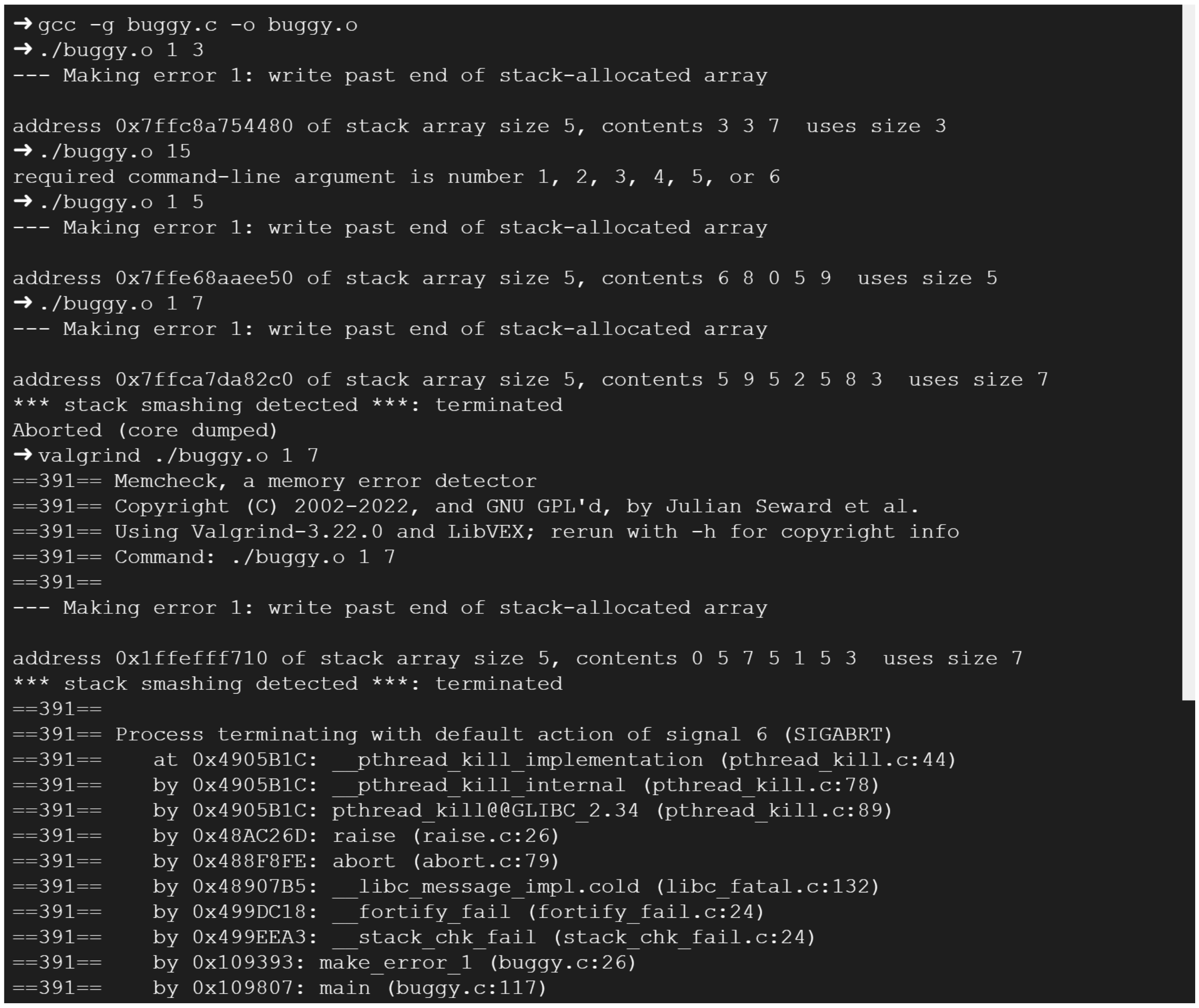
CS 211
Lab04 - Results Document

Names:Shazaib Dawood, Muneeb, Ahmed

Lab04 - Part I - Exploring Memory Leaks & Errors with valgrind

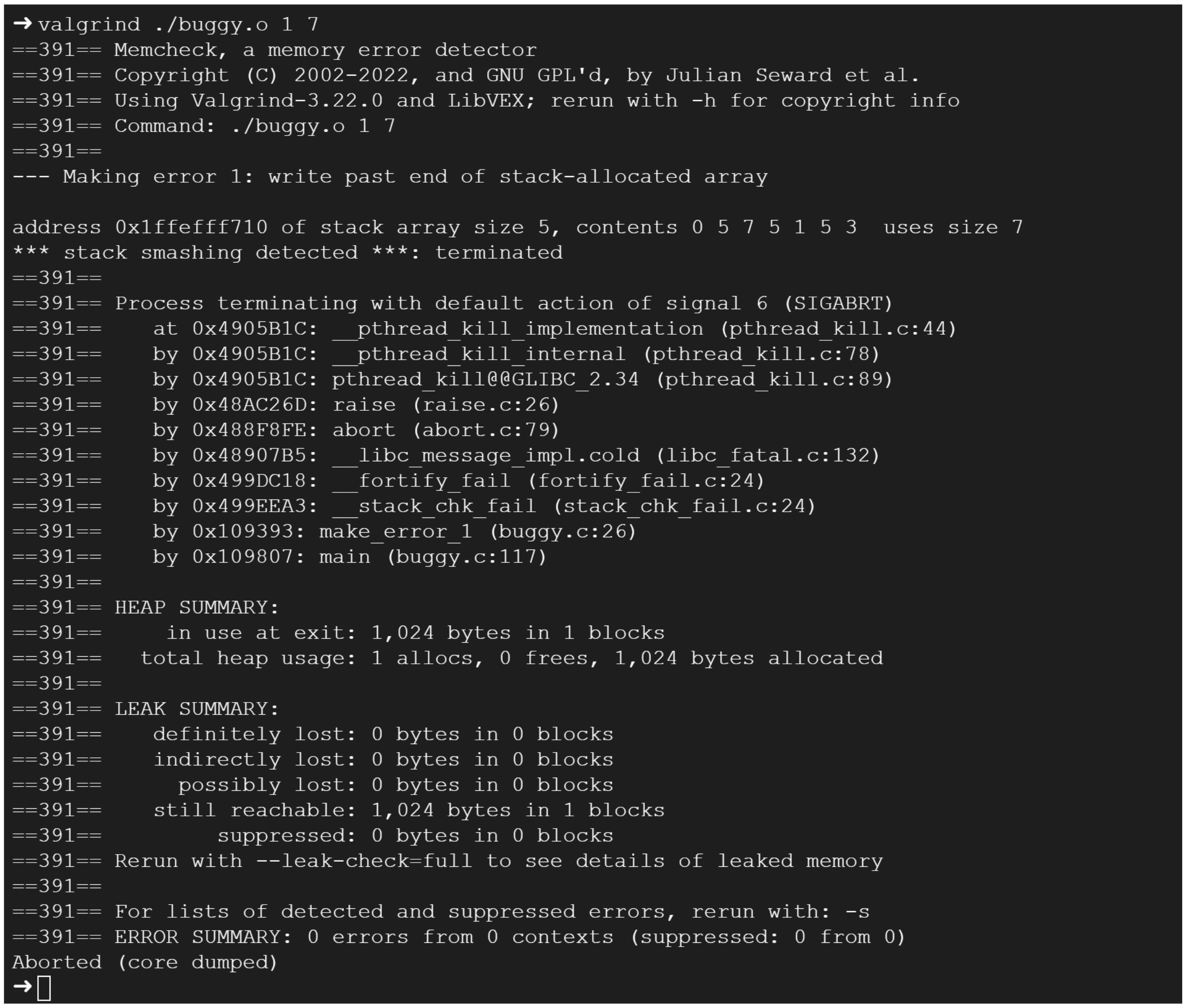
1. ***Briefly describe error #1 and provide a screenshot of the valgrind output for the case with stack smashing.***

Declaring an array of 5 elements and then assigning values to indexes outside the arrays size.



```
➜ gcc -g buggy.c -o buggy.o
➜ ./buggy.o 1 3
--- Making error 1: write past end of stack-allocated array

address 0x7ffc8a754480 of stack array size 5, contents 3 3 7  uses size 3
➜ ./buggy.o 15
required command-line argument is number 1, 2, 3, 4, 5, or 6
➜ ./buggy.o 1 5
--- Making error 1: write past end of stack-allocated array

address 0x7ffe68aaee50 of stack array size 5, contents 6 8 0 5 9  uses size 5
➜ ./buggy.o 1 7
--- Making error 1: write past end of stack-allocated array

address 0x7ffca7da82c0 of stack array size 5, contents 5 9 5 2 5 8 3  uses size 7
*** stack smashing detected ***: terminated
Aborted (core dumped)
➜ valgrind ./buggy.o 1 7
==391== Memcheck, a memory error detector
==391== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==391== Using Valgrind-3.22.0 and LibVEX; rerun with -h for copyright info
==391== Command: ./buggy.o 1 7
==391==
--- Making error 1: write past end of stack-allocated array

address 0x1ffefff710 of stack array size 5, contents 0 5 7 5 1 5 3  uses size 7
*** stack smashing detected ***: terminated
==391==
==391== Process terminating with default action of signal 6 (SIGABRT)
==391==    at 0x4905B1C: __pthread_kill_implementation (pthread_kill.c:44)
==391==    by 0x4905B1C: __pthread_kill_internal (pthread_kill.c:78)
==391==    by 0x4905B1C: pthread_kill@@GLIBC_2.34 (pthread_kill.c:89)
==391==    by 0x48AC26D: raise (raise.c:26)
==391==    by 0x488F8FE: abort (abort.c:79)
==391==    by 0x48907B5: __libc_message_impl.cold (libc_fatal.c:132)
==391==    by 0x499DC18: __fortify_fail (fortify_fail.c:24)
==391==    by 0x499EEA3: __stack_chk_fail (stack_chk_fail.c:24)
==391==    by 0x109393: make_error_1 (buggy.c:26)
==391==    by 0x109807: main (buggy.c:117)
```

