#uni/semester3/Betriebssysteme/chapter14/code-fragen

1. First, write a simple program called null.c that creates a pointer to an integer, sets it to NULL, and then tries to dereference it. Compile this into an executable called null. What happens when you run this program?

```
vladb@VladB ~/G/h/S/B/B/c14_memory-api (master)> ./null
fish: './null' terminated by signal SIGSEGV (Address boundary error)
```

Mit bash kommt:
Segmentation fault: 11

```
vl161bra@ct-bsys-ws20-12:~/htwg/S3/BS/BSCode/c14_memory-api$ ./null
Segmentation fault
```

2. Next, compile this program with symbol information included (with the -g flag). Doing so let's put more information into the executable, enabling the debugger to access more useful information about variable names and the like. Run the program under the debugger by typing gdb null and then, once gdb is running, typing run. What does gdb show you?

```
vl161bra@ct-bsys-ws20-12:~/htwg/S3/BS/BSCode/c14_memory-api$ gcc -o -g null null.c
vl161bra@ct-bsys-ws20-12:~/htwg/S3/BS/BSCode/c14_memory-api$ gdb null
GNU gdb (Debian 8.2.1-2+b3) 8.2.1
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from null...(no debugging symbols found)...done.
(gdb)
(gdb) run
Starting program: /home/vl161bra/htwg/S3/BS/BSCode/c14_memory-api/null

Program received signal SIGSEGV, Segmentation fault.
0x0000555555555150 in main ()
(gdb)
```

3. Finally, use the valgrind tool on this program. We'll use the memcheck tool that is a part of valgrind to analyze what happens. Run this by typing in the following: valgrind --leak-check=yes null. What happens when you run this? Can you interpret the output from the tool?

```
vl161bra@ct-bsys-ws20-12:~/htwg/S3/BS/BSCode/c14_memory-api$ valgrind --leak-check=yes ./null
==32427== Memcheck, a memory error detector
==32427== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==32427== Using Valgrind-3.14.0 and LibVEX; rerun with -h for copyright info
==32427== Command: ./null
==32427==
==32427== Invalid read of size 4
==32427==    at 0x109150: main (in /home/vl161bra/htwg/S3/BS/BSCode/c14_memory-api/null)
==32427==  Address 0x0 is not stack'd, malloc'd or (recently) free'd
==32427==
==32427==
==32427== Process terminating with default action of signal 11 (SIGSEGV)
==32427==  Access not within mapped region at address 0x0
==32427==    at 0x109150: main (in /home/vl161bra/htwg/S3/BS/BSCode/c14_memory-api/null)
==32427==  If you believe this happened as a result of a stack
==32427==  overflow in your program's main thread (unlikely but
==32427==  possible), you can try to increase the size of the
==32427==  main thread stack using the --main-stacksize= flag.
==32427==  The main thread stack size used in this run was 8388608.
==32427==
==32427== HEAP SUMMARY:
==32427==     in use at exit: 0 bytes in 0 blocks
==32427==   total heap usage: 0 allocs, 0 frees, 0 bytes allocated
==32427==
==32427== All heap blocks were freed -- no leaks are possible
==32427==
==32427== For counts of detected and suppressed errors, rerun with: -v
==32427== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
Segmentation fault
```

➜ Wir versuchen auf was zuzugreifen, was noch nicht allokiert wurde

➜ wir versuchen auf etwas zuzugreifen, was uns nicht gehört (segmentation fault)

4. Write a simple program that allocates memory using malloc() but forgets to free it before exiting. What happens when this program runs? Can you use gdb to find any problems with it? How about valgrind (again with the --leak-check=yes flag)?

```
10vl161bra@ct-bsys-ws20-12:~/htwg/S3/BS/BSCode/c14_memory-api$ gdb malloc
GNU gdb (Debian 8.2.1-2+b3) 8.2.1
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from malloc...done.
(gdb) run
Starting program: /home/vl161bra/htwg/S3/BS/BSCode/c14_memory-api/malloc
10[Inferior 1 (process 5128) exited normally]
```

➜ Mit gdb kein Fehler gefunden

```
vl161bra@ct-bsys-ws20-12:~/htwg/S3/BS/BSCode/c14_memory-api$ valgrind --leak-check=yes ./malloc
==5614== Memcheck, a memory error detector
==5614== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==5614== Using Valgrind-3.14.0 and LibVEX; rerun with -h for copyright info
==5614== Command: ./malloc
==5614==
10==5614==
==5614== HEAP SUMMARY:
==5614==     in use at exit: 4 bytes in 1 blocks
==5614==   total heap usage: 2 allocs, 1 frees, 1,028 bytes allocated
==5614==
==5614== 4 bytes in 1 blocks are definitely lost in loss record 1 of 1
==5614==    at 0x483577F: malloc (vg_replace_malloc.c:299)
==5614==    by 0x10915D: main (malloc.c:8)
==5614==
==5614== LEAK SUMMARY:
==5614==    definitely lost: 4 bytes in 1 blocks
==5614==    indirectly lost: 0 bytes in 0 blocks
==5614==      possibly lost: 0 bytes in 0 blocks
==5614==    still reachable: 0 bytes in 0 blocks
==5614==         suppressed: 0 bytes in 0 blocks
==5614==
==5614== For counts of detected and suppressed errors, rerun with: -v
==5614== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

➜ 1 error, nämlich dass 4 Bytes nicht freigegeben wurden.

5. Write a program that creates an array of integers called data of size 100 using malloc; then, set data[100] to zero. What happens when you run this program? What happens when you run this program using valgrind? Is the program correct?

