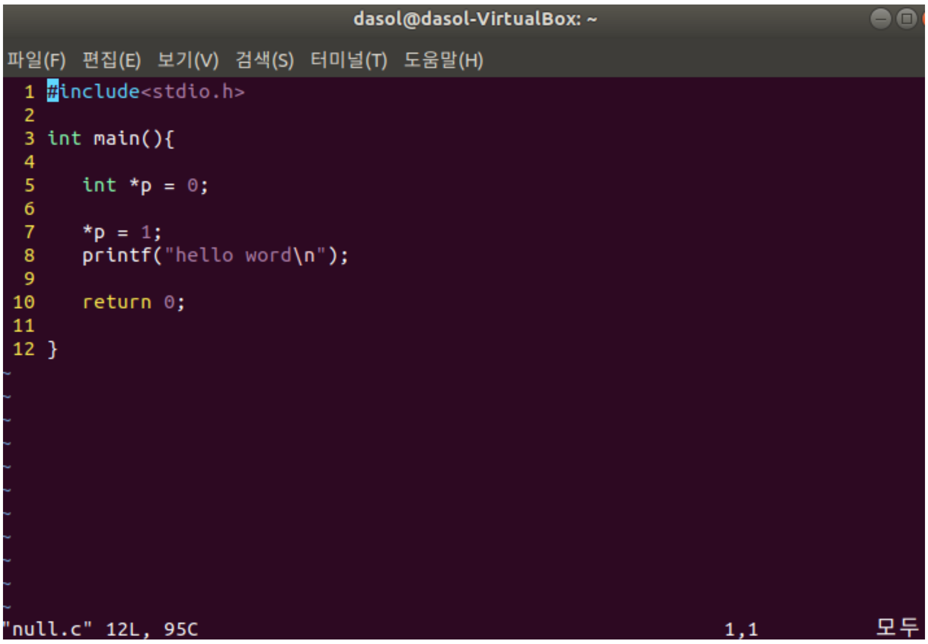
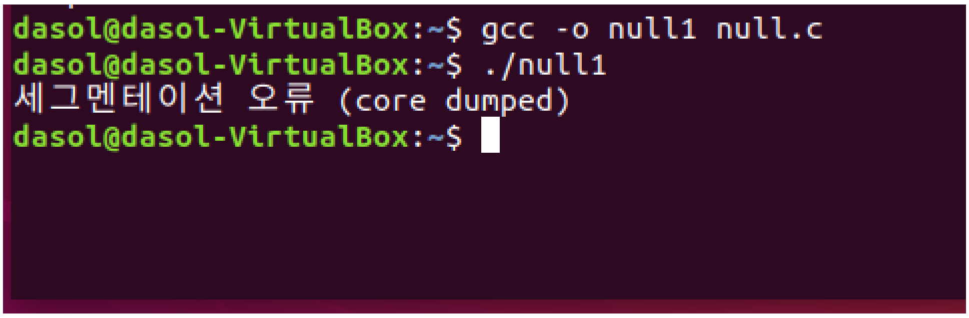
1. First, write a simple program called null.c that creates a pointer to an integer, sets it to NULL, and then tries to dereference it. Compile this into an executable called null. What happens when you run this program?


dasol@dasol-VirtualBox: ~
파일(F) 편집(E) 보기(V) 검색(S) 터미널(T) 도움말(H)

```
#include<stdio.h>

int main(){

    int *p = 0;

    *p = 1;
    printf("hello word\n");

    return 0;

}
~
~
~
~
~
~
~
~
~
~
~
"null.c" 12L, 95C                                  1,1           모두
```

This is my code for Q1. I made simple program called null.c that creates a pointer to an integer, sets it to NULL, and then tried to dereference it

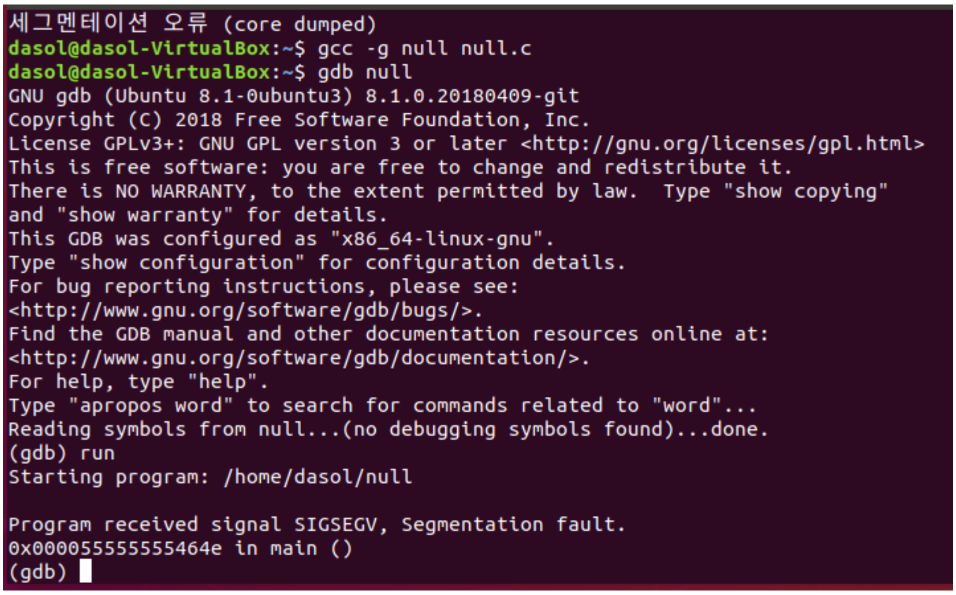


```
dasol@dasol-VirtualBox:~$ gcc -o null1 null.c
dasol@dasol-VirtualBox:~$ ./null1
세그멘테이션 오류 (core dumped)
dasol@dasol-VirtualBox:~$
```

This is what happened when compile this into an executable called null. It has segmentation fault.

Because my program's pointer dereference null. It calls 'Null Pointer Dereference'. The operating system uses the null as the first page address of memory (0x00000000). Therefore, if you access it, the program usually terminates abnormally.

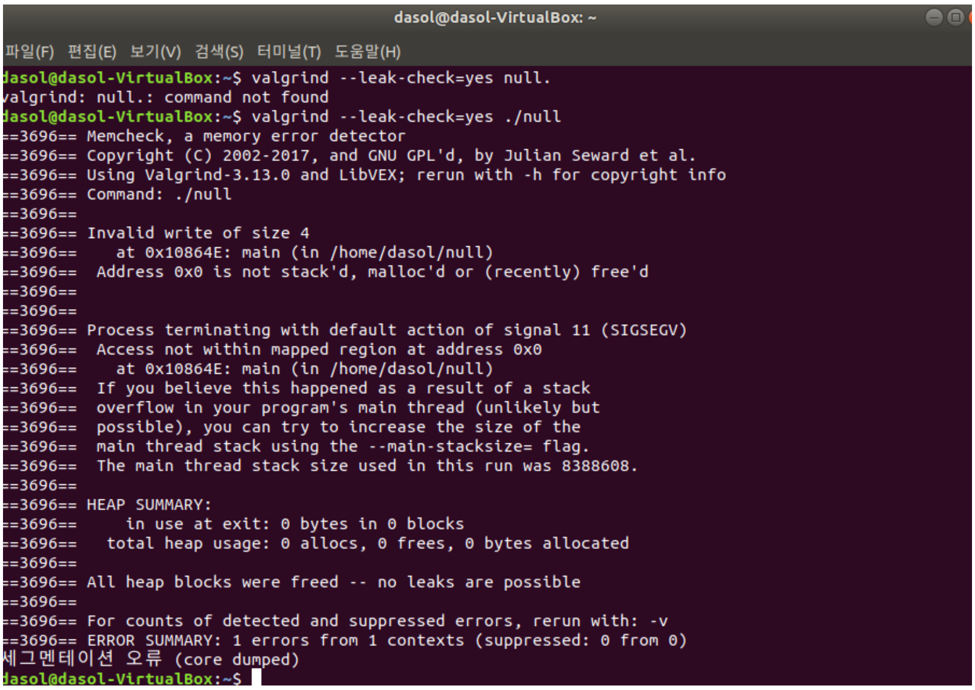
2. Next, compile this program with symbol information included (with the -g flag). Doing so let's put more information into the executable, enabling the debugger to access more useful information about variable names and the like. Run the program under the debugger by typing gdb null and then, once gdb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



```
세그멘테이션 오류 (core dumped)
dasol@dasol-VirtualBox:~$ gcc -g null null.c
dasol@dasol-VirtualBox:~$ gdb null
GNU gdb (Ubuntu 8.1-0ubuntu3) 8.1.0.20180409-git
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) run
Starting program: /home/dasol/null

Program received signal SIGSEGV, Segmentation fault.
0x000055555555464e in main ()
(gdb) 
```