CS 211
Lab04 - Results Document



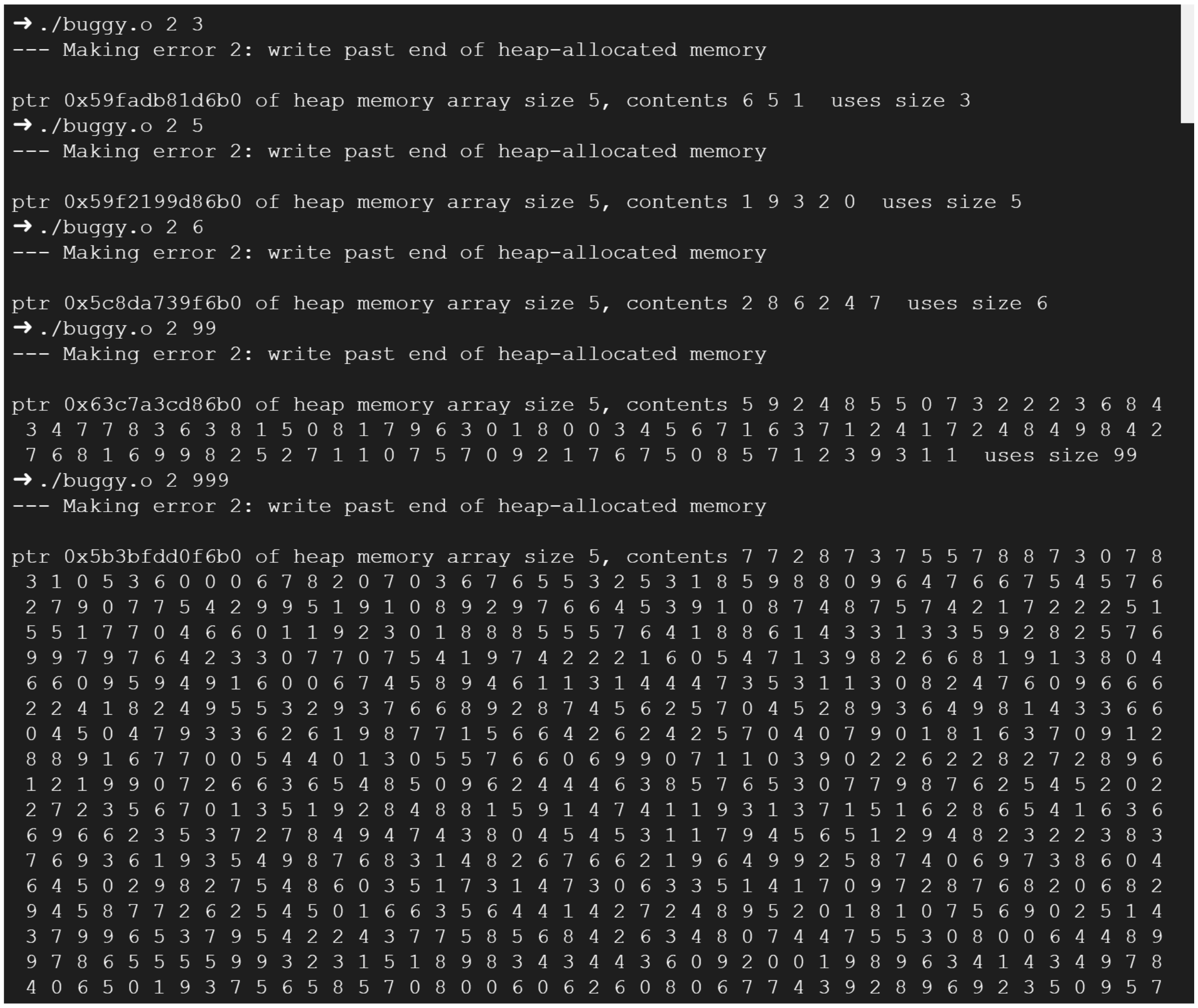
```
➜ valgrind ./buggy.o 1 7
==391== Memcheck, a memory error detector
==391== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==391== Using Valgrind-3.22.0 and LibVEX; rerun with -h for copyright info
==391== Command: ./buggy.o 1 7
==391==
--- Making error 1: write past end of stack-allocated array

address 0x1ffefff710 of stack array size 5, contents 0 5 7 5 1 5 3  uses size 7
*** stack smashing detected ***: terminated
==391==
==391== Process terminating with default action of signal 6 (SIGABRT)
==391==    at 0x4905B1C: __pthread_kill_implementation (pthread_kill.c:44)
==391==    by 0x4905B1C: __pthread_kill_internal (pthread_kill.c:78)
==391==    by 0x4905B1C: pthread_kill@@GLIBC_2.34 (pthread_kill.c:89)
==391==    by 0x48AC26D: raise (raise.c:26)
==391==    by 0x488F8FE: abort (abort.c:79)
==391==    by 0x48907B5: __libc_message_impl.cold (libc_fatal.c:132)
==391==    by 0x499DC18: __fortify_fail (fortify_fail.c:24)
==391==    by 0x499EEA3: __stack_chk_fail (stack_chk_fail.c:24)
==391==    by 0x109393: make_error_1 (buggy.c:26)
==391==    by 0x109807: main (buggy.c:117)
==391==
==391== HEAP SUMMARY:
==391==     in use at exit: 1,024 bytes in 1 blocks
==391==   total heap usage: 1 allocs, 0 frees, 1,024 bytes allocated
==391==
==391== LEAK SUMMARY:
==391==    definitely lost: 0 bytes in 0 blocks
==391==    indirectly lost: 0 bytes in 0 blocks
==391==      possibly lost: 0 bytes in 0 blocks
==391==    still reachable: 1,024 bytes in 1 blocks
==391==         suppressed: 0 bytes in 0 blocks
==391== Rerun with --leak-check=full to see details of leaked memory
==391==
==391== For lists of detected and suppressed errors, rerun with: -s
==391== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
Aborted (core dumped)
➜
```

2. ***Briefly describe error #2, report the large size needed to get a segmentation fault, and provide a screenshot of the valgrind output for the case with a small overrun.***

   99999 was needed to cause a segmentation fault.

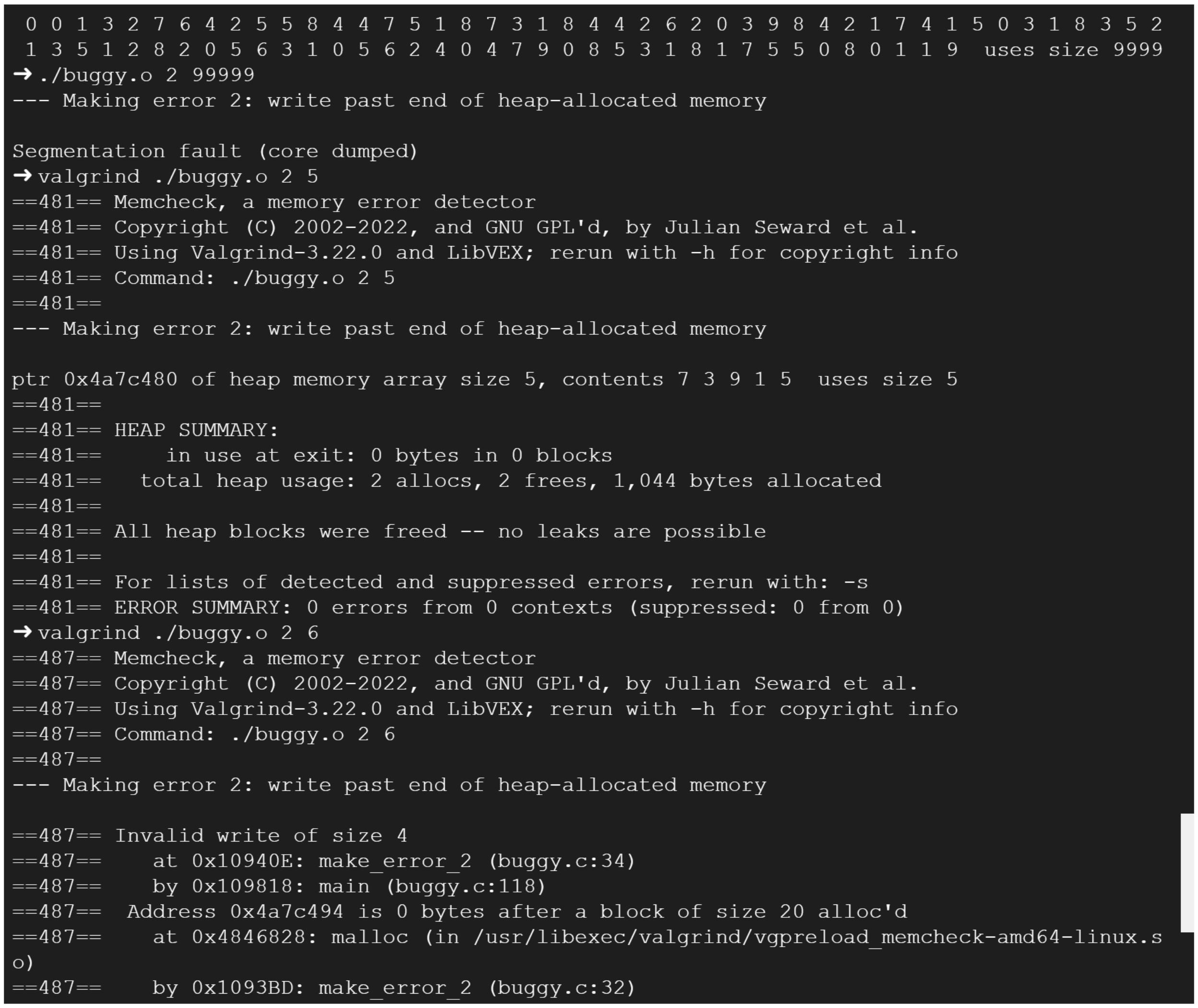
   Error occurred because we were accessing elements outside the size that was dynamically allocated in the heap.

