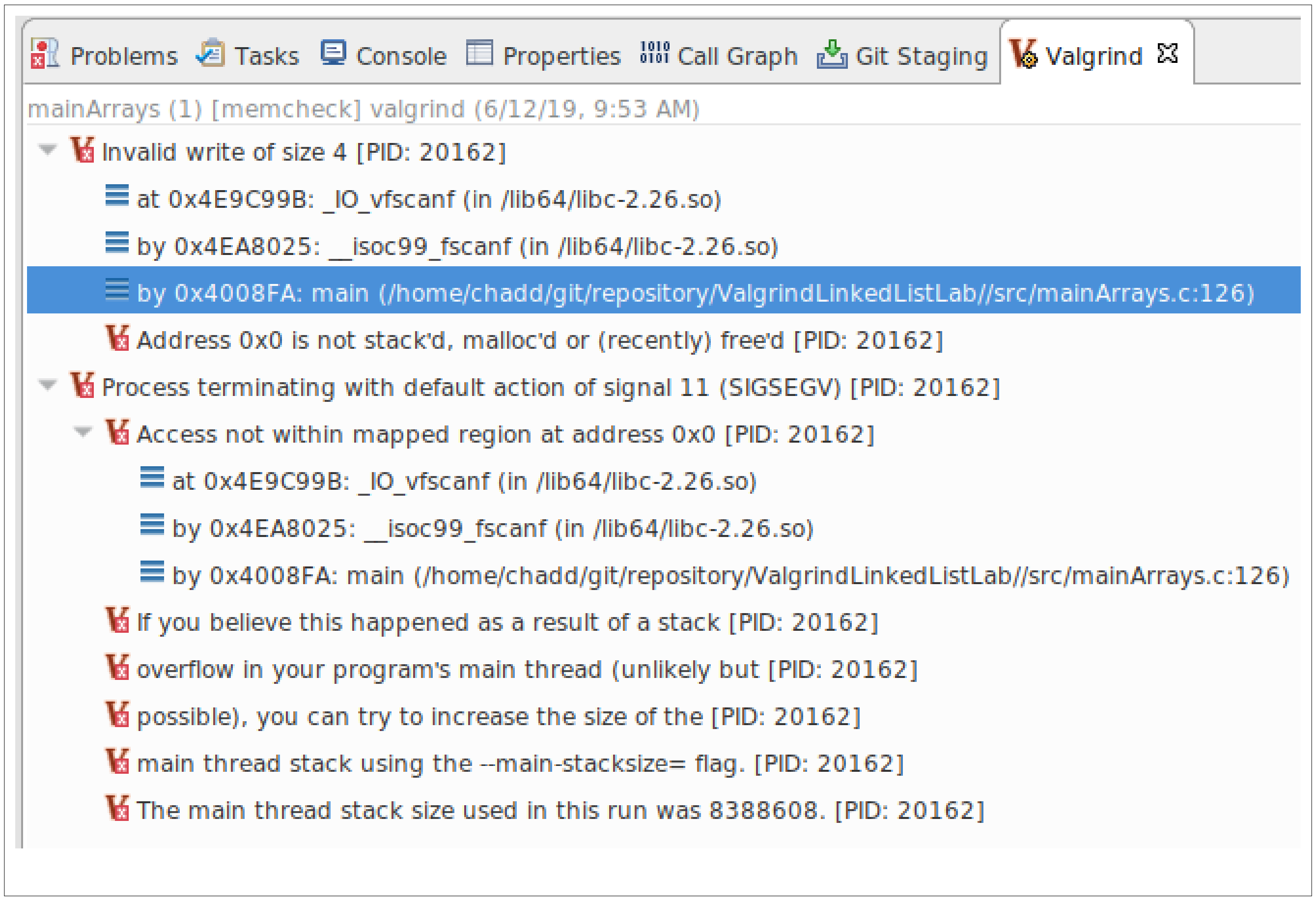
Use Valgrind to debug mainList as well. mainList may not crash, but does produce bad output!