```
vl161bra@ct-bsys-ws20-12:~/htwg/S3/BS/BSCode/c14_memory-api$ valgrind --leak-check=yes ./array
==7043== Memcheck, a memory error detector
==7043== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==7043== Using Valgrind-3.14.0 and LibVEX; rerun with -h for copyright info
==7043== Command: ./array
==7043==
==7043== Invalid write of size 4
==7043==    at 0x10915C: main (in /home/vl161bra/htwg/S3/BS/BSCode/c14_memory-api/array)
==7043==  Address 0x4a181d0 is 0 bytes after a block of size 400 alloc'd
==7043==    at 0x483577F: malloc (vg_replace_malloc.c:299)
==7043==    by 0x10914D: main (in /home/vl161bra/htwg/S3/BS/BSCode/c14_memory-api/array)
==7043==
==7043==
==7043== HEAP SUMMARY:
==7043==     in use at exit: 400 bytes in 1 blocks
==7043==   total heap usage: 1 allocs, 0 frees, 400 bytes allocated
==7043==
==7043== 400 bytes in 1 blocks are definitely lost in loss record 1 of 1
==7043==    at 0x483577F: malloc (vg_replace_malloc.c:299)
==7043==    by 0x10914D: main (in /home/vl161bra/htwg/S3/BS/BSCode/c14_memory-api/array)
==7043==
==7043== LEAK SUMMARY:
==7043==    definitely lost: 400 bytes in 1 blocks
==7043==    indirectly lost: 0 bytes in 0 blocks
==7043==      possibly lost: 0 bytes in 0 blocks
==7043==    still reachable: 0 bytes in 0 blocks
==7043==         suppressed: 0 bytes in 0 blocks
==7043==
==7043== For counts of detected and suppressed errors, rerun with: -v
==7043== ERROR SUMMARY: 2 errors from 2 contexts (suppressed: 0 from 0)
```

➜ Ein error kommt, weil das ganze Array nicht freigegeben wurde

➜ Das andere kommt, weil eigentlich mit array[100] möchten wir den int mit offset +101 zugreifen, welcher noch nicht allokiert wurde. Nur data[0] bis data[99] allokiert

6. Create a program that allocates an array of integers (as above), frees them, and then tries to print the value of one of the elements of the array. Does the program run? What happens when you use valgrind on it?

   ➜ Das Programm wird ausgeführt und printet 0 aus

   ➜ Ein Error kommt weil wir versuchen auf nicht mehr allokierter Speicher zuzugreifen

```
0vl161bra@ct-bsys-ws20-12:~/htwg/S3/BS/BSCode/c14_memory-api$ valgrind --leak-check=yes ./free_array
==8560== Memcheck, a memory error detector
==8560== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==8560== Using Valgrind-3.14.0 and LibVEX; rerun with -h for copyright info
==8560== Command: ./free_array
==8560==
==8560== Invalid read of size 4
==8560==    at 0x109188: main (in /home/vl161bra/htwg/S3/BS/BSCode/c14_memory-api/free_array)
==8560==  Address 0x4a18108 is 200 bytes inside a block of size 400 free'd
==8560==    at 0x48369AB: free (vg_replace_malloc.c:530)
==8560==    by 0x10917D: main (in /home/vl161bra/htwg/S3/BS/BSCode/c14_memory-api/free_array)
==8560==  Block was alloc'd at
==8560==    at 0x483577F: malloc (vg_replace_malloc.c:299)
==8560==    by 0x10916D: main (in /home/vl161bra/htwg/S3/BS/BSCode/c14_memory-api/free_array)
==8560==
0==8560==
==8560== HEAP SUMMARY:
==8560==     in use at exit: 0 bytes in 0 blocks
==8560==   total heap usage: 2 allocs, 2 frees, 1,424 bytes allocated
==8560==
==8560== All heap blocks were freed -- no leaks are possible
==8560==
==8560== For counts of detected and suppressed errors, rerun with: -v
==8560== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