CS 211
Lab04 - Results Document



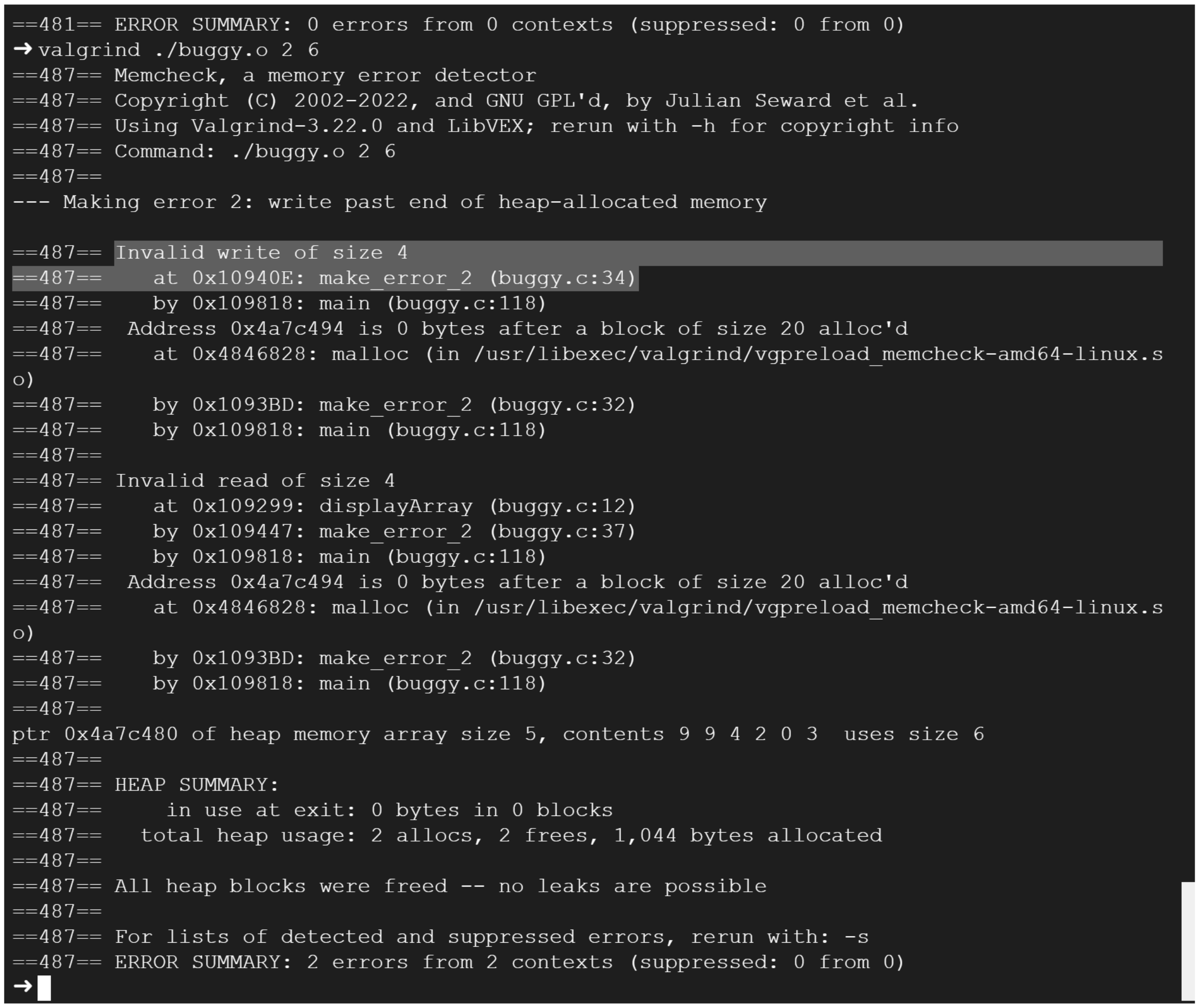
```
➜ ./buggy.o 2 3
--- Making error 2: write past end of heap-allocated memory

ptr 0x59fadb81d6b0 of heap memory array size 5, contents 6 5 1  uses size 3
➜ ./buggy.o 2 5
--- Making error 2: write past end of heap-allocated memory

ptr 0x59f2199d86b0 of heap memory array size 5, contents 1 9 3 2 0  uses size 5
➜ ./buggy.o 2 6
--- Making error 2: write past end of heap-allocated memory

ptr 0x5c8da739f6b0 of heap memory array size 5, contents 2 8 6 2 4 7  uses size 6
➜ ./buggy.o 2 99
--- Making error 2: write past end of heap-allocated memory

ptr 0x63c7a3cd86b0 of heap memory array size 5, contents 5 9 2 4 8 5 5 0 7 3 2 2 2 3 6 8 4
 3 4 7 7 8 3 6 3 8 1 5 0 8 1 7 9 6 3 0 1 8 0 0 3 4 5 6 7 1 6 3 7 1 2 4 1 7 2 4 8 4 9 8 4 2
 7 6 8 1 6 9 9 8 2 5 2 7 1 1 0 7 5 7 0 9 2 1 7 6 7 5 0 8 5 7 1 2 3 9 3 1 1  uses size 99
➜ ./buggy.o 2 999
--- Making error 2: write past end of heap-allocated memory

ptr 0x5b3bfdd0f6b0 of heap memory array size 5, contents 7 7 2 8 7 3 7 5 5 7 8 8 7 3 0 7 8
 3 1 0 5 3 6 0 0 0 6 7 8 2 0 7 0 3 6 7 6 5 5 3 2 5 3 1 8 5 9 8 8 0 9 6 4 7 6 6 7 5 4 5 7 6
 2 7 9 0 7 7 5 4 2 9 9 5 1 9 1 0 8 9 2 9 7 6 6 4 5 3 9 1 0 8 7 4 8 7 5 7 4 2 1 7 2 2 2 5 1
 5 5 1 7 7 0 4 6 6 0 1 1 9 2 3 0 1 8 8 8 5 5 5 7 6 4 1 8 8 6 1 4 3 3 1 3 3 5 9 2 8 2 5 7 6
 9 9 7 9 7 6 4 2 3 3 0 7 7 0 7 5 4 1 9 7 4 2 2 2 1 6 0 5 4 7 1 3 9 8 2 6 6 8 1 9 1 3 8 0 4
 6 6 0 9 5 9 4 9 1 6 0 0 6 7 4 5 8 9 4 6 1 1 3 1 4 4 4 7 3 5 3 1 1 3 0 8 2 4 7 6 0 9 6 6 6
 2 2 4 1 8 2 4 9 5 5 3 2 9 3 7 6 6 8 9 2 8 7 4 5 6 2 5 7 0 4 5 2 8 9 3 6 4 9 8 1 4 3 3 6 6
 0 4 5 0 4 7 9 3 3 6 2 6 1 9 8 7 7 1 5 6 6 4 2 6 2 4 2 5 7 0 4 0 7 9 0 1 8 1 6 3 7 0 9 1 2
 8 8 9 1 6 7 7 0 0 5 4 4 0 1 3 0 5 5 7 6 6 0 6 9 9 0 7 1 1 0 3 9 0 2 2 6 2 2 8 2 7 2 8 9 6
 1 2 1 9 9 0 7 2 6 6 3 6 5 4 8 5 0 9 6 2 4 4 4 6 3 8 5 7 6 5 3 0 7 7 9 8 7 6 2 5 4 5 2 0 2
 2 7 2 3 5 6 7 0 1 3 5 1 9 2 8 4 8 8 1 5 9 1 4 7 4 1 1 9 3 1 3 7 1 5 1 6 2 8 6 5 4 1 6 3 6
 6 9 6 6 2 3 5 3 7 2 7 8 4 9 4 7 4 3 8 0 4 5 4 5 3 1 1 7 9 4 5 6 5 1 2 9 4 8 2 3 2 2 3 8 3
 7 6 9 3 6 1 9 3 5 4 9 8 7 6 8 3 1 4 8 2 6 7 6 6 2 1 9 6 4 9 9 2 5 8 7 4 0 6 9 7 3 8 6 0 4
 6 4 5 0 2 9 8 2 7 5 4 8 6 0 3 5 1 7 3 1 4 7 3 0 6 3 3 5 1 4 1 7 0 9 7 2 8 7 6 8 2 0 6 8 2
 9 4 5 8 7 7 2 6 2 5 4 5 0 1 6 6 3 5 6 4 4 1 4 2 7 2 4 8 9 5 2 0 1 8 1 0 7 5 6 9 0 2 5 1 4
 3 7 9 9 6 5 3 7 9 5 4 2 2 4 3 7 7 5 8 5 6 8 4 2 6 3 4 8 0 7 4 4 7 5 5 3 0 8 0 0 6 4 4 8 9
 9 7 8 6 5 5 5 5 9 9 3 2 3 1 5 1 8 9 8 3 4 3 4 4 3 6 0 9 2 0 0 1 9 8 9 6 3 4 1 4 3 4 9 7 8
 4 0 6 5 0 1 9 3 7 5 6 5 8 5 7 0 8 0 0 6 0 6 2 6 0 8 0 6 7 7 4 3 9 2 8 9 6 9 2 3 5 0 9 5 7
```

CS 211
Lab04 - Results Document



```
 0 0 1 3 2 7 6 4 2 5 5 8 4 4 7 5 1 8 7 3 1 8 4 4 2 6 2 0 3 9 8 4 2 1 7 4 1 5 0 3 1 8 3 5 2
 1 3 5 1 2 8 2 0 5 6 3 1 0 5 6 2 4 0 4 7 9 0 8 5 3 1 8 1 7 5 5 0 8 0 1 1 9  uses size 9999
➜ ./buggy.o 2 99999
--- Making error 2: write past end of heap-allocated memory

Segmentation fault (core dumped)
➜ valgrind ./buggy.o 2 5
==481== Memcheck, a memory error detector
==481== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==481== Using Valgrind-3.22.0 and LibVEX; rerun with -h for copyright info
==481== Command: ./buggy.o 2 5
==481==
--- Making error 2: write past end of heap-allocated memory

ptr 0x4a7c480 of heap memory array size 5, contents 7 3 9 1 5  uses size 5
==481==
==481== HEAP SUMMARY:
==481==     in use at exit: 0 bytes in 0 blocks
==481==   total heap usage: 2 allocs, 2 frees, 1,044 bytes allocated
==481==
==481== All heap blocks were freed -- no leaks are possible
==481==
==481== For lists of detected and suppressed errors, rerun with: -s
==481== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
➜ valgrind ./buggy.o 2 6
==487== Memcheck, a memory error detector
==487== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==487== Using Valgrind-3.22.0 and LibVEX; rerun with -h for copyright info
==487== Command: ./buggy.o 2 6
==487==
--- Making error 2: write past end of heap-allocated memory

==487== Invalid write of size 4
==487==    at 0x10940E: make_error_2 (buggy.c:34)
==487==    by 0x109818: main (buggy.c:118)
==487==  Address 0x4a7c494 is 0 bytes after a block of size 20 alloc'd
==487==    at 0x4846828: malloc (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.s
o)
==487==    by 0x1093BD: make_error_2 (buggy.c:32)
```