I compiled null.c by using gdb. It told me main function's address and segmentation fault.

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?

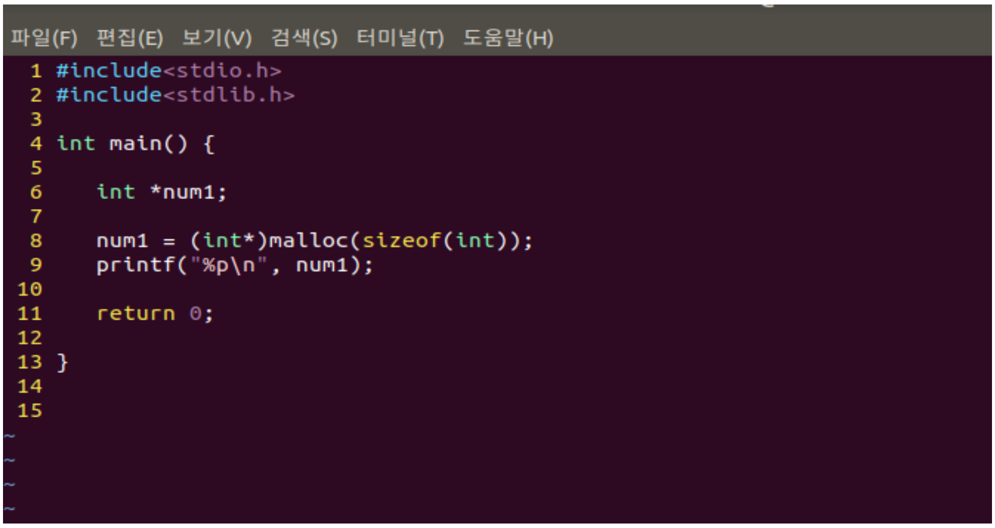


dasol@dasol-VirtualBox: ~
파일(F) 편집(E) 보기(V) 검색(S) 터미널(T) 도움말(H)

```
dasol@dasol-VirtualBox:~$ valgrind --leak-check=yes null.
valgrind: null.: command not found
dasol@dasol-VirtualBox:~$ valgrind --leak-check=yes ./null
==3696== Memcheck, a memory error detector
==3696== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==3696== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==3696== Command: ./null
==3696== 
==3696== Invalid write of size 4
==3696==    at 0x10864E: main (in /home/dasol/null)
==3696==  Address 0x0 is not stack'd, malloc'd or (recently) free'd
==3696== 
==3696== 
==3696== Process terminating with default action of signal 11 (SIGSEGV)
==3696==  Access not within mapped region at address 0x0
==3696==    at 0x10864E: main (in /home/dasol/null)
==3696==  If you believe this happened as a result of a stack
==3696==  overflow in your program's main thread (unlikely but
==3696==  possible), you can try to increase the size of the
==3696==  main thread stack using the --main-stacksize= flag.
==3696==  The main thread stack size used in this run was 8388608.
==3696== 
==3696== HEAP SUMMARY:
==3696==     in use at exit: 0 bytes in 0 blocks
==3696==   total heap usage: 0 allocs, 0 frees, 0 bytes allocated
==3696== 
==3696== All heap blocks were freed -- no leaks are possible
==3696== 
==3696== For counts of detected and suppressed errors, rerun with: -v
==3696== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
세그멘테이션 오류 (core dumped)
dasol@dasol-VirtualBox:~$ 
```

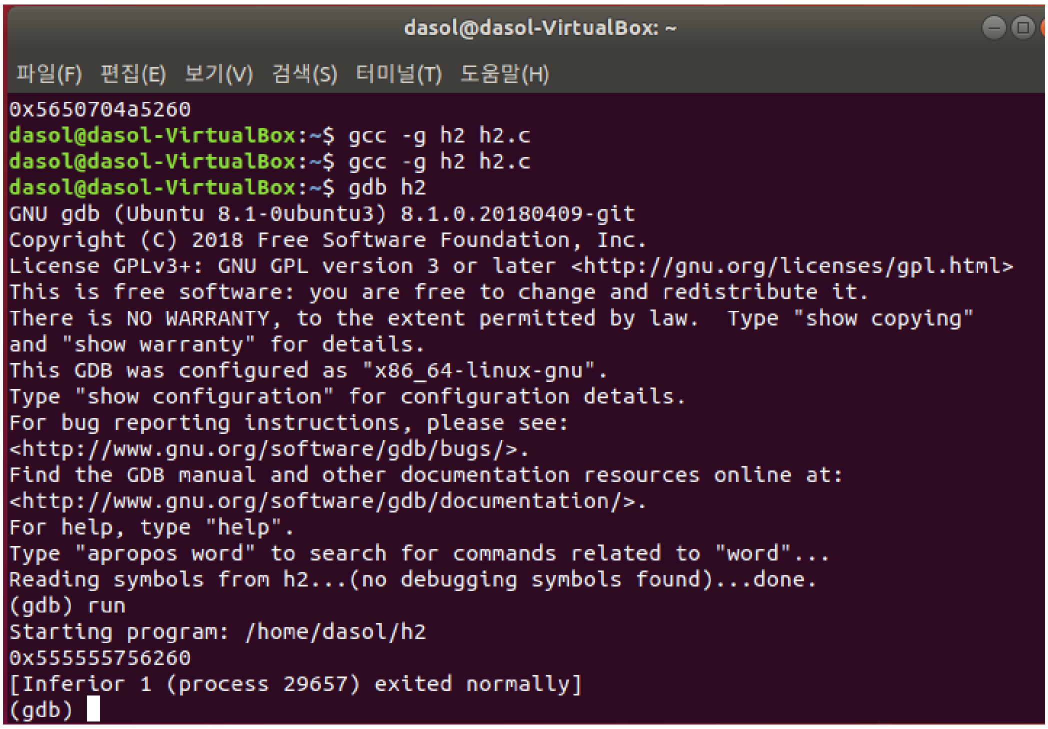
I used the valgrind to find problem. It told me the problem is segmentation fault. It means my program tried to access unallowable memory area. Access not within mapped region at address 0x0.

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)?