Ask questions!

http://valgrind.org/docs/manual/mc-manual.html#mc-manual.errormsgs

NOTES

mainArrays The first errors you see should be:



```
mainArrays (1) [memcheck] valgrind (6/12/19, 9:53 AM)
  Invalid write of size 4 [PID: 20162]
      at 0x4E9C99B: _IO_vfscanf (in /lib64/libc-2.26.so)
      by 0x4EA8025: __isoc99_fscanf (in /lib64/libc-2.26.so)
      by 0x4008FA: main (/home/chadd/git/repository/ValgrindLinkedListLab//src/mainArrays.c:126)
      Address 0x0 is not stack'd, malloc'd or (recently) free'd [PID: 20162]
  Process terminating with default action of signal 11 (SIGSEGV) [PID: 20162]
      Access not within mapped region at address 0x0 [PID: 20162]
          at 0x4E9C99B: _IO_vfscanf (in /lib64/libc-2.26.so)
          by 0x4EA8025: __isoc99_fscanf (in /lib64/libc-2.26.so)
          by 0x4008FA: main (/home/chadd/git/repository/ValgrindLinkedListLab//src/mainArrays.c:126)
      If you believe this happened as a result of a stack [PID: 20162]
      overflow in your program's main thread (unlikely but [PID: 20162]
      possible), you can try to increase the size of the [PID: 20162]
      main thread stack using the --main-stacksize= flag. [PID: 20162]
      The main thread stack size used in this run was 8388608. [PID: 20162]
```

NOTE: As you edit the code, the line numbers you see in the error messages may vary!

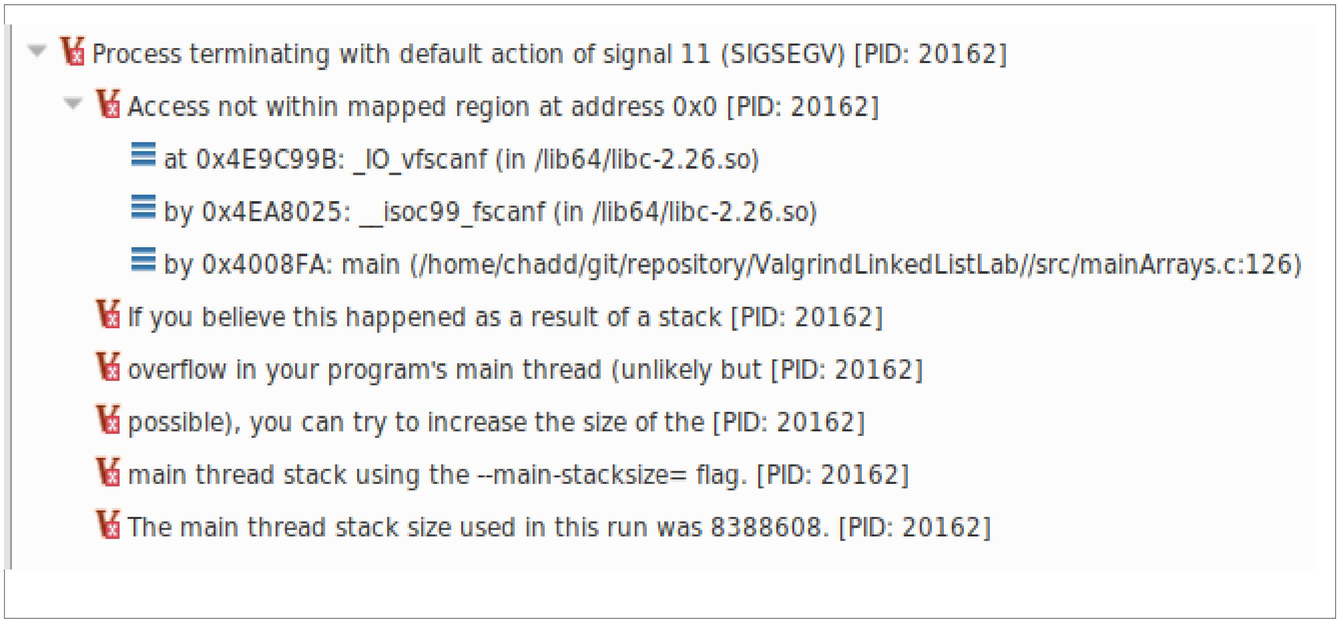
Invalid write of size 4: Four bytes were written into an invalid memory location. What data types do we know that are 4 bytes? An invalid memory location is an address in memory not allocated for writing to our program. The stack trace (blue lines icon) shows the set of functions on the runtime stack when the error occurred. Double clicking on the stack trace will take you to the particular line of code in Eclipse. The top function listed was called by the function below it, and so on. The bottom line in the stack trace (highlighted above) is where execution started in your code. *The highest up line in your code is likely where you want to start looking for your error.*