CS 211
Lab04 - Results Document

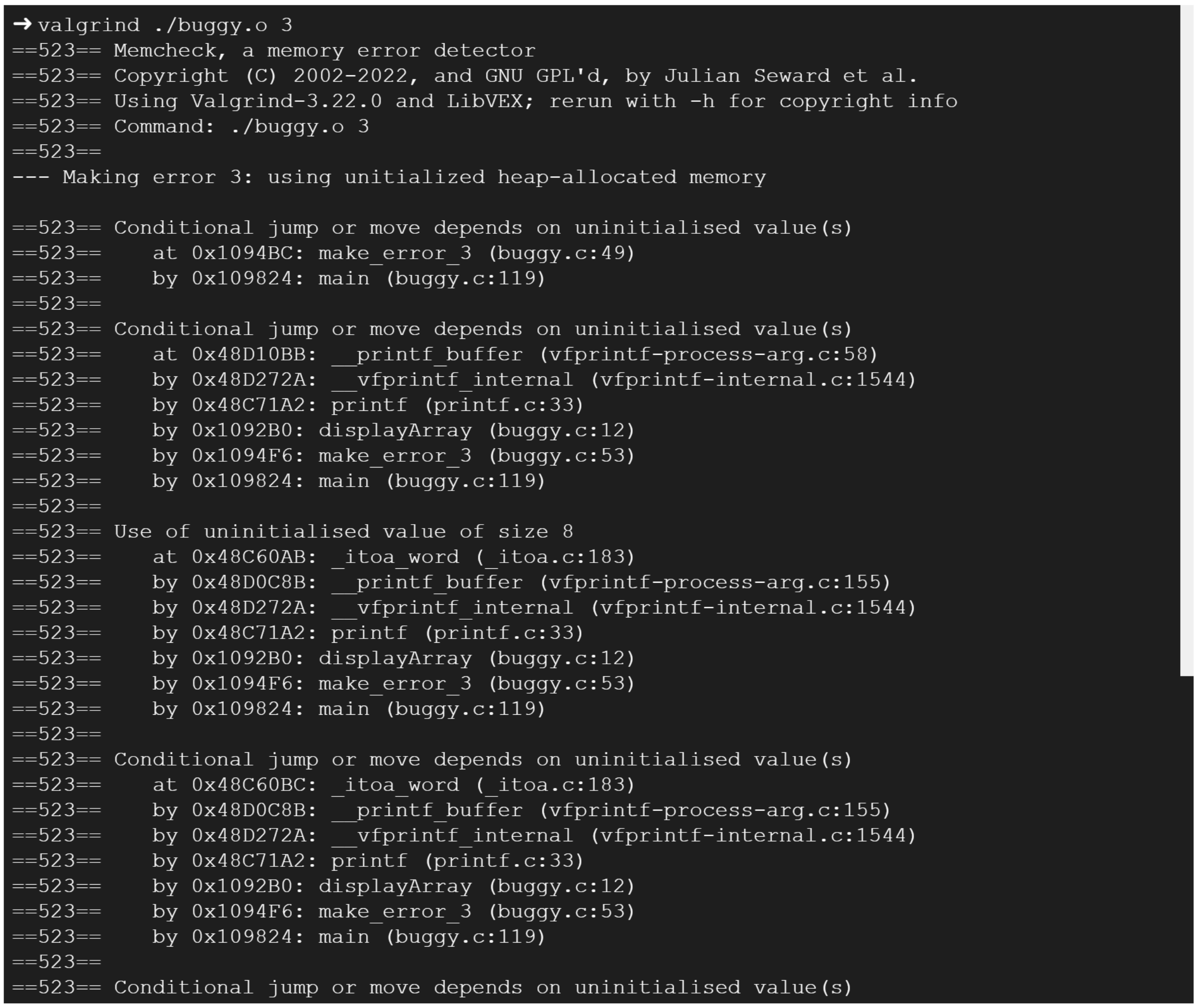


```
==481== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
➜ valgrind ./buggy.o 2 6
==487== Memcheck, a memory error detector
==487== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==487== Using Valgrind-3.22.0 and LibVEX; rerun with -h for copyright info
==487== Command: ./buggy.o 2 6
==487==
--- Making error 2: write past end of heap-allocated memory

==487== Invalid write of size 4
==487==    at 0x10940E: make_error_2 (buggy.c:34)
==487==    by 0x109818: main (buggy.c:118)
==487==  Address 0x4a7c494 is 0 bytes after a block of size 20 alloc'd
==487==    at 0x4846828: malloc (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.s
o)
==487==    by 0x1093BD: make_error_2 (buggy.c:32)
==487==    by 0x109818: main (buggy.c:118)
==487==
==487== Invalid read of size 4
==487==    at 0x109299: displayArray (buggy.c:12)
==487==    by 0x109447: make_error_2 (buggy.c:37)
==487==    by 0x109818: main (buggy.c:118)
==487==  Address 0x4a7c494 is 0 bytes after a block of size 20 alloc'd
==487==    at 0x4846828: malloc (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.s
o)
==487==    by 0x1093BD: make_error_2 (buggy.c:32)
==487==    by 0x109818: main (buggy.c:118)
==487==
ptr 0x4a7c480 of heap memory array size 5, contents 9 9 4 2 0 3  uses size 6
==487==
==487== HEAP SUMMARY:
==487==     in use at exit: 0 bytes in 0 blocks
==487==   total heap usage: 2 allocs, 2 frees, 1,044 bytes allocated
==487==
==487== All heap blocks were freed -- no leaks are possible
==487==
==487== For lists of detected and suppressed errors, rerun with: -s
==487== ERROR SUMMARY: 2 errors from 2 contexts (suppressed: 0 from 0)
➜
```

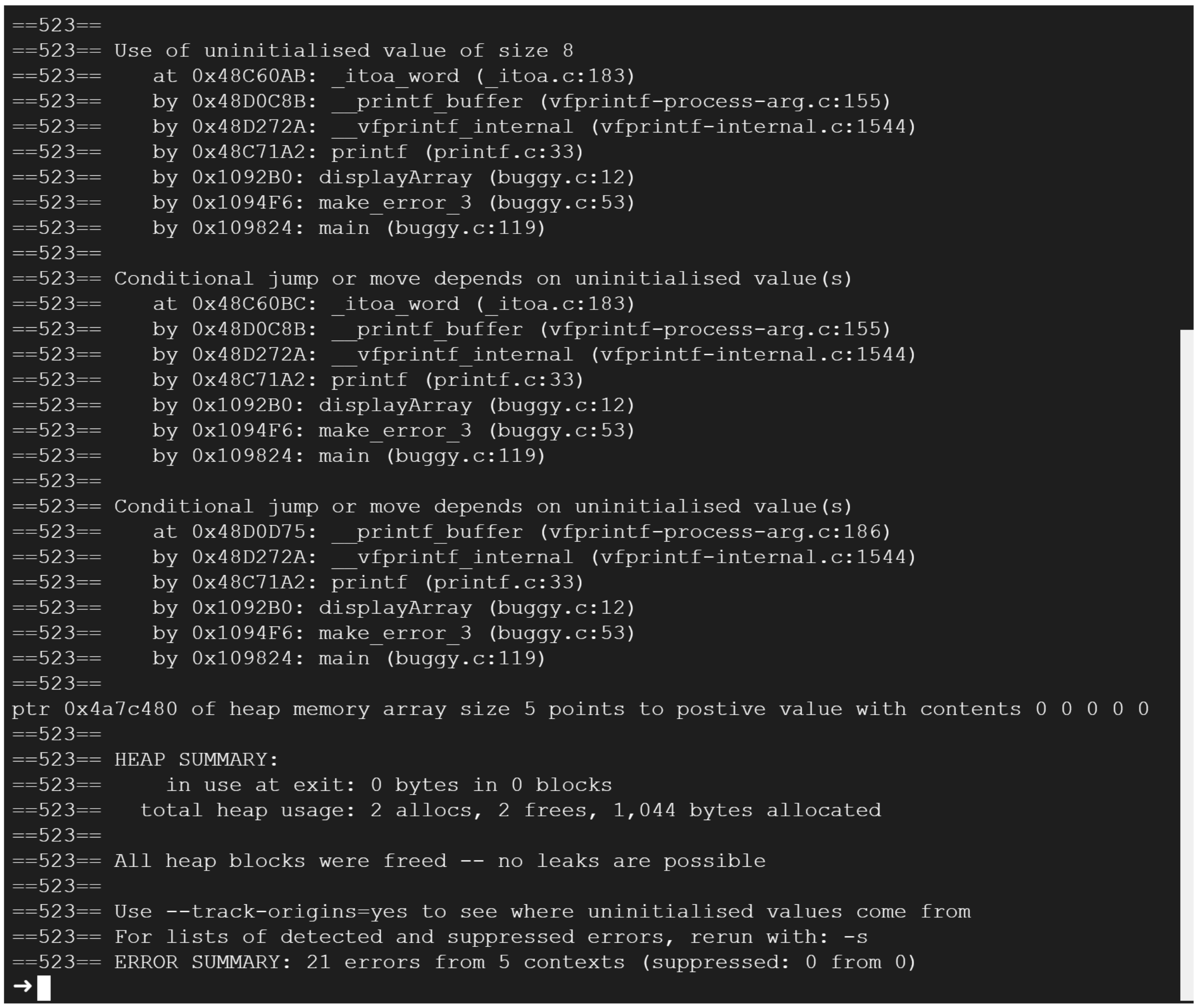
3. ***Briefly describe error #3 and provide a screenshot of the valgrind output.***

   Element of the array is being accessed without any values being assigned to it.