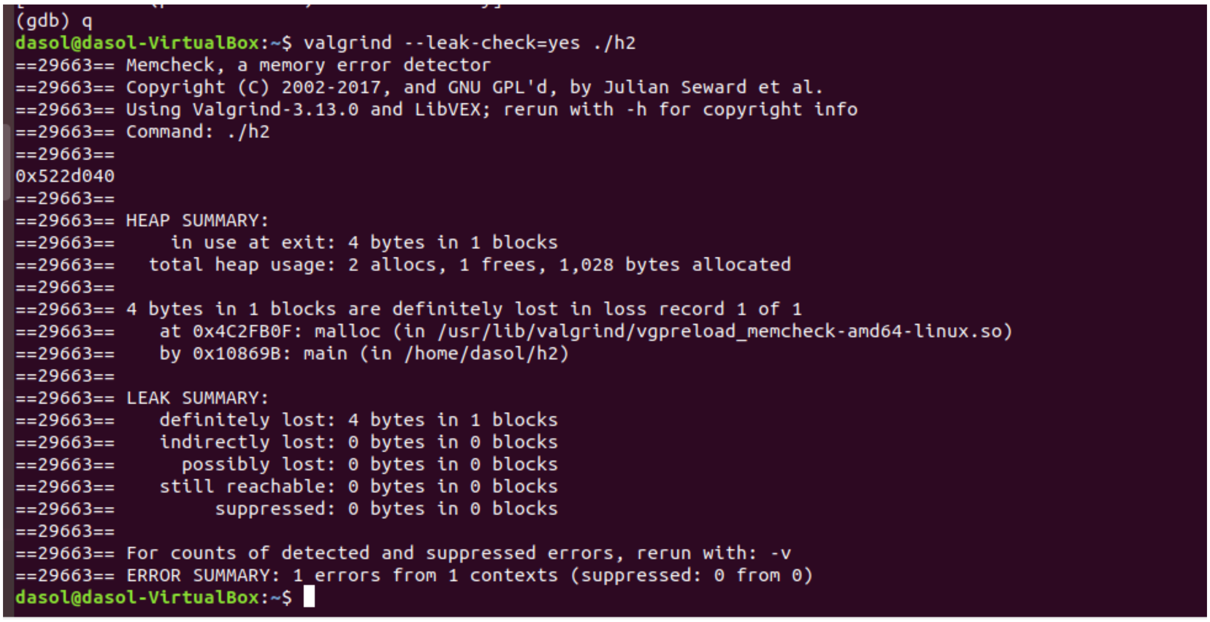
```
#include<stdio.h>
#include<stdlib.h>

int main() {

    int *num1;

    num1 = (int*)malloc(sizeof(int));
    printf("%p\n", num1);

    return 0;

}


~
~
~
~
```

This is my code for Q4. I made simple program called h2 that allocates memory using malloc() but forgets to free it before exiting.


dasol@dasol-VirtualBox: ~
파일(F) 편집(E) 보기(V) 검색(S) 터미널(T) 도움말(H)

```
0x5650704a5260
dasol@dasol-VirtualBox:~$ gcc -g h2 h2.c
dasol@dasol-VirtualBox:~$ gcc -g h2 h2.c
dasol@dasol-VirtualBox:~$ gdb h2
GNU gdb (Ubuntu 8.1-0ubuntu3) 8.1.0.20180409-git
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 h2...(no debugging symbols found)...done.
(gdb) run
Starting program: /home/dasol/h2
0x555555756260
[Inferior 1 (process 29657) exited normally]
(gdb) 
```

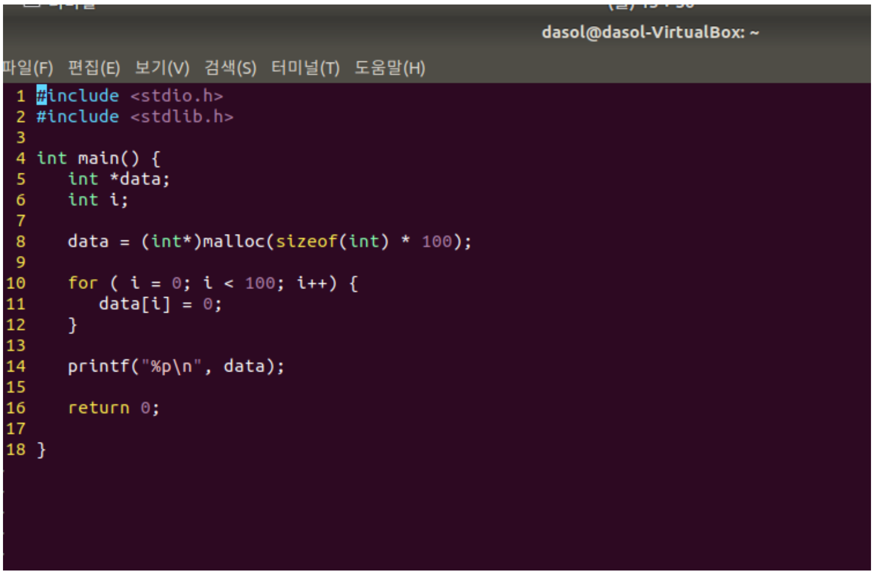
This is what happened when compile this into an executable called null by gdb. It has no problem.



```
(gdb) q
dasol@dasol-VirtualBox:~$ valgrind --leak-check=yes ./h2
==29663== Memcheck, a memory error detector
==29663== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==29663== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==29663== Command: ./h2
==29663==
0x522d040
==29663==
==29663== HEAP SUMMARY:
==29663==     in use at exit: 4 bytes in 1 blocks
==29663==   total heap usage: 2 allocs, 1 frees, 1,028 bytes allocated
==29663==
==29663== 4 bytes in 1 blocks are definitely lost in loss record 1 of 1
==29663==    at 0x4C2FB0F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==29663==    by 0x10869B: main (in /home/dasol/h2)
==29663==
==29663== LEAK SUMMARY:
==29663==    definitely lost: 4 bytes in 1 blocks
==29663==    indirectly lost: 0 bytes in 0 blocks
==29663==      possibly lost: 0 bytes in 0 blocks
==29663==    still reachable: 0 bytes in 0 blocks
==29663==         suppressed: 0 bytes in 0 blocks
==29663==
==29663== For counts of detected and suppressed errors, rerun with: -v
==29663== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
dasol@dasol-VirtualBox:~$ 
```

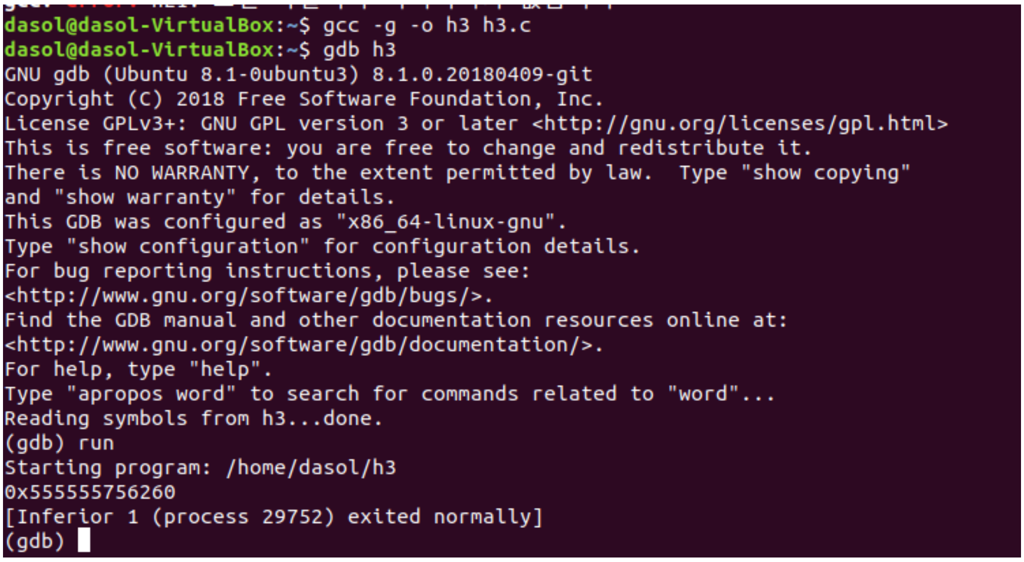
But when I compile it by using valgrind, it has a problem. It told me that my program has leak of memory at heap memory. Generally, memory that allocated by malloc() stored in heap memory. If my program forgets to free it before exiting, memory in heap memory didn't not delete unlike stack memory. Therefore, programmer has to free malloc() memory to prevent leak of memory.