Can you determine which function is called on line 126 in dynamicArrays.c?

The last line in the error message (red V icon) describes the error. **Address 0x0 is not stack'd, malloc'd or (recently) free'd.** This means address 0 is not a stack address, allocated on the heap, or a heap address you recently free()ed.

So, your looking at line 126, looking for a datatype that is 4 bytes large that you are writing to, and you are looking for a pointer (or value being used as a pointer) that contains the value zero. You might want to set a breakpoint on this line, start the debugger, and then check the values of all the variables on that line.

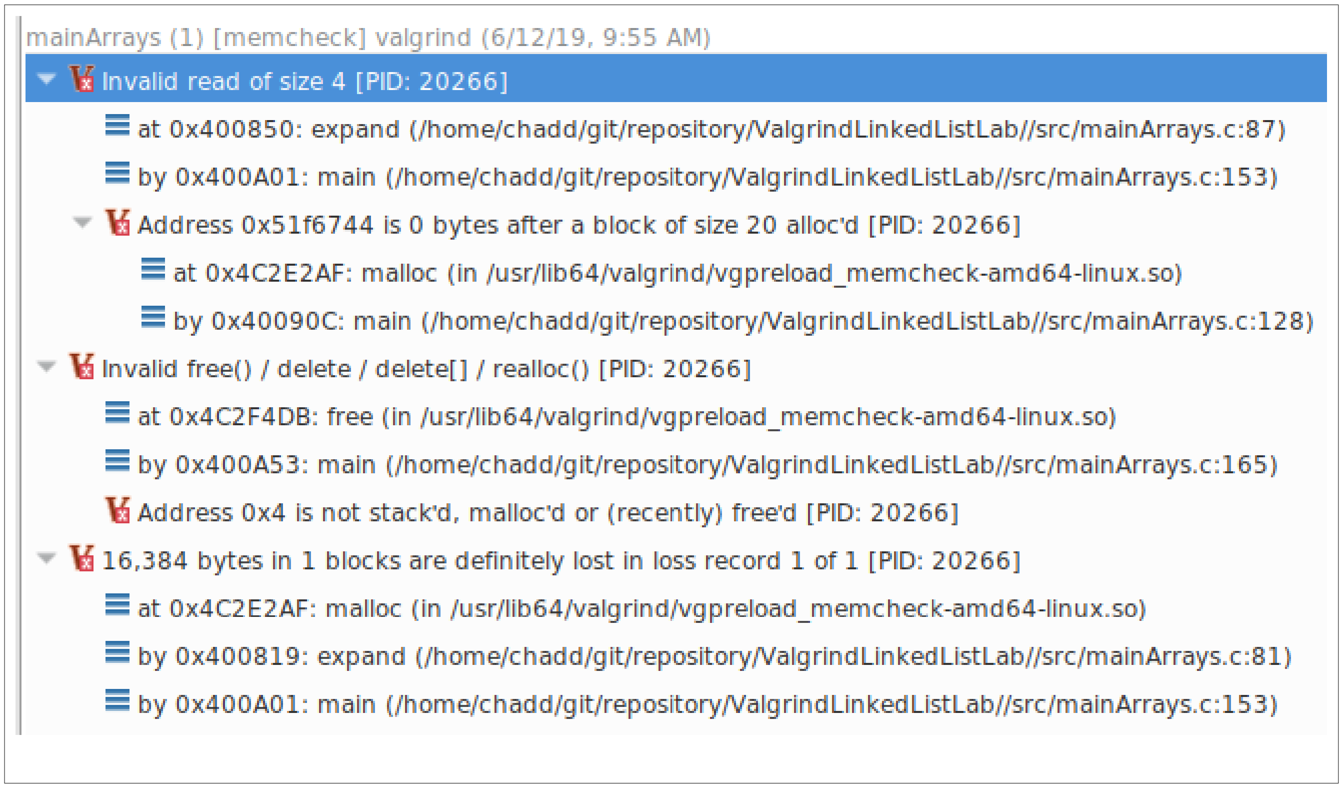
The second error, **Process terminating with default action of signal 11**, indicates your program crashed when accessing address 0.



```
Process terminating with default action of signal 11 (SIGSEGV) [PID: 20162]
    Access not within mapped region at address 0x0 [PID: 20162]
        at 0x4E9C99B: _IO_vfscanf (in /lib64/libc-2.26.so)
        by 0x4EA8025: __isoc99_fscanf (in /lib64/libc-2.26.so)
        by 0x4008FA: main (/home/chadd/git/repository/ValgrindLinkedListLab//src/mainArrays.c:126)
    If you believe this happened as a result of a stack [PID: 20162]
    overflow in your program's main thread (unlikely but [PID: 20162]
    possible), you can try to increase the size of the [PID: 20162]
    main thread stack using the --main-stacksize= flag. [PID: 20162]
    The main thread stack size used in this run was 8388608. [PID: 20162]
```