CS 211
Lab04 - Results Document



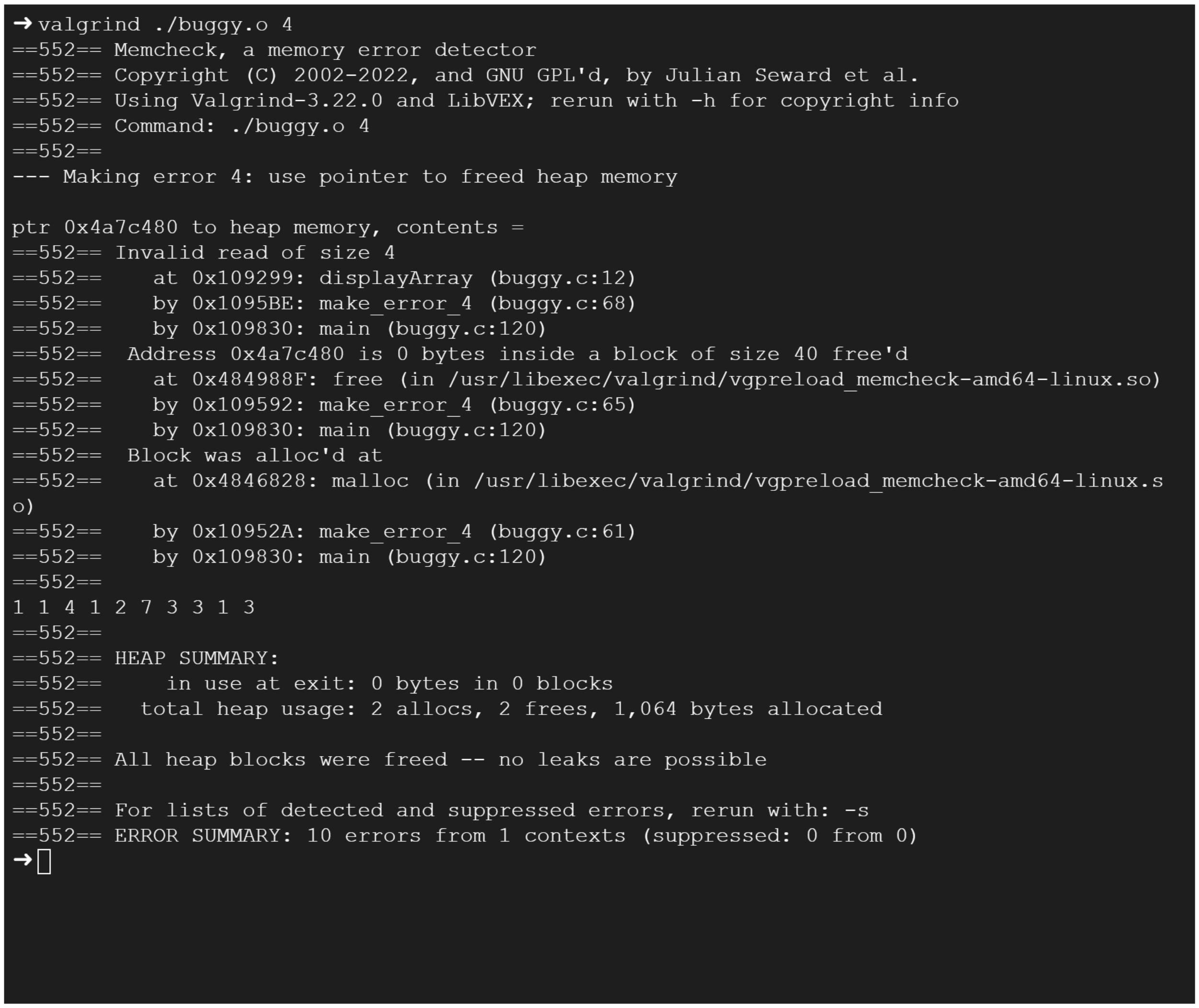
```
➜ valgrind ./buggy.o 3
==523== Memcheck, a memory error detector
==523== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==523== Using Valgrind-3.22.0 and LibVEX; rerun with -h for copyright info
==523== Command: ./buggy.o 3
==523==
--- Making error 3: using unitialized heap-allocated memory

==523== Conditional jump or move depends on uninitialised value(s)
==523==    at 0x1094BC: make_error_3 (buggy.c:49)
==523==    by 0x109824: main (buggy.c:119)
==523==
==523== Conditional jump or move depends on uninitialised value(s)
==523==    at 0x48D10BB: __printf_buffer (vfprintf-process-arg.c:58)
==523==    by 0x48D272A: __vfprintf_internal (vfprintf-internal.c:1544)
==523==    by 0x48C71A2: printf (printf.c:33)
==523==    by 0x1092B0: displayArray (buggy.c:12)
==523==    by 0x1094F6: make_error_3 (buggy.c:53)
==523==    by 0x109824: main (buggy.c:119)
==523==
==523== Use of uninitialised value of size 8
==523==    at 0x48C60AB: _itoa_word (_itoa.c:183)
==523==    by 0x48D0C8B: __printf_buffer (vfprintf-process-arg.c:155)
==523==    by 0x48D272A: __vfprintf_internal (vfprintf-internal.c:1544)
==523==    by 0x48C71A2: printf (printf.c:33)
==523==    by 0x1092B0: displayArray (buggy.c:12)
==523==    by 0x1094F6: make_error_3 (buggy.c:53)
==523==    by 0x109824: main (buggy.c:119)
==523==
==523== Conditional jump or move depends on uninitialised value(s)
==523==    at 0x48C60BC: _itoa_word (_itoa.c:183)
==523==    by 0x48D0C8B: __printf_buffer (vfprintf-process-arg.c:155)
==523==    by 0x48D272A: __vfprintf_internal (vfprintf-internal.c:1544)
==523==    by 0x48C71A2: printf (printf.c:33)
==523==    by 0x1092B0: displayArray (buggy.c:12)
==523==    by 0x1094F6: make_error_3 (buggy.c:53)
==523==    by 0x109824: main (buggy.c:119)
==523==
==523== Conditional jump or move depends on uninitialised value(s)
```



```
==523==
==523== Use of uninitialised value of size 8
==523==    at 0x48C60AB: _itoa_word (_itoa.c:183)
==523==    by 0x48D0C8B: __printf_buffer (vfprintf-process-arg.c:155)
==523==    by 0x48D272A: __vfprintf_internal (vfprintf-internal.c:1544)
==523==    by 0x48C71A2: printf (printf.c:33)
==523==    by 0x1092B0: displayArray (buggy.c:12)
==523==    by 0x1094F6: make_error_3 (buggy.c:53)
==523==    by 0x109824: main (buggy.c:119)
==523==
==523== Conditional jump or move depends on uninitialised value(s)
==523==    at 0x48C60BC: _itoa_word (_itoa.c:183)
==523==    by 0x48D0C8B: __printf_buffer (vfprintf-process-arg.c:155)
==523==    by 0x48D272A: __vfprintf_internal (vfprintf-internal.c:1544)
==523==    by 0x48C71A2: printf (printf.c:33)
==523==    by 0x1092B0: displayArray (buggy.c:12)
==523==    by 0x1094F6: make_error_3 (buggy.c:53)
==523==    by 0x109824: main (buggy.c:119)
==523==
==523== Conditional jump or move depends on uninitialised value(s)
==523==    at 0x48D0D75: __printf_buffer (vfprintf-process-arg.c:186)
==523==    by 0x48D272A: __vfprintf_internal (vfprintf-internal.c:1544)
==523==    by 0x48C71A2: printf (printf.c:33)
==523==    by 0x1092B0: displayArray (buggy.c:12)
==523==    by 0x1094F6: make_error_3 (buggy.c:53)
==523==    by 0x109824: main (buggy.c:119)
==523==
ptr 0x4a7c480 of heap memory array size 5 points to postive value with contents 0 0 0 0 0
==523==
==523== HEAP SUMMARY:
==523==     in use at exit: 0 bytes in 0 blocks
==523==   total heap usage: 2 allocs, 2 frees, 1,044 bytes allocated
==523==
==523== All heap blocks were freed -- no leaks are possible
==523==
==523== Use --track-origins=yes to see where uninitialised values come from
==523== For lists of detected and suppressed errors, rerun with: -s
==523== ERROR SUMMARY: 21 errors from 5 contexts (suppressed: 0 from 0)
➜
```

4. ***Briefly describe error #4 and provide a screenshot of the valgrind output.***

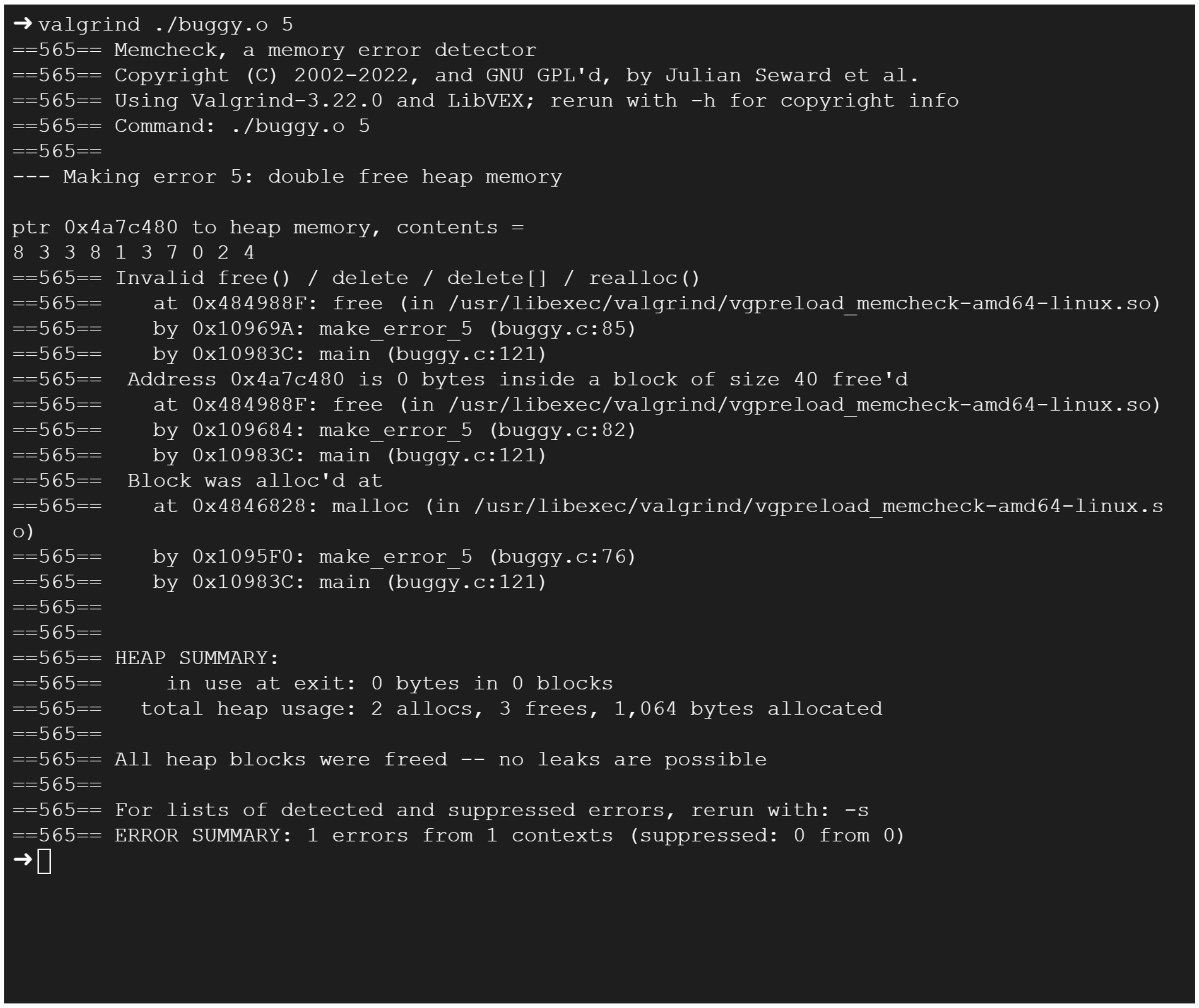
   Memory is being used after being freed.



```
➜ valgrind ./buggy.o 4
==552== Memcheck, a memory error detector
==552== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==552== Using Valgrind-3.22.0 and LibVEX; rerun with -h for copyright info
==552== Command: ./buggy.o 4
==552==
--- Making error 4: use pointer to freed heap memory

ptr 0x4a7c480 to heap memory, contents =
==552== Invalid read of size 4
==552==    at 0x109299: displayArray (buggy.c:12)
==552==    by 0x1095BE: make_error_4 (buggy.c:68)
==552==    by 0x109830: main (buggy.c:120)
==552==  Address 0x4a7c480 is 0 bytes inside a block of size 40 free'd
==552==    at 0x484988F: free (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==552==    by 0x109592: make_error_4 (buggy.c:65)
==552==    by 0x109830: main (buggy.c:120)
==552==  Block was alloc'd at
==552==    at 0x4846828: malloc (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.s
o)
==552==    by 0x10952A: make_error_4 (buggy.c:61)
==552==    by 0x109830: main (buggy.c:120)
==552==
1 1 4 1 2 7 3 3 1 3
==552==
==552== HEAP SUMMARY:
==552==     in use at exit: 0 bytes in 0 blocks
==552==   total heap usage: 2 allocs, 2 frees, 1,064 bytes allocated
==552==
==552== All heap blocks were freed -- no leaks are possible
==552==
==552== For lists of detected and suppressed errors, rerun with: -s
==552== ERROR SUMMARY: 10 errors from 1 contexts (suppressed: 0 from 0)
➜
```

5. ***Briefly describe error #5 and provide a screenshot of the valgrind output.***

   Dynamically allocated memory is being freed twice.