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?


dasol@dasol-VirtualBox: ~
파일(F) 편집(E) 보기(V) 검색(S) 터미널(T) 도움말(H)

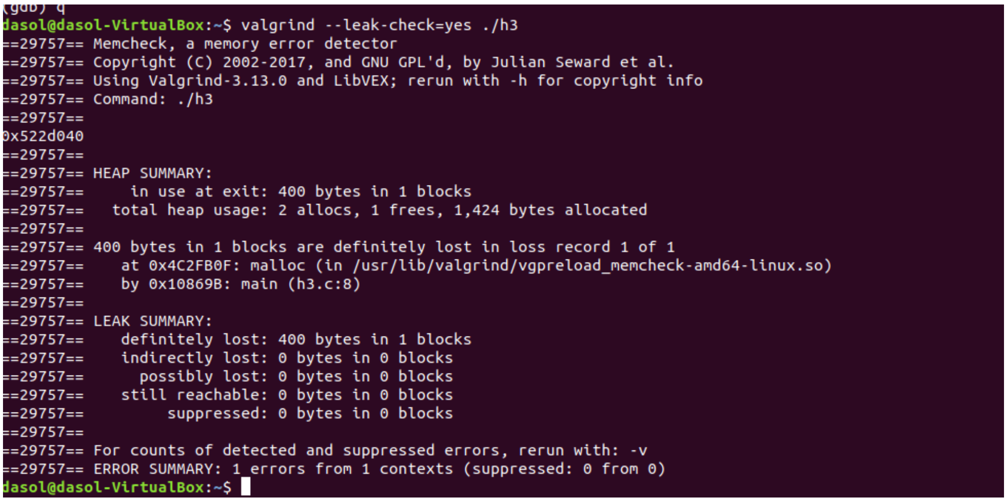
```
#include <stdio.h>
#include <stdlib.h>

int main() {
   int *data;
   int i;

   data = (int*)malloc(sizeof(int) * 100);

   for ( i = 0; i < 100; i++) {
      data[i] = 0;
   }

   printf("%p\n", data);

   return 0;

}
```

This is my code for Q5. I made simple program called h3 that created an array of integers called data of size 100 using malloc; then, set data[100] to zero.



```
dasol@dasol-VirtualBox:~$ gcc -g -o h3 h3.c
dasol@dasol-VirtualBox:~$ gdb h3
GNU gdb (Ubuntu 8.1-0ubuntu3) 8.1.0.20180409-git
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 h3...done.
(gdb) run
Starting program: /home/dasol/h3
0x555555756260
[Inferior 1 (process 29752) exited normally]
(gdb) 
```

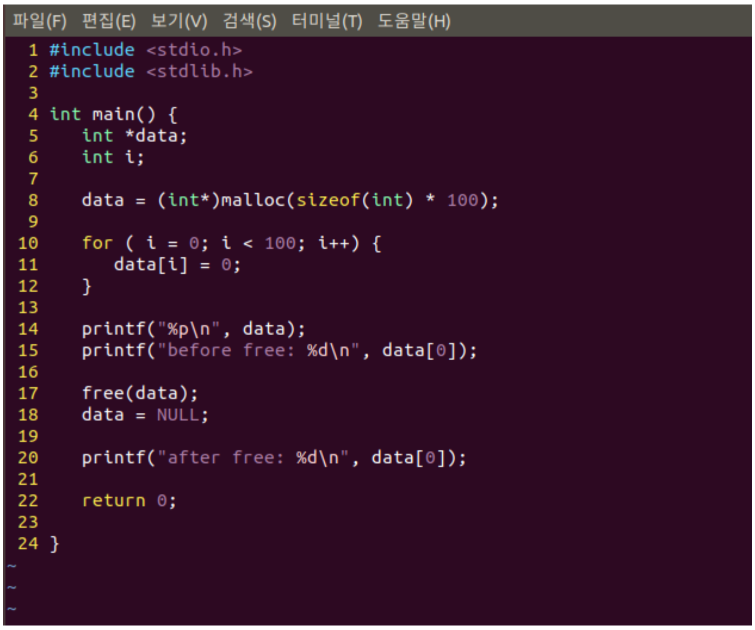
This is what happened when compile this code by gdb. It also has no problem.



```
(gdb) q
dasol@dasol-VirtualBox:~$ valgrind --leak-check=yes ./h3
==29757== Memcheck, a memory error detector
==29757== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==29757== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==29757== Command: ./h3
==29757==
0x522d040
==29757==
==29757== HEAP SUMMARY:
==29757==     in use at exit: 400 bytes in 1 blocks
==29757==   total heap usage: 2 allocs, 1 frees, 1,424 bytes allocated
==29757==
==29757== 400 bytes in 1 blocks are definitely lost in loss record 1 of 1
==29757==    at 0x4C2FB0F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==29757==    by 0x10869B: main (h3.c:8)
==29757==
==29757== LEAK SUMMARY:
==29757==    definitely lost: 400 bytes in 1 blocks
==29757==    indirectly lost: 0 bytes in 0 blocks
==29757==      possibly lost: 0 bytes in 0 blocks
==29757==    still reachable: 0 bytes in 0 blocks
==29757==         suppressed: 0 bytes in 0 blocks
==29757==
==29757== For counts of detected and suppressed errors, rerun with: -v
==29757== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
dasol@dasol-VirtualBox:~$
```

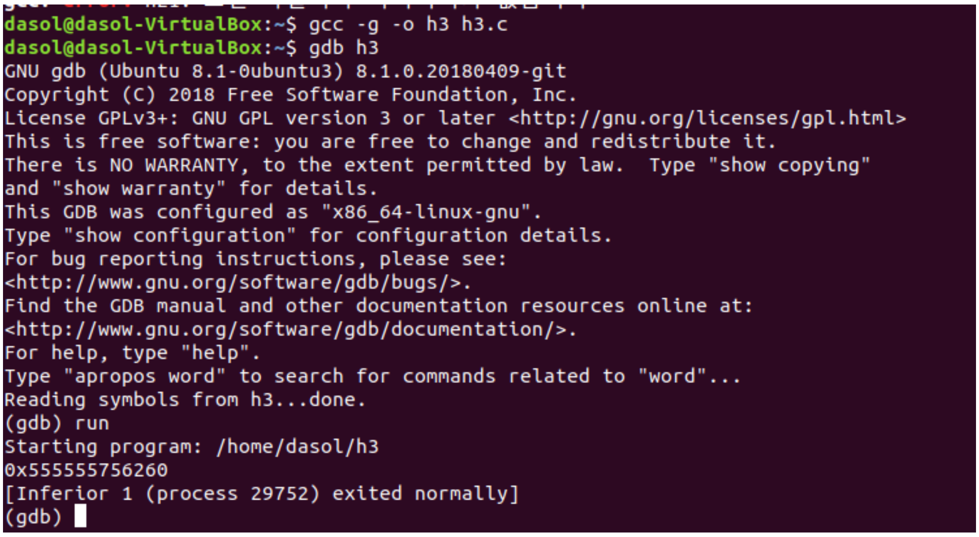
As similar with Q4, when I compile it by using valgrind, it has a problem. It told me that my program has leak of memory at heap memory. Generally, memory that allocated by malloc() stored in heap memory. If my program forgets to free it before exiting, memory in heap memory didn't not delete unlike stack memory. Therefore, programmer has to free malloc() memory to prevent leak of memory.