Drop down the arrow beside **Access not within mapped region at address 0x0** to confirm that this stems from line 126 in your code.

Once you fix that one error, three more should appear.



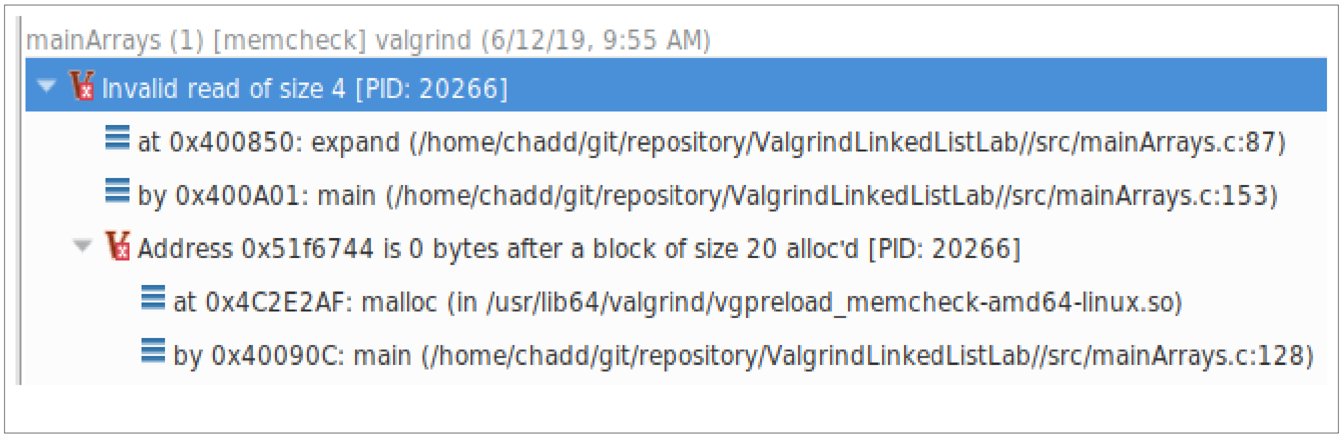
```
mainArrays (1) [memcheck] valgrind (6/12/19, 9:55 AM)
  Invalid read of size 4 [PID: 20266]
      at 0x400850: expand (/home/chadd/git/repository/ValgrindLinkedListLab//src/mainArrays.c:87)
      by 0x400A01: main (/home/chadd/git/repository/ValgrindLinkedListLab//src/mainArrays.c:153)
    Address 0x51f6744 is 0 bytes after a block of size 20 alloc'd [PID: 20266]
        at 0x4C2E2AF: malloc (in /usr/lib64/valgrind/vgpreload_memcheck-amd64-linux.so)
        by 0x40090C: main (/home/chadd/git/repository/ValgrindLinkedListLab//src/mainArrays.c:128)
  Invalid free() / delete / delete[] / realloc() [PID: 20266]
      at 0x4C2F4DB: free (in /usr/lib64/valgrind/vgpreload_memcheck-amd64-linux.so)
      by 0x400A53: main (/home/chadd/git/repository/ValgrindLinkedListLab//src/mainArrays.c:165)
      Address 0x4 is not stack'd, malloc'd or (recently) free'd [PID: 20266]
  16,384 bytes in 1 blocks are definitely lost in loss record 1 of 1 [PID: 20266]
      at 0x4C2E2AF: malloc (in /usr/lib64/valgrind/vgpreload_memcheck-amd64-linux.so)
      by 0x400819: expand (/home/chadd/git/repository/ValgrindLinkedListLab//src/mainArrays.c:81)
      by 0x400A01: main (/home/chadd/git/repository/ValgrindLinkedListLab//src/mainArrays.c:153)
```