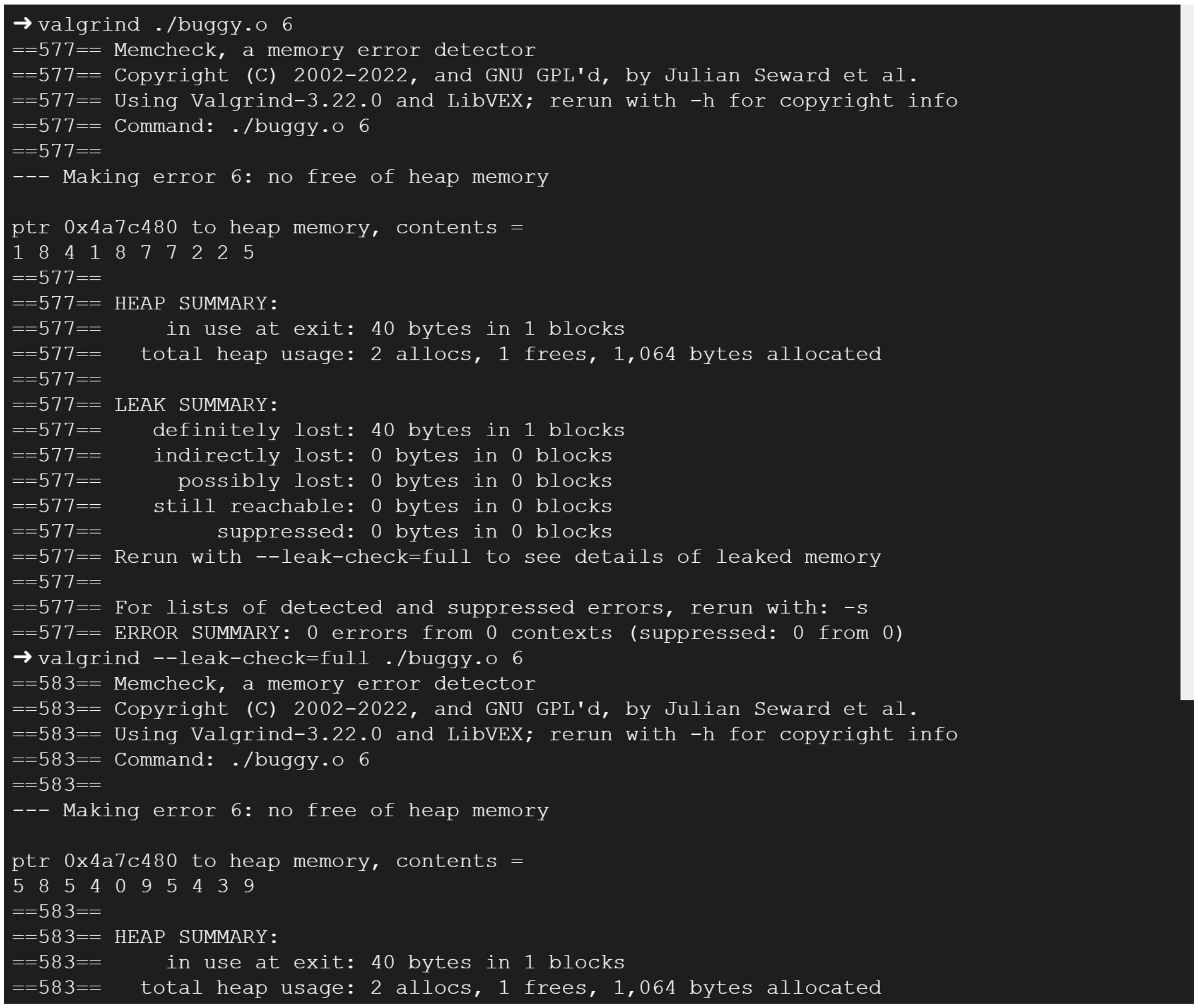


```
➜ valgrind ./buggy.o 5
==565== Memcheck, a memory error detector
==565== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==565== Using Valgrind-3.22.0 and LibVEX; rerun with -h for copyright info
==565== Command: ./buggy.o 5
==565==
--- Making error 5: double free heap memory

ptr 0x4a7c480 to heap memory, contents =
8 3 3 8 1 3 7 0 2 4
==565== Invalid free() / delete / delete[] / realloc()
==565==    at 0x484988F: free (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==565==    by 0x10969A: make_error_5 (buggy.c:85)
==565==    by 0x10983C: main (buggy.c:121)
==565==  Address 0x4a7c480 is 0 bytes inside a block of size 40 free'd
==565==    at 0x484988F: free (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==565==    by 0x109684: make_error_5 (buggy.c:82)
==565==    by 0x10983C: main (buggy.c:121)
==565==  Block was alloc'd at
==565==    at 0x4846828: malloc (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.s
o)
==565==    by 0x1095F0: make_error_5 (buggy.c:76)
==565==    by 0x10983C: main (buggy.c:121)
==565==
==565==
==565== HEAP SUMMARY:
==565==     in use at exit: 0 bytes in 0 blocks
==565==   total heap usage: 2 allocs, 3 frees, 1,064 bytes allocated
==565==
==565== All heap blocks were freed -- no leaks are possible
==565==
==565== For lists of detected and suppressed errors, rerun with: -s
==565== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
➜ 
```

6. ***Briefly describe error #6 and provide a screenshot of the valgrind output.***

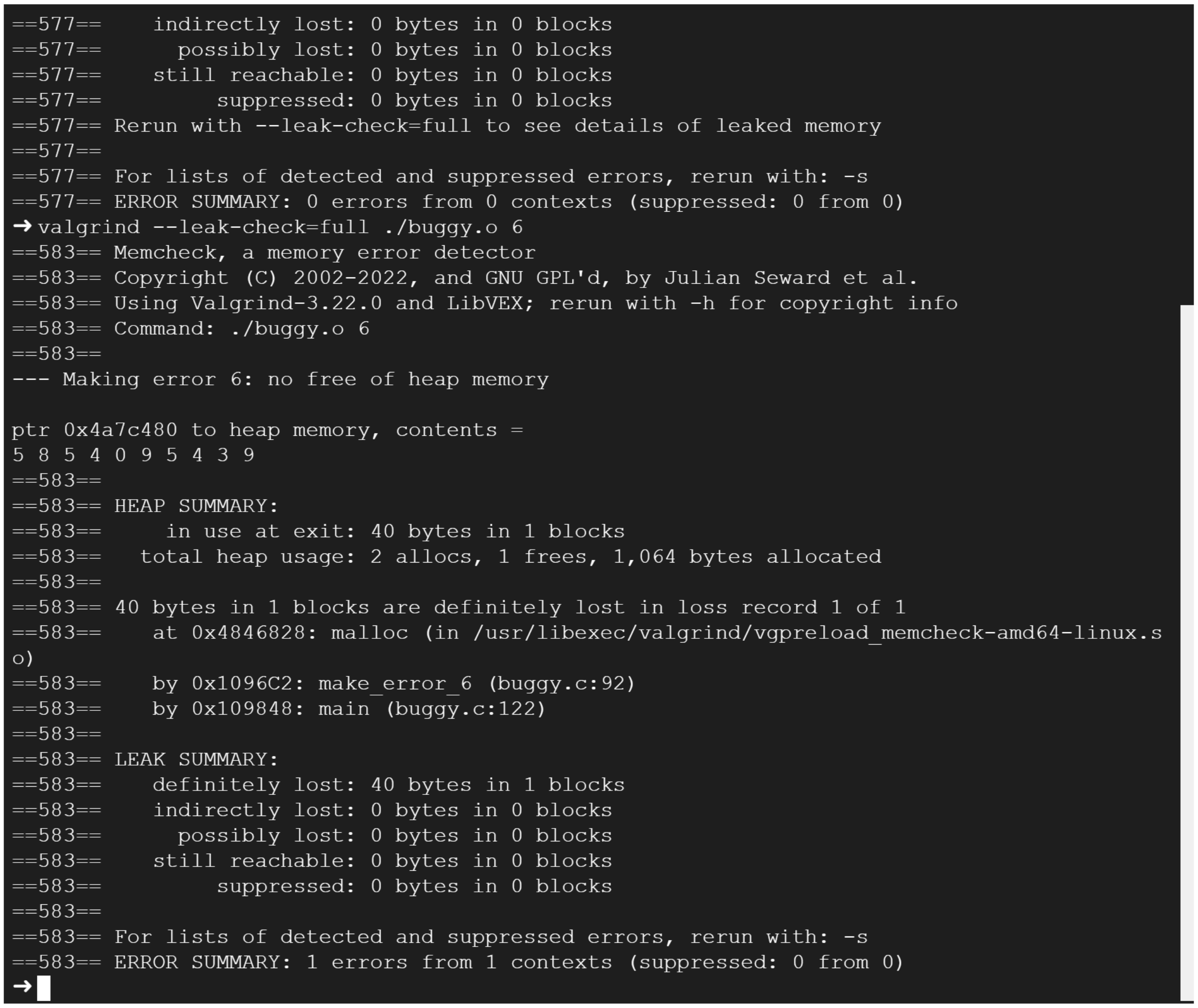
Dynamically allocated memory is not benign freed at program termination.