7. Now pass a funny value to free (e.g., a pointer in the middle of the array you allocated above). What happens? Do you need tools to find this type of problem?

```
vl161bra@ct-bsys-ws20-12:~/htwg/S3/BS/BSCode/c14_memory-api$ ./frei_funny_array
free(): invalid pointer
Aborted
```

```
vl161bra@ct-bsys-ws20-12:~/htwg/S3/BS/BSCode/c14_memory-api$ valgrind --leak-check=yes ./frei_funny_a
rray
==10459== Memcheck, a memory error detector
==10459== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==10459== Using Valgrind-3.14.0 and LibVEX; rerun with -h for copyright info
==10459== Command: ./frei_funny_array
==10459==
==10459== Invalid free() / delete / delete[] / realloc()
==10459==    at 0x48369AB: free (vg_replace_malloc.c:530)
==10459==    by 0x109173: main (in /home/vl161bra/htwg/S3/BS/BSCode/c14_memory-api/frei_funny_array)
==10459==  Address 0x4a18108 is 200 bytes inside a block of size 400 alloc'd
==10459==    at 0x483577F: malloc (vg_replace_malloc.c:299)
==10459==    by 0x10915D: main (in /home/vl161bra/htwg/S3/BS/BSCode/c14_memory-api/frei_funny_array)
==10459==
==10459==
==10459== HEAP SUMMARY:
==10459==     in use at exit: 400 bytes in 1 blocks
==10459==   total heap usage: 1 allocs, 1 frees, 400 bytes allocated
==10459==
==10459== 400 bytes in 1 blocks are definitely lost in loss record 1 of 1
==10459==    at 0x483577F: malloc (vg_replace_malloc.c:299)
==10459==    by 0x10915D: main (in /home/vl161bra/htwg/S3/BS/BSCode/c14_memory-api/frei_funny_array)
==10459==
==10459== LEAK SUMMARY:
==10459==    definitely lost: 400 bytes in 1 blocks
==10459==    indirectly lost: 0 bytes in 0 blocks
==10459==      possibly lost: 0 bytes in 0 blocks
==10459==    still reachable: 0 bytes in 0 blocks
==10459==         suppressed: 0 bytes in 0 blocks
==10459==
==10459== For counts of detected and suppressed errors, rerun with: -v
==10459== ERROR SUMMARY: 2 errors from 2 contexts (suppressed: 0 from 0)
```

```
==18176== Invalid free() / delete / delete[] / realloc()
==18176==    at 0x48369AB: free (vg_replace_malloc.c:530)
==18176==    by 0x109173: main (in /home/vl161bra/htwg/S3/BS/BSCode/c14_memory-api/frei_funny_array)
==18176==  Address 0x4a18108 is 200 bytes inside a block of size 400 alloc'd
==18176==    at 0x483577F: malloc (vg_replace_malloc.c:299)
==18176==    by 0x10915D: main (in /home/vl161bra/htwg/S3/BS/BSCode/c14_memory-api/frei_funny_array)
==18176==
==18176==
==18176== HEAP SUMMARY:
==18176==     in use at exit: 400 bytes in 1 blocks
==18176==   total heap usage: 1 allocs, 1 frees, 400 bytes allocated
==18176==
==18176== Searching for pointers to 1 not-freed blocks
==18176== Checked 67,640 bytes
==18176==
==18176== 400 bytes in 1 blocks are definitely lost in loss record 1 of 1
==18176==    at 0x483577F: malloc (vg_replace_malloc.c:299)
==18176==    by 0x10915D: main (in /home/vl161bra/htwg/S3/BS/BSCode/c14_memory-api/frei_funny_array)
==18176==
==18176== LEAK SUMMARY:
==18176==    definitely lost: 400 bytes in 1 blocks
==18176==    indirectly lost: 0 bytes in 0 blocks
==18176==      possibly lost: 0 bytes in 0 blocks
==18176==    still reachable: 0 bytes in 0 blocks
==18176==         suppressed: 0 bytes in 0 blocks
==18176==
==18176== ERROR SUMMARY: 2 errors from 2 contexts (suppressed: 0 from 0)
==18176==
==18176== 1 errors in context 1 of 2:
==18176== Invalid free() / delete / delete[] / realloc()
==18176==    at 0x48369AB: free (vg_replace_malloc.c:530)
==18176==    by 0x109173: main (in /home/vl161bra/htwg/S3/BS/BSCode/c14_memory-api/frei_funny_array)
==18176==  Address 0x4a18108 is 200 bytes inside a block of size 400 alloc'd
==18176==    at 0x483577F: malloc (vg_replace_malloc.c:299)
==18176==    by 0x10915D: main (in /home/vl161bra/htwg/S3/BS/BSCode/c14_memory-api/frei_funny_array)
==18176==
==18176== ERROR SUMMARY: 2 errors from 2 contexts (suppressed: 0 from 0)
```

➜ Invalid free() da mit free nur pointers verwenden dürfen die vom malloc zurückgeliefert wurden.

8. Try out some of the other interfaces to memory allocation. For example, create a simple vector-like data structure and related routines that use realloc() to manage the vector. Use

an array to store the vectors elements; when a user adds an entry to the vector, use realloc() to allocate more space for it. How well does such a vector perform? How does it compare to a linked list? Use valgrind to help you find bugs.
➜ Linked List sehr angenehm, da bei reallokieren muss man nur ein weiteres Element allokieren, während Liste muss man mehr allokieren, eventuell wenn Speicher nicht gefunden wurde und es umkopieren werden muss.
➜ Vectors sind gut für zufälligen Zugriff auf das Array und schlecht für einfügen oder Löschen am Anfang
➜ Verketteten Listen sind gut beim Insertion und deletion am Anfang und irgendwoanders. Beim Schluss nicht so, da man durch alles gehen muss.

data structures - Linked List vs Vector - Stack Overflow