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?



```
#include <stdio.h>
#include <stdlib.h>

int main() {
    int *data;
    int i;

    data = (int*)malloc(sizeof(int) * 100);

    for ( i = 0; i < 100; i++) {
        data[i] = 0;
    }

    printf("%p\n", data);
    printf("before free: %d\n", data[0]);

    free(data);
    data = NULL;

    printf("after free: %d\n", data[0]);

    return 0;
}
~
~
~
```

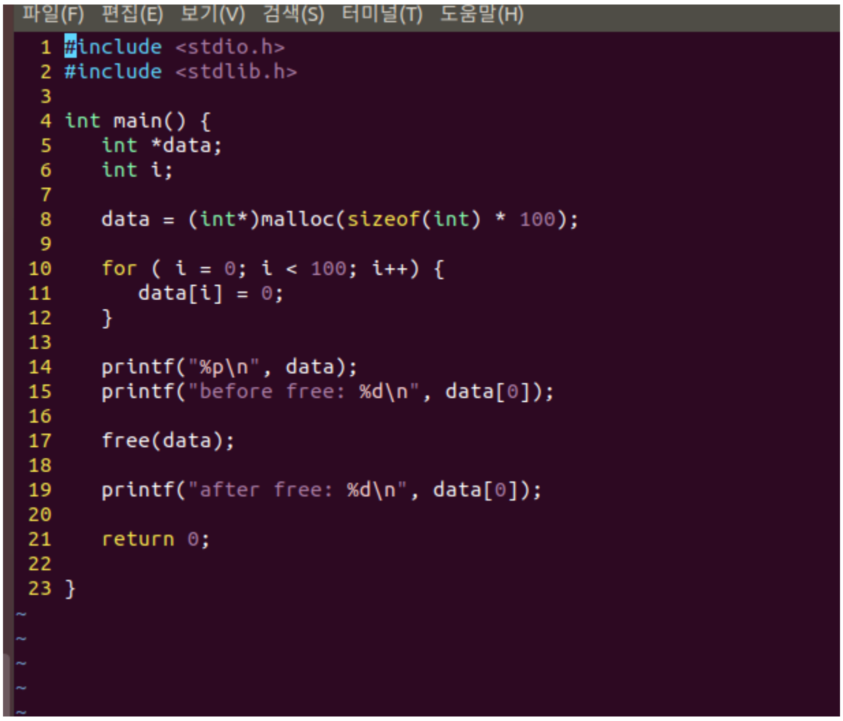
This is my code for Q5. I made simple program called h6 that added free function and tried to print the value of one of the elements of the array.



```
dasol@dasol-VirtualBox:~$ gcc -g -o h3 h3.c
dasol@dasol-VirtualBox:~$ gdb h3
GNU gdb (Ubuntu 8.1-0ubuntu3) 8.1.0.20180409-git
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 h3...done.
(gdb) run
Starting program: /home/dasol/h3
0x555555756260
[Inferior 1 (process 29752) exited normally]
(gdb) 
```

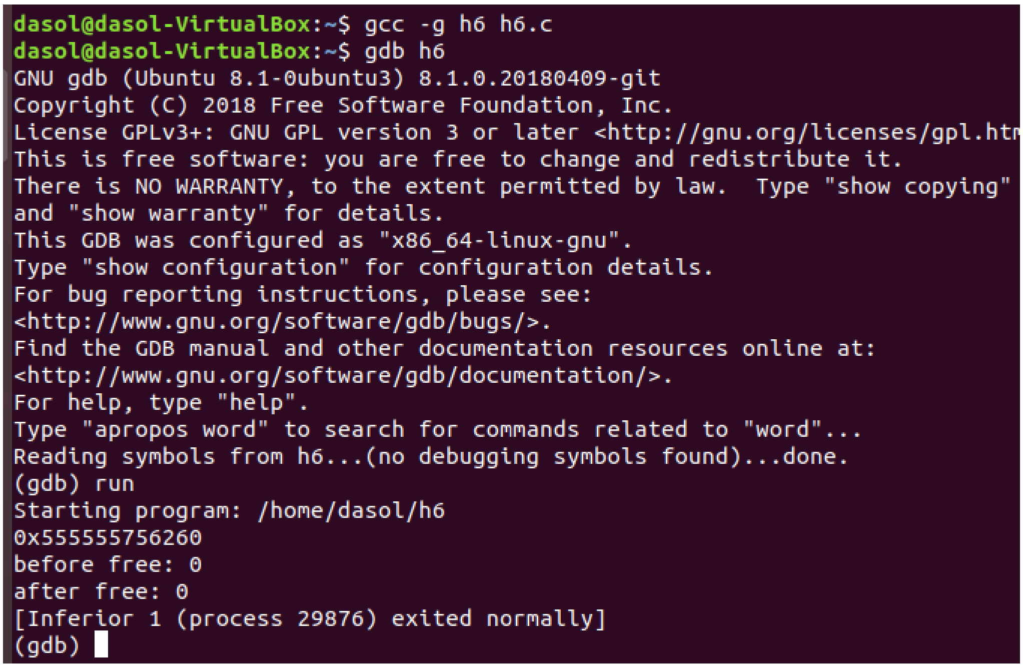
This is what happened when compile this into an executable called null by gdb. It has segmentation fault. As similar with Q1, it dereferences null.

I initialized memories as a null value because,



```c
#include <stdio.h>
#include <stdlib.h>

int main() {
   int *data;
   int i;

   data = (int*)malloc(sizeof(int) * 100);

   for ( i = 0; i < 100; i++) {
      data[i] = 0;
   }

   printf("%p\n", data);
   printf("before free: %d\n", data[0]);

   free(data);

   printf("after free: %d\n", data[0]);

   return 0;

}
```