Here you see 3 errors. We will look at these in turn. You will need to solve all three plus any other you find to complete this lab.

Invalid read of size 4

Invalid free() / delete / delete[] / realloc()

16,384 bytes in 1 blocks are definitely lost in loss record 1 of 1

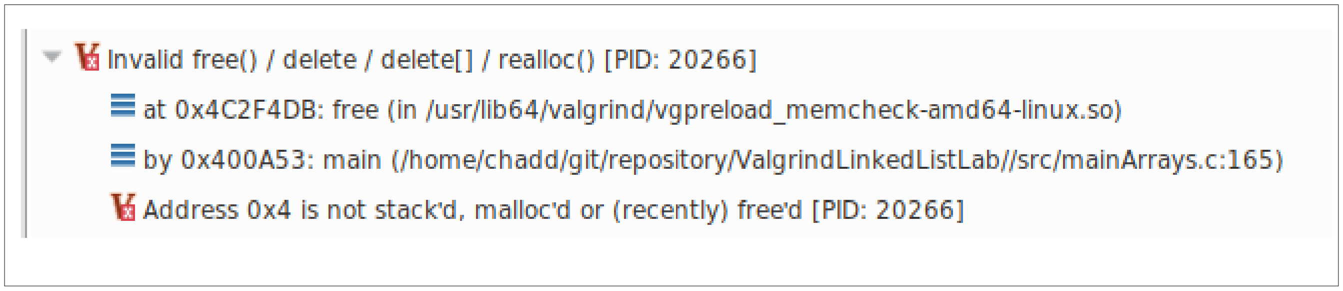


```
mainArrays (1) [memcheck] valgrind (6/12/19, 9:55 AM)
▼ Invalid read of size 4 [PID: 20266]
      at 0x400850: expand (/home/chadd/git/repository/ValgrindLinkedListLab//src/mainArrays.c:87)
      by 0x400A01: main (/home/chadd/git/repository/ValgrindLinkedListLab//src/mainArrays.c:153)
   ▼ Address 0x51f6744 is 0 bytes after a block of size 20 alloc'd [PID: 20266]
         at 0x4C2E2AF: malloc (in /usr/lib64/valgrind/vgpreload_memcheck-amd64-linux.so)
         by 0x40090C: main (/home/chadd/git/repository/ValgrindLinkedListLab//src/mainArrays.c:128)
```

Invalid read of size 4
An invalid read of size 4 means you read 4 bytes from an area of memory that you did not allocate. What data type might cause us to read 4 bytes? Typically this means a pointer has moved off of allocated memory or was never allocated properly. This error produces two stack traces. One where the invalid read happens and another describing where memory close to the read was allocated.

We can see the stack trace at the top, the function **expand** is called by **main.** Check out lines 87 and 153 to see what is on those lines. Again, the highest line in the stack trace that mentions your code is where to start looking for the error. In this case, line 87.