CS 211
Lab04 - Results Document



```
➜ valgrind ./buggy.o 6
==577== Memcheck, a memory error detector
==577== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==577== Using Valgrind-3.22.0 and LibVEX; rerun with -h for copyright info
==577== Command: ./buggy.o 6
==577==
--- Making error 6: no free of heap memory

ptr 0x4a7c480 to heap memory, contents =
1 8 4 1 8 7 7 2 2 5
==577==
==577== HEAP SUMMARY:
==577==     in use at exit: 40 bytes in 1 blocks
==577==   total heap usage: 2 allocs, 1 frees, 1,064 bytes allocated
==577==
==577== LEAK SUMMARY:
==577==    definitely lost: 40 bytes in 1 blocks
==577==    indirectly lost: 0 bytes in 0 blocks
==577==      possibly lost: 0 bytes in 0 blocks
==577==    still reachable: 0 bytes in 0 blocks
==577==         suppressed: 0 bytes in 0 blocks
==577== Rerun with --leak-check=full to see details of leaked memory
==577==
==577== For lists of detected and suppressed errors, rerun with: -s
==577== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
➜ valgrind --leak-check=full ./buggy.o 6
==583== Memcheck, a memory error detector
==583== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==583== Using Valgrind-3.22.0 and LibVEX; rerun with -h for copyright info
==583== Command: ./buggy.o 6
==583==
--- Making error 6: no free of heap memory

ptr 0x4a7c480 to heap memory, contents =
5 8 5 4 0 9 5 4 3 9
==583==
==583== HEAP SUMMARY:
==583==     in use at exit: 40 bytes in 1 blocks
==583==   total heap usage: 2 allocs, 1 frees, 1,064 bytes allocated
```

CS 211
Lab04 - Results Document



```
==577==    indirectly lost: 0 bytes in 0 blocks
==577==      possibly lost: 0 bytes in 0 blocks
==577==    still reachable: 0 bytes in 0 blocks
==577==         suppressed: 0 bytes in 0 blocks
==577== Rerun with --leak-check=full to see details of leaked memory
==577==
==577== For lists of detected and suppressed errors, rerun with: -s
==577== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
➜ valgrind --leak-check=full ./buggy.o 6
==583== Memcheck, a memory error detector
==583== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==583== Using Valgrind-3.22.0 and LibVEX; rerun with -h for copyright info
==583== Command: ./buggy.o 6
==583==
--- Making error 6: no free of heap memory

ptr 0x4a7c480 to heap memory, contents =
5 8 5 4 0 9 5 4 3 9
==583==
==583== HEAP SUMMARY:
==583==     in use at exit: 40 bytes in 1 blocks
==583==   total heap usage: 2 allocs, 1 frees, 1,064 bytes allocated
==583==
==583== 40 bytes in 1 blocks are definitely lost in loss record 1 of 1
==583==    at 0x4846828: malloc (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.s
o)
==583==    by 0x1096C2: make_error_6 (buggy.c:92)
==583==    by 0x109848: main (buggy.c:122)
==583==
==583== LEAK SUMMARY:
==583==    definitely lost: 40 bytes in 1 blocks
==583==    indirectly lost: 0 bytes in 0 blocks
==583==      possibly lost: 0 bytes in 0 blocks
==583==    still reachable: 0 bytes in 0 blocks
==583==         suppressed: 0 bytes in 0 blocks
==583==
==583== For lists of detected and suppressed errors, rerun with: -s
==583== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
➜
```

CS 211
Lab04 - Results Document

Lab04 - Part II - Arrays vs. Pointers with gdb

Steps 1 & 2 - expected vs. actual output for gdb commands

Consider the following declarations:

```
int main(void) {
        int arr[] = {0, 10, 20, 30, 40, 50};
        int *ptr = arr;
        //put break here
        return 0;
}
```

After these declarations, suppose the array has the memory address location **0x7fffb63b6080**.

Then, gdb is run with a break on the line identified in the comment. The following gdb commands are then run. Complete the table below by first filling in the expected output and then actually run gdb to determine the actual output. Also, the memory addresses will be different for your actual run.

It is okay if the expected output column is not correct; this exercise is about clearing up confusions related to array names and pointers; i.e. in what ways are array names and pointers the same, and how are array names and pointers different?

Note: **`(gdb) p arr`** is identical to **`(gdb) print arr`**

gdb command	expected output	actual output
`(gdb) p &arr[0]`	0x7fffb63b6080	0x7ffc67181970
`(gdb) p arr`	0	{0, 10, 20, 30, 40, 50}
`(gdb) p arr[1]`	10	10
`(gdb) p &arr[1]`	0x7fffb63b6084	0x7ffc67181974
`(gdb) p arr + 2`	2	0x7ffc67181978
`(gdb) p &arr[3] - &arr[1]`	0x7fffb63b6008	2
`(gdb) p sizeof(arr)`	24	24
`(gdb) p arr = arr + 1`	1	Invalid cast

CS 211
Lab04 - Results Document

CS 211
Lab04 - Results Document

gdb command	expected output	actual output
`(gdb) p ptr`	0	0x7ffc67181970
`(gdb) p ptr[1]`	10	10
`(gdb) p &ptr[1]`	0x7fffb63b6084	0x7ffc67181974
`(gdb) p ptr + 2`	2	0x7ffc67181978
`(gdb) p &ptr[3] - &ptr[1]`	0x7fffb63b6008	2
`(gdb) p sizeof(ptr)`	24	8
`(gdb) p ptr = ptr + 1`	1	0x7ffc67181974

Discussion/Explanation Prompts:

Steps 1 & 2 – stack arrays vs. pointers
based on your exploration with gdb, describe the ways in which `arr` and `ptr` are identical? How are `arr` and `ptr` different? A stack array and a pointer to it are almost interchangeable. What do your results say about the overall nature of stack arrays vs. pointers to them?

Both point to the same memory address and can access the elements of the array using square brackets.

Arr is fixed and cannot be reassigned or changed while ptr can be reassigned to point to a new memory address.
The sizes of both are different since arr represents the entire array while ptr is just a pointer to the first int.

The nature of the two is that arr has a fixed size and location while ptr can be changed.

CS 211
Lab04 - Results Document

Step 3 - array parameter passing

The two parameters to goofy( ) behave identically, despite one being explicitly defined as an array, while the other is a pointer. What happens in parameter passing to make this so? Draw a picture of the state of memory to shed light on the matter.

CS 211
Lab04 - Results Document

Original Arr

0	10	20	30	40	50
0	1	2	3	4	5

Inside goofy ()

a [2] = 20
b [2] = 20
original

a [2] = 140
b [2] = 101
modified

After goofy ()

0	10	101	30	40	50
0	1	2	3	4	5

CS 211
Lab04 - Results Document

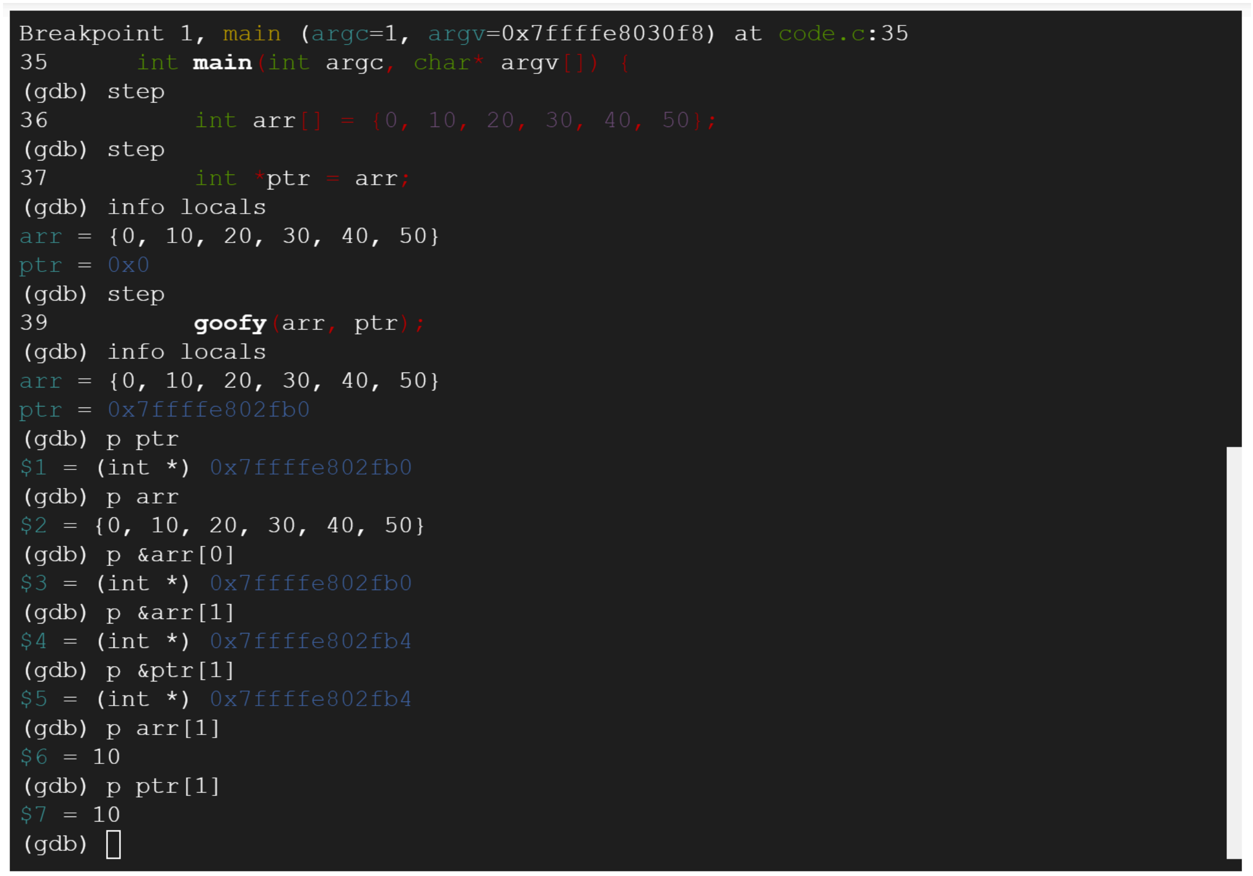
Step 3 – persistent changes to function parameters
Based on your exploration with gdb, which methods of array parameter passing and assignments within function have persistent effects and which do not? Explain why.