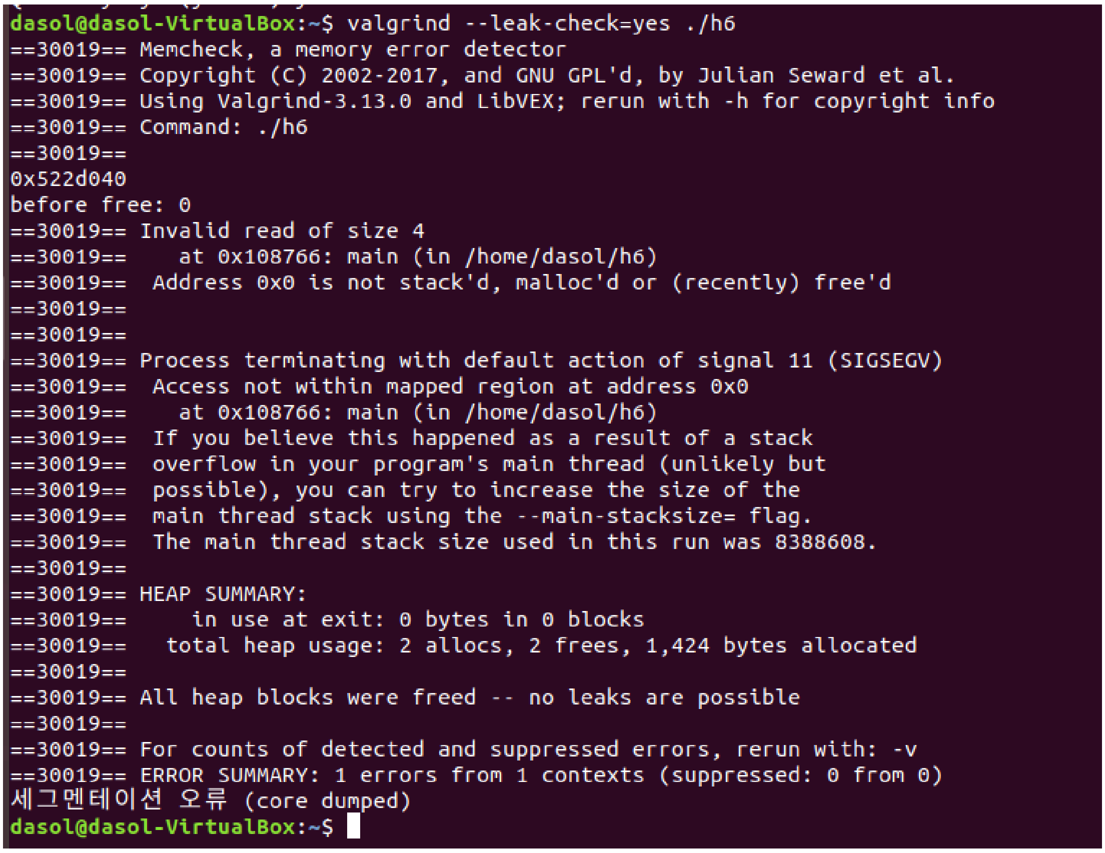
This is code that did not initialize as a null value.



```
dasol@dasol-VirtualBox:~$ gcc -g h6 h6.c
dasol@dasol-VirtualBox:~$ gdb h6
GNU gdb (Ubuntu 8.1-0ubuntu3) 8.1.0.20180409-git
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.htm
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 h6...(no debugging symbols found)...done.
(gdb) run
Starting program: /home/dasol/h6
0x555555756260
before free: 0
after free: 0
[Inferior 1 (process 29876) exited normally]
(gdb)
```

It works normally, when compile by gdb, because it only freed not initialize. But in the memory, it has trash value. So, it is dangerous not to initialize malloc() memory after using it.

Go back to the code in which memory is freed and initialized.



```
dasol@dasol-VirtualBox:~$ valgrind --leak-check=yes ./h6
==30019== Memcheck, a memory error detector
==30019== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==30019== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==30019== Command: ./h6
==30019==
0x522d040
before free: 0
==30019== Invalid read of size 4
==30019==    at 0x108766: main (in /home/dasol/h6)
==30019==  Address 0x0 is not stack'd, malloc'd or (recently) free'd
==30019==
==30019==
==30019== Process terminating with default action of signal 11 (SIGSEGV)
==30019==  Access not within mapped region at address 0x0
==30019==    at 0x108766: main (in /home/dasol/h6)
==30019==  If you believe this happened as a result of a stack
==30019==  overflow in your program's main thread (unlikely but
==30019==  possible), you can try to increase the size of the
==30019==  main thread stack using the --main-stacksize= flag.
==30019==  The main thread stack size used in this run was 8388608.
==30019==
==30019== HEAP SUMMARY:
==30019==     in use at exit: 0 bytes in 0 blocks
==30019==   total heap usage: 2 allocs, 2 frees, 1,424 bytes allocated
==30019==
==30019== All heap blocks were freed -- no leaks are possible
==30019==
==30019== For counts of detected and suppressed errors, rerun with: -v
==30019== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
세그멘테이션 오류 (core dumped)
dasol@dasol-VirtualBox:~$
```

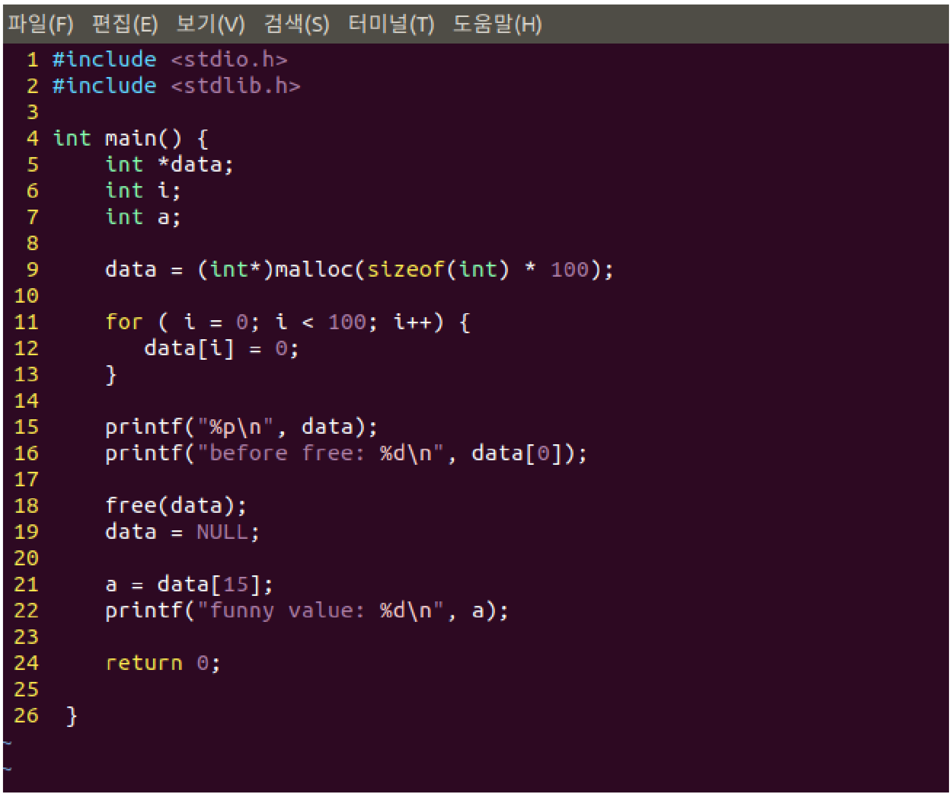
When code is compiled by the valgrind, it told me the problem is segmentation fault.

A segmentation fault occurs when a program attempts to access an unallowable memory area or attempts to access a memory area in an unallowable manner. Segmentation is a technique of memory management and protection used by the operating system. This has been largely replaced by paging, but the term of segmentation is still used like 'segmentation fault'. The code's problem is accessing not within mapped region at address 0x0

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?

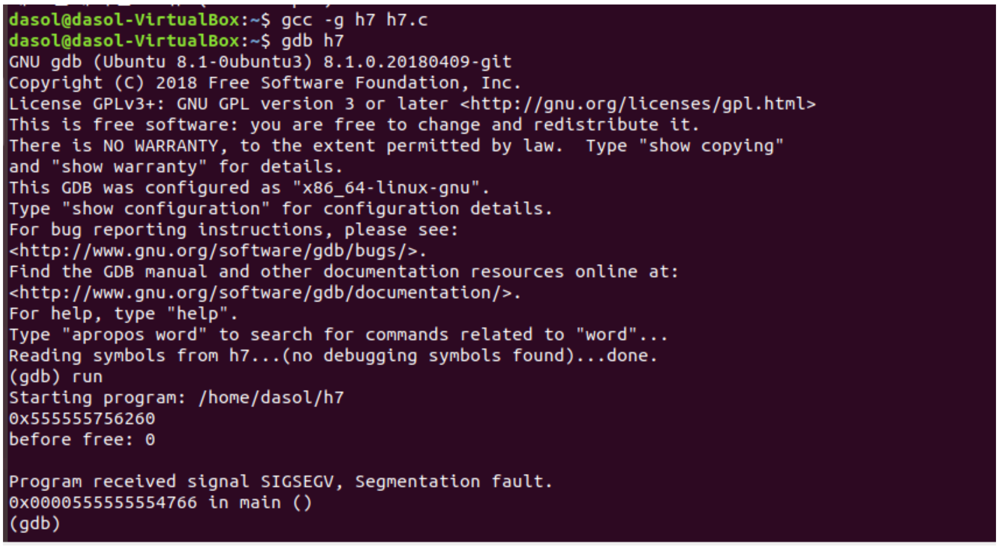
I don't know what this means exactly, so tried both. First code tried to pass a value which is free, and second code tried to free a specific value.