The error description is: **Address 0x51f744 is 0 bytes after a block of size 20 alloc'd**. This means you are reading 4 bytes that are adjacent to (0 bytes away from) a 20 byte block of memory you did allocate (note that 20/4 is 5. Be sure to identify where the 20 and 4 come from. Once you fix this error you should recognize where the 5 comes from). Check out line 128 (from the highlighted line above). What is happening on this source line? Why is **malloc** listed at the top of the second stack trace.

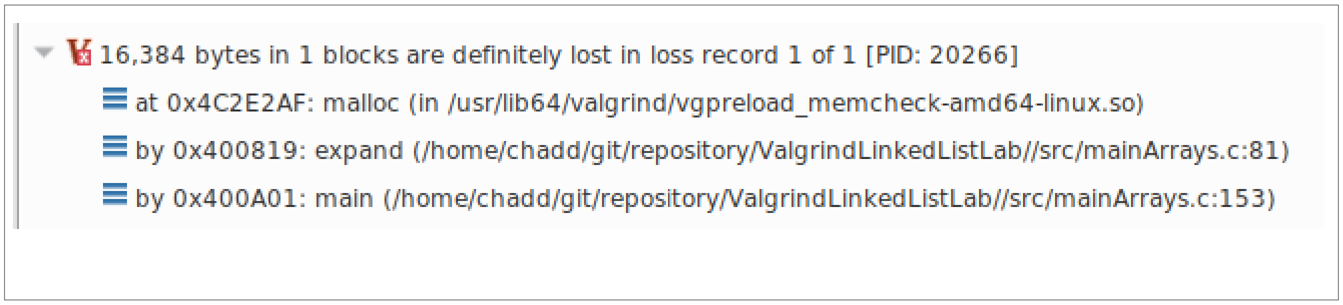


```
▼ Invalid free() / delete / delete[] / realloc() [PID: 20266]
      at 0x4C2F4DB: free (in /usr/lib64/valgrind/vgpreload_memcheck-amd64-linux.so)
      by 0x400A53: main (/home/chadd/git/repository/ValgrindLinkedListLab//src/mainArrays.c:165)
      Address 0x4 is not stack'd, malloc'd or (recently) free'd [PID: 20266]
```

Invalid free() / delete / delete[] / realloc

Invalid free indicates you have call free() on a pointer that does not correctly point to dynamically allocated memory. The pointer may have never been allocated, corrupted, or moved. Again, the highest line in the stack trace that mentions your code, line 165, is the place to start looking for the error.

This may be a difficult error to track down because line 165 tells you which pointer is being freed but you need to trace the previous usage of that pointer to determine why its value is invalid. Often it is useful to put a breakpoint on the line containing free() to allow you to inspect the address of the pointer. It also may be useful to watch the address the pointer holds during the lines previous to the free() to see if the address the pointer holds changes unexpectedly.