Persistent - a[2] *= 7;
When array values are change within a function, that change modifies the original memory

Non persistent - *p_str = “Chicago”;
Reassigns the pointer to a new string which doesn’t carry over when the function ends because the pointers are passed by value

CS 211
Lab04 - Results Document

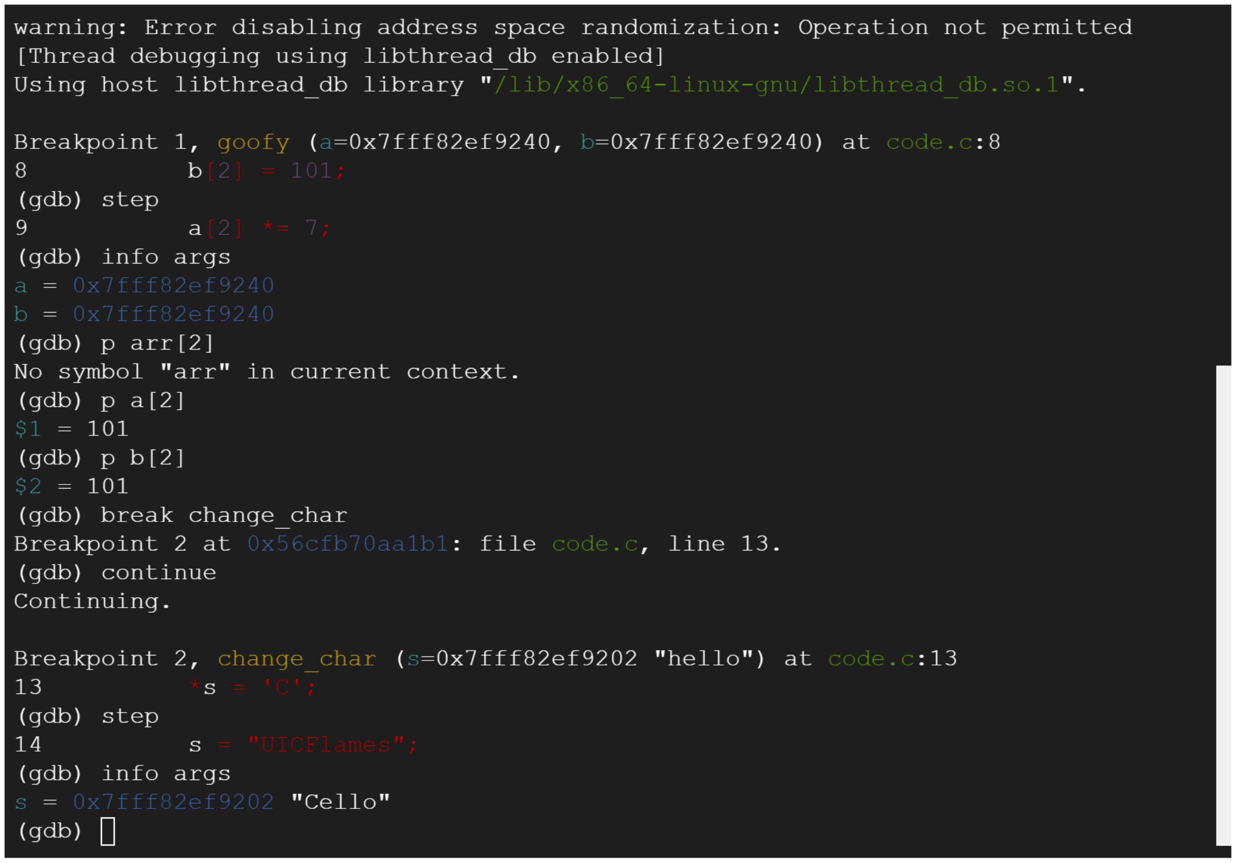
Steps 1 & 2 - record of code compilation, program execution, a few gdb commands, & outputs:



```
Breakpoint 1, main (argc=1, argv=0x7ffffe8030f8) at code.c:35
35      int main(int argc, char* argv[]) {
(gdb) step
36              int arr[] = {0, 10, 20, 30, 40, 50};
(gdb) step
37              int *ptr = arr;
(gdb) info locals
arr = {0, 10, 20, 30, 40, 50}
ptr = 0x0
(gdb) step
39              goofy(arr, ptr);
(gdb) info locals
arr = {0, 10, 20, 30, 40, 50}
ptr = 0x7ffffe802fb0
(gdb) p ptr
$1 = (int *) 0x7ffffe802fb0
(gdb) p arr
$2 = {0, 10, 20, 30, 40, 50}
(gdb) p &arr[0]
$3 = (int *) 0x7ffffe802fb0
(gdb) p &arr[1]
$4 = (int *) 0x7ffffe802fb4
(gdb) p &ptr[1]
$5 = (int *) 0x7ffffe802fb4
(gdb) p arr[1]
$6 = 10
(gdb) p ptr[1]
$7 = 10
(gdb)
```

CS 211
Lab04 - Results Document

Step 3 - a few key gdb commands & outputs:



```
warning: Error disabling address space randomization: Operation not permitted
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Breakpoint 1, goofy (a=0x7fff82ef9240, b=0x7fff82ef9240) at code.c:8
8             b[2] = 101;
(gdb) step
9             a[2] *= 7;
(gdb) info args
a = 0x7fff82ef9240
b = 0x7fff82ef9240
(gdb) p arr[2]
No symbol "arr" in current context.
(gdb) p a[2]
$1 = 101
(gdb) p b[2]
$2 = 101
(gdb) break change_char
Breakpoint 2 at 0x56cfb70aa1b1: file code.c, line 13.
(gdb) continue
Continuing.

Breakpoint 2, change_char (s=0x7fff82ef9202 "hello") at code.c:13
13            *s = 'C';
(gdb) step
14            s = "UICFlames";
(gdb) info args
s = 0x7fff82ef9202 "Cello"
(gdb) 
```

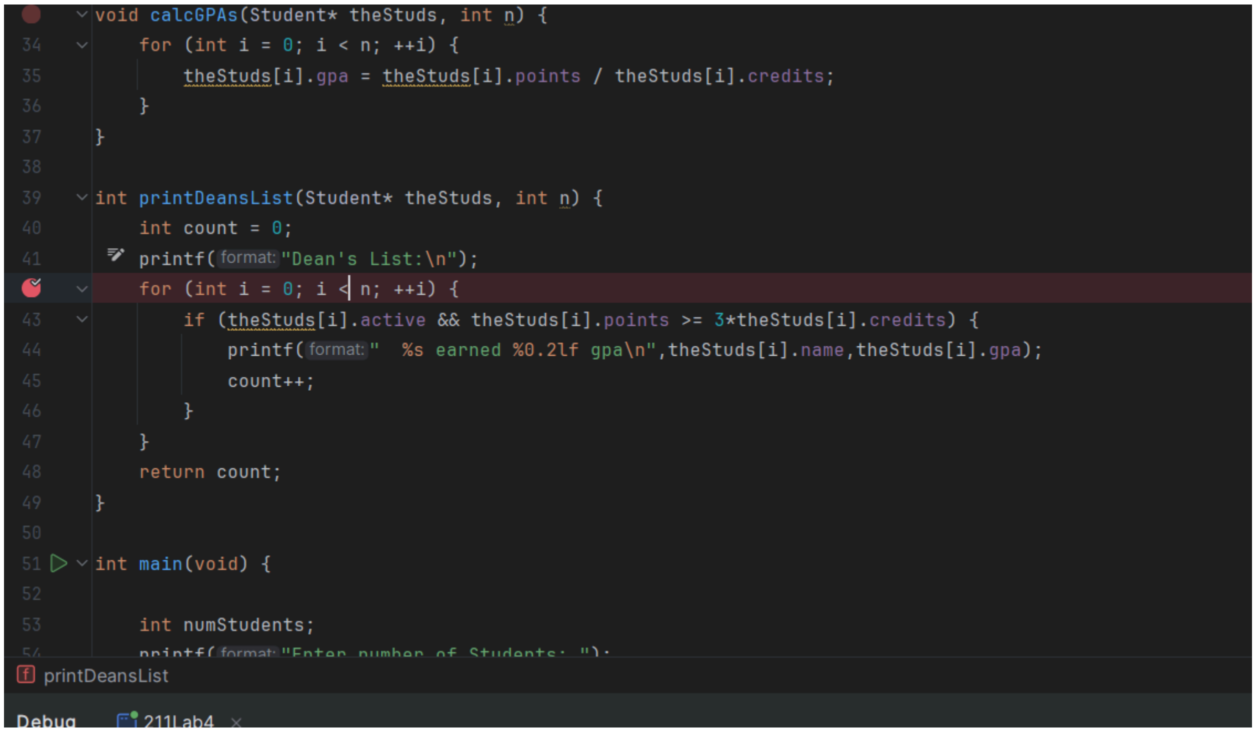
CS 211
Lab04 - Results Document

Lab04 - Part III - Find/Fix the Mistakes in a GUI Debugger

Identify each of the bugs you find and explain how you fixed them.

- *Bug #1: scanf("%d", numStudents); to scanf("%d", &numStudents);*
- *Bug #2: for (int i = 0; i < numStudents; i) to for (int i = 0; i < numStudents; i++)*
- *Bug #3: aStud = (Student*)malloc(sizeof(Student)); - remove since already alloc. In main*
- *Bug #4: scanf("%d", (aStud->points)); to scanf("%d", &(aStud->points));*
- *Bug #5: theStuds[i].gpa = theStuds[i].points / theStuds[i].credits; to (double)theStuds[i].points / theStuds[i].credits; for correct floating point*
- *Bug #6: if (theStuds[i].active && theStuds[i].points >= 3 * theStuds[i].credits) to if (theStuds[i].active && theStuds[i].gpa >= 3.0) for proper point*
- *Bug #7: for (int i = 0; i <= n; ++i) to for (int i = 0; i < n; ++i)*
- *Bug #8: free(studs); was missing*

Include one screenshot of your GUI debugger showing the process of finding/fixing one of the bugs.



```
void calcGPAs(Student* theStuds, int n) {
    for (int i = 0; i < n; ++i) {
        theStuds[i].gpa = theStuds[i].points / theStuds[i].credits;
    }
}

int printDeansList(Student* theStuds, int n) {
    int count = 0;
    printf(format: "Dean's List:\n");
    for (int i = 0; i < n; ++i) {
        if (theStuds[i].active && theStuds[i].points >= 3*theStuds[i].credits) {
            printf(format: "  %s earned %0.2lf gpa\n",theStuds[i].name,theStuds[i].gpa);
            count++;
        }
    }
    return count;
}

int main(void) {

    int numStudents;
    printf(format: "Enter number of Students: ");
```

printDeansList

Debug   211Lab4