First code



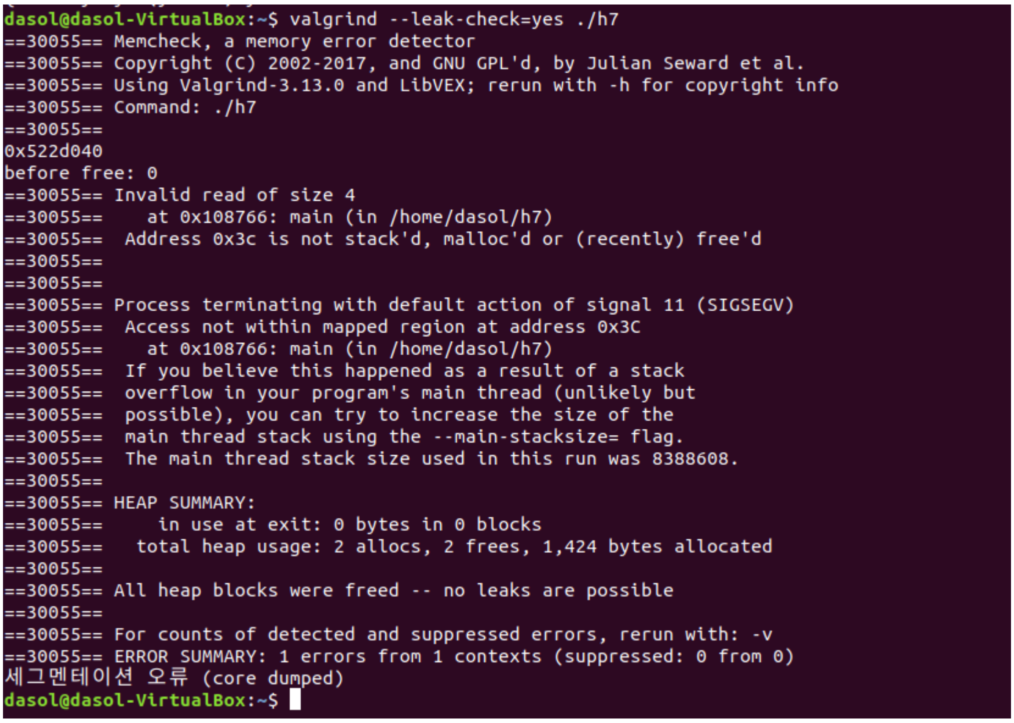
```
#include <stdio.h>
#include <stdlib.h>

int main() {
    int *data;
    int i;
    int a;

    data = (int*)malloc(sizeof(int) * 100);

    for ( i = 0; i < 100; i++) {
       data[i] = 0;
    }

    printf("%p\n", data);
    printf("before free: %d\n", data[0]);

    free(data);
    data = NULL;

    a = data[15];
    printf("funny value: %d\n", a);

    return 0;

 }
~
~
```

This is my code for Q7. I made simple program called h7 that tried to pass a value which is free.



```
dasol@dasol-VirtualBox:~$ gcc -g h7 h7.c
dasol@dasol-VirtualBox:~$ gdb h7
GNU gdb (Ubuntu 8.1-0ubuntu3) 8.1.0.20180409-git
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 h7...(no debugging symbols found)...done.
(gdb) run
Starting program: /home/dasol/h7
0x555555756260
before free: 0

Program received signal SIGSEGV, Segmentation fault.
0x0000555555554766 in main ()
(gdb)
```

This is what happened when compile this into an executable called null by gdb. It has segmentation fault. As similar with Q1 and Q6, it dereferences null.

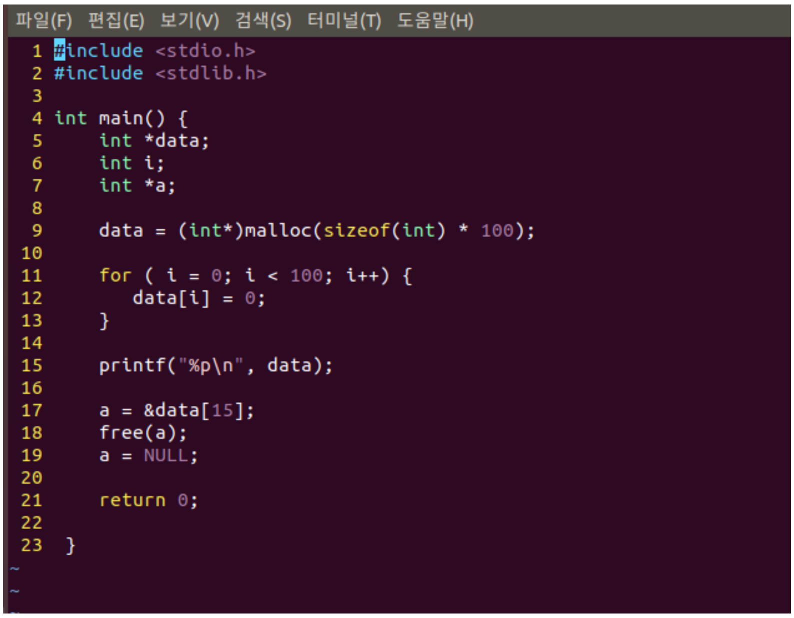


```
dasol@dasol-VirtualBox:~$ valgrind --leak-check=yes ./h7
==30055== Memcheck, a memory error detector
==30055== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==30055== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==30055== Command: ./h7
==30055==
0x522d040
before free: 0
==30055== Invalid read of size 4
==30055==    at 0x108766: main (in /home/dasol/h7)
==30055==  Address 0x3c is not stack'd, malloc'd or (recently) free'd
==30055==
==30055==
==30055== Process terminating with default action of signal 11 (SIGSEGV)
==30055==  Access not within mapped region at address 0x3C
==30055==    at 0x108766: main (in /home/dasol/h7)
==30055==  If you believe this happened as a result of a stack
==30055==  overflow in your program's main thread (unlikely but
==30055==  possible), you can try to increase the size of the
==30055==  main thread stack using the --main-stacksize= flag.
==30055==  The main thread stack size used in this run was 8388608.
==30055==
==30055== HEAP SUMMARY:
==30055==     in use at exit: 0 bytes in 0 blocks
==30055==   total heap usage: 2 allocs, 2 frees, 1,424 bytes allocated
==30055==
==30055== All heap blocks were freed -- no leaks are possible
==30055==
==30055== For counts of detected and suppressed errors, rerun with: -v
==30055== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
세그멘테이션 오류 (core dumped)
dasol@dasol-VirtualBox:~$
```

When code is compiled by the valgrind, it told me the problem is segmentation fault.

