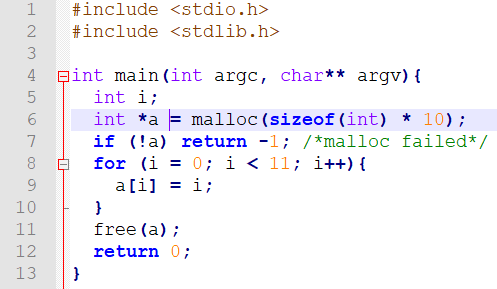
## Case 1



==25013== Memcheck, a memory error detector

==25013== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.

==25013== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info

==25013== Command: ./case1

==25013==

==25013== Invalid write of size 4

==25013== at 0x1086D5: main (case1.c:9)

==25013== Address 0x522d068 is 0 bytes after a block of size 40 alloc'd

==25013== at 0x4C2FB0F: malloc (in /usr/lib/valgrind/vgpreload\_memcheck-amd64-linux.so)

==25013== by 0x1086A2: main (case1.c:6)

==25013==

==25013==

==25013== HEAP SUMMARY:

==25013== in use at exit: 0 bytes in 0 blocks

==25013== total heap usage: 1 allocs, 1 frees, 40 bytes allocated

==25013==

==25013== All heap blocks were freed -- no leaks are possible

==25013==

==25013== For counts of detected and suppressed errors, rerun with: -v

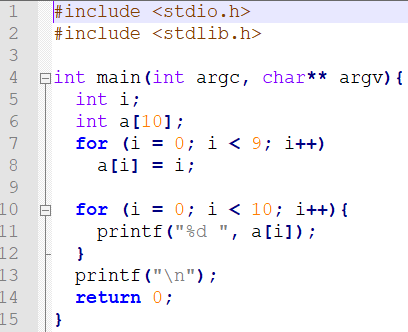
==25013== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)

### Analysis

This message says that the program did an illegal 4-byte read of address 0x522d068, which, as far as Memcheck can tell, is not a valid stack address, nor corresponds to any current heap blocks or recently freed heap blocks. The read is happening at line 9 of **case1.c**, called from line 6 of the same file. This is so because only 10 integer size memory was allocated when 11 integers were written, exceeding the reserved area for that variable type pointer.

Errors associated with an identified (current or freed) heap block like reading freed memory, Valgrind reports the location where the error happened, and also where the associated heap block was allocated/freed in the heap summary.

## Case 2



==25045== Memcheck, a memory error detector

==25045== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.

==25045== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info

==25045== Command: ./case2

==25045==

==25045== Conditional jump or move depends on uninitialised value(s)

==25045== at 0x4E9896A: vfprintf (vfprintf.c:1642)

==25045== by 0x4EA0FA5: printf (printf.c:33)

==25045== by 0x10875B: main (case2.c:11)

==25045==

==25045== Use of uninitialised value of size 8

==25045== at 0x4E948FB: \_itoa\_word (\_itoa.c:179)

==25045== by 0x4E97F9D: vfprintf (vfprintf.c:1642)

==25045== by 0x4EA0FA5: printf (printf.c:33)

==25045== by 0x10875B: main (case2.c:11)

==25045==

==25045== Conditional jump or move depends on uninitialised value(s)

==25045== at 0x4E94905: \_itoa\_word (\_itoa.c:179)

==25045== by 0x4E97F9D: vfprintf (vfprintf.c:1642)

==25045== by 0x4EA0FA5: printf (printf.c:33)

==25045== by 0x10875B: main (case2.c:11)

==25045==

==25045== Conditional jump or move depends on uninitialised value(s)

==25045== at 0x4E980A4: vfprintf (vfprintf.c:1642)

==25045== by 0x4EA0FA5: printf (printf.c:33)

==25045== by 0x10875B: main (case2.c:11)

==25045==

==25045== Conditional jump or move depends on uninitialised value(s)

==25045== at 0x4E98BDC: vfprintf (vfprintf.c:1642)

==25045== by 0x4EA0FA5: printf (printf.c:33)

==25045== by 0x10875B: main (case2.c:11)

==25045==

0 1 2 3 4 5 6 7 8 31

==25045==

==25045== HEAP SUMMARY:

==25045== in use at exit: 0 bytes in 0 blocks

==25045== total heap usage: 1 allocs, 1 frees, 1,024 bytes allocated

==25045==

==25045== All heap blocks were freed -- no leaks are possible

==25045==

==25045== For counts of detected and suppressed errors, rerun with: -v

==25045== Use --track-origins=yes to see where uninitialised values come from

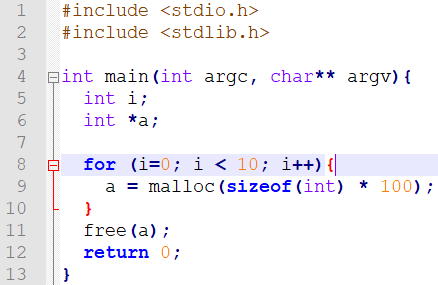
==25045== ERROR SUMMARY: 7 errors from 5 contexts (suppressed: 0 from 0)

### Analysis

The output format of Valgrind is “==25045==” where 25045 is the process ID. Therefore, the only output of the program itself is this line: 0 1 2 3 4 5 6 7 8 31

Out of that, the same root cause is flagging many errors. However, once we take a closer look to them, we see that everyone is connected to the single cause of **a[9]** array not being initialized prior to printf it on **line 11** of **case2.c** file. Memory checker just found different places inside the libraries called by printf where jump depends on value pointed by the array in the uninitialized position.

## Case 3



==24927== Memcheck, a memory error detector

==24927== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.

==24927== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info

==24927== Command: ./case3

==24927==

==24927==

==24927== HEAP SUMMARY:

==24927== in use at exit: 3,600 bytes in 9 blocks

==24927== total heap usage: 10 allocs, 1 frees, 4,000 bytes allocated

==24927==

==24927== LEAK SUMMARY:

==24927== definitely lost: 3,600 bytes in 9 blocks

==24927== indirectly lost: 0 bytes in 0 blocks

==24927== possibly lost: 0 bytes in 0 blocks

==24927== still reachable: 0 bytes in 0 blocks

==24927== suppressed: 0 bytes in 0 blocks

==24927== Rerun with --leak-check=full to see details of leaked memory

==24927==

==24927== For counts of detected and suppressed errors, rerun with: -v

==24927== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)

### Analysis

Valgrind states that 10 memory allocation was performed, all of them, successfully. In this case, we have no errors on memory usage. However, since those 10 allocations are being assigned to the same pointer, only last memory block was freed in **line 11** of **case3.c** file. Valgrind, properly advised that we have a LEAK issue on those 9 memory blocks, representing in total 3600 bytes never freed.