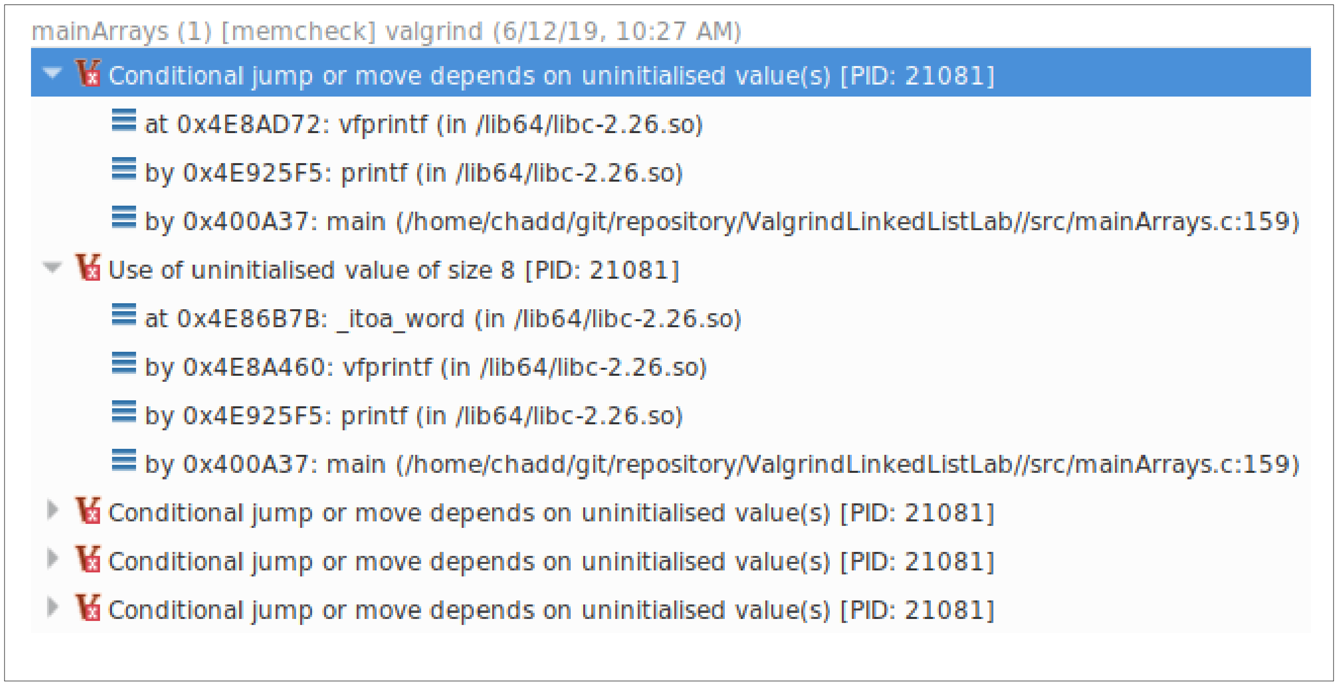
```
16,384 bytes in 1 blocks are definitely lost in loss record 1 of 1 [PID: 20266]
    at 0x4C2E2AF: malloc (in /usr/lib64/valgrind/vgpreload_memcheck-amd64-linux.so)
    by 0x400819: expand (/home/chadd/git/repository/ValgrindLinkedListLab//src/mainArrays.c:81)
    by 0x400A01: main (/home/chadd/git/repository/ValgrindLinkedListLab//src/mainArrays.c:153)
```

16,384 bytes in 1 blocks are definitely lost in loss record 1 of 1

16,384 bytes in 1 blocks are definitely lost indicates a *memory leak*: you do not free some dynamic memory you allocated before the program terminates. This could be because you never call free() altogether or you move the pointer to point to a different location without calling free(). Since we have been seeing lots of errors that touch 4 bytes at a time, you might make an educated guess and say 16384 / 4 is 4096. Does that help?

The stack trace shows you the line in your code that calls malloc() that creates the dynamic memory that is not free()d. **Your task will be to determine where that piece of dynamic memory needs to be deallocated and call free() appropriately**.

Now you should see the following five errors. We'll explore two unique types of errors.


```
mainArrays (1) [memcheck] valgrind (6/12/19, 10:27 AM)
▼ Conditional jump or move depends on uninitialised value(s) [PID: 21081]
      at 0x4E8AD72: vfprintf (in /lib64/libc-2.26.so)
      by 0x4E925F5: printf (in /lib64/libc-2.26.so)
      by 0x400A37: main (/home/chadd/git/repository/ValgrindLinkedListLab//src/mainArrays.c:159)
▼ Use of uninitialised value of size 8 [PID: 21081]
      at 0x4E86B7B: _itoa_word (in /lib64/libc-2.26.so)
      by 0x4E8A460: vfprintf (in /lib64/libc-2.26.so)
      by 0x4E925F5: printf (in /lib64/libc-2.26.so)
      by 0x400A37: main (/home/chadd/git/repository/ValgrindLinkedListLab//src/mainArrays.c:159)
▶ Conditional jump or move depends on uninitialised value(s) [PID: 21081]
▶ Conditional jump or move depends on uninitialised value(s) [PID: 21081]
▶ Conditional jump or move depends on uninitialised value(s) [PID: 21081]
```

`Conditional jump or move depends on uninitialized values(s)`

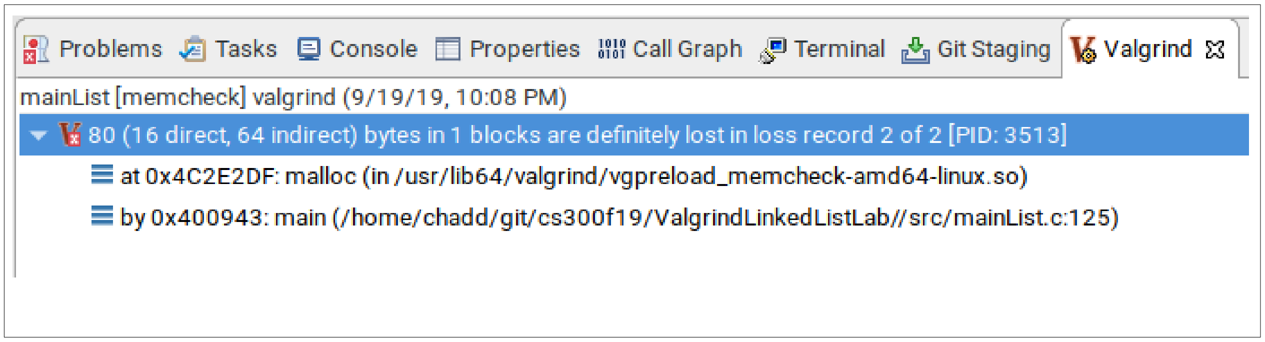
This error indicates that some uninitialized value is being used in a control structure (if/switch/loop). You can see the call stack starts at line 159 in your code. This line calls printf which calls vfprintf. Ultimately, the control structure that uses the uninitialized value is in vfprintf. However, we can guess that the data we send to printf (line 159) is the uninitialized value. **Your task is to determine why that value on line 159 is uninitialized and make sure it gets set correctly.**

`Use of uninitialized value of size 8.`

This error is similar to the above error, an uninitialized value is used in some manner. The stack trace again starts at line 159 and goes into _itoa_word via printf. Since both this error and the above error have the same starting point, and there is only one variable passed into printf, likely these errors stem from the same uninitialized data.

Be sure to expand the other **Conditional jump** errors and fix them.

mainList: The first error you see should be:



```
mainList [memcheck] valgrind (9/19/19, 10:08 PM)
  80 (16 direct, 64 indirect) bytes in 1 blocks are definitely lost in loss record 2 of 2 [PID: 3513]
      at 0x4C2E2DF: malloc (in /usr/lib64/valgrind/vgpreload_memcheck-amd64-linux.so)
      by 0x400943: main (/home/chadd/git/cs300f19/ValgrindLinkedListLab//src/mainList.c:125)
```

Be sure to use the small black arrow on the left to drop down the full stack trace.

The **Red V icon** denotes an error. The lines nested below that line describe the error. You might see multiple errors (multiple Red V icons) in one run. Generally, just start with the first error and work your way down.

80 bytes in 1 blocks are definitely lost indicates a *memory leak*: some dynamic memory does not get deallocated before the program terminates. This could be because free() is never called or the pointer moves to a different location without first calling free().

The **stack trace** (blue lines icon) shows the set of functions on the runtime stack when the error occurred. Double clicking on a line in the stack trace will take you to the particular line of code in Eclipse. The top function listed was called by the function below it, and so on. The bottom line in the stack trace (highlighted above) is where execution started in your code. *The highest up line in your code is likely where you want to start looking for your error.*

By looking at the stack trace, you can see line 125 in mainList.c is in the function main(). You can also see that line 125 calls the function malloc(). This is where the dynamic memory is allocated.

Your task is to determine when that piece of dynamic memory needs to be deallocated and call free() appropriately.

mainList also contains logic errors. Fix the logic errors and continue running Valgrind to make sure you don't introduce any more memory errors.