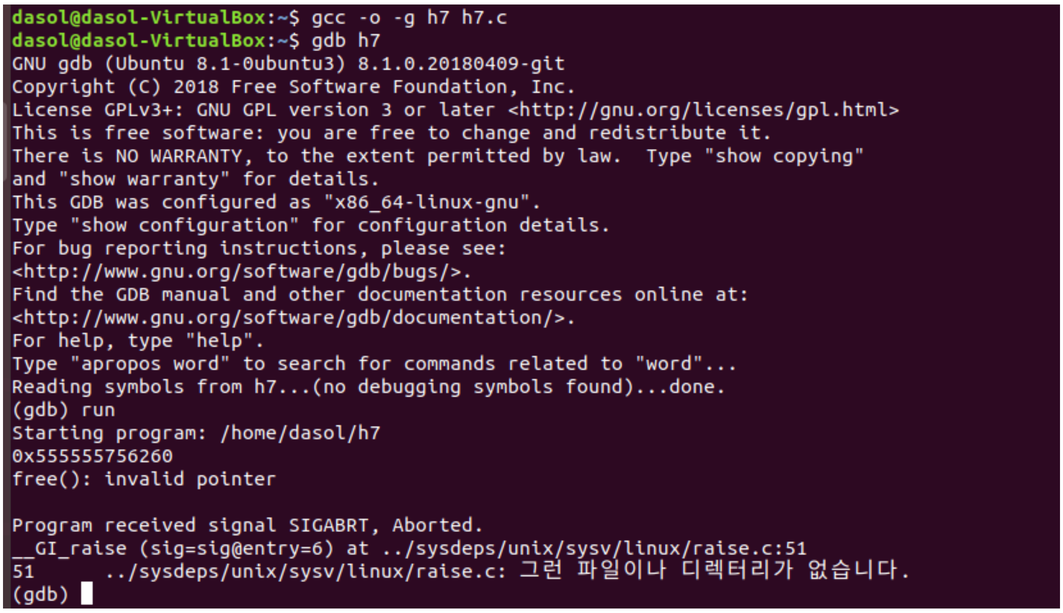
A segmentation fault occurs when a program attempts to access an unallowable memory area or attempts to access a memory area in an unallowable manner. Segmentation is a technique of memory management and protection used by the operating system. This has been largely replaced by paging, but the term of segmentation is still used like 'segmentation fault'. The code's problem is accessing not within mapped region at address 0x0

Second code



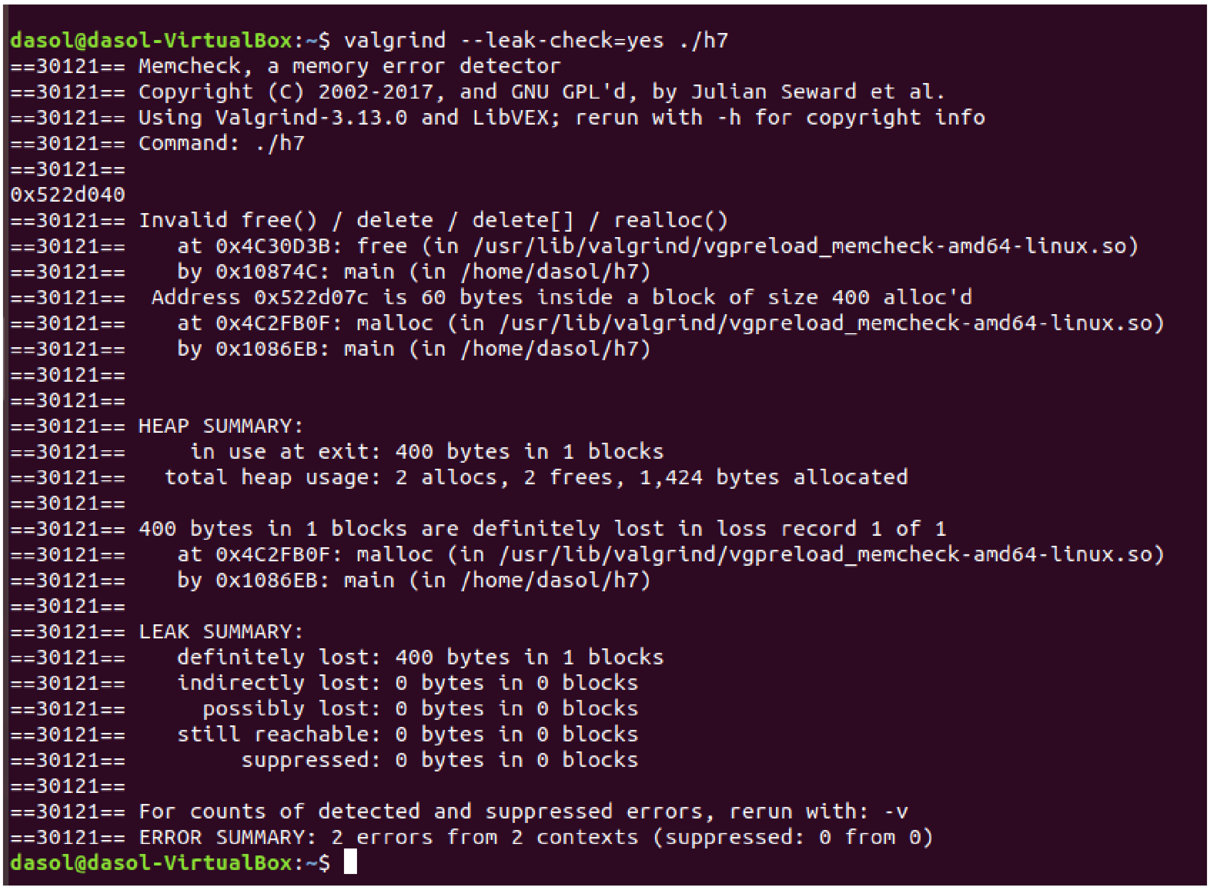
```
#include <stdio.h>
#include <stdlib.h>

int main() {
    int *data;
    int i;
    int *a;

    data = (int*)malloc(sizeof(int) * 100);

    for ( i = 0; i < 100; i++) {
       data[i] = 0;
    }

    printf("%p\n", data);

    a = &data[15];
    free(a);
    a = NULL;

    return 0;

 }
```

This is code that tried to free a specific value among array.



```
dasol@dasol-VirtualBox:~$ gcc -o -g h7 h7.c
dasol@dasol-VirtualBox:~$ gdb h7
GNU gdb (Ubuntu 8.1-0ubuntu3) 8.1.0.20180409-git
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 h7...(no debugging symbols found)...done.
(gdb) run
Starting program: /home/dasol/h7
0x555555756260
free(): invalid pointer

Program received signal SIGABRT, Aborted.
__GI_raise (sig=sig@entry=6) at ../sysdeps/unix/sysv/linux/raise.c:51
51      ../sysdeps/unix/sysv/linux/raise.c: 그런 파일이나 디렉터리가 없습니다.
(gdb) 
```

This is what happened when compile this code by gdb. It told me the pointer is unvalid. But I don’t understand what it is exactly wrong.



```
dasol@dasol-VirtualBox:~$ valgrind --leak-check=yes ./h7
==30121== Memcheck, a memory error detector
==30121== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==30121== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==30121== Command: ./h7
==30121==
0x522d040
==30121== Invalid free() / delete / delete[] / realloc()
==30121==    at 0x4C30D3B: free (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==30121==    by 0x10874C: main (in /home/dasol/h7)
==30121==  Address 0x522d07c is 60 bytes inside a block of size 400 alloc'd
==30121==    at 0x4C2FB0F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==30121==    by 0x1086EB: main (in /home/dasol/h7)
==30121==
==30121==
==30121== HEAP SUMMARY:
==30121==     in use at exit: 400 bytes in 1 blocks
==30121==   total heap usage: 2 allocs, 2 frees, 1,424 bytes allocated
==30121==
==30121== 400 bytes in 1 blocks are definitely lost in loss record 1 of 1
==30121==    at 0x4C2FB0F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==30121==    by 0x1086EB: main (in /home/dasol/h7)
==30121==
==30121== LEAK SUMMARY:
==30121==    definitely lost: 400 bytes in 1 blocks
==30121==    indirectly lost: 0 bytes in 0 blocks
==30121==      possibly lost: 0 bytes in 0 blocks
==30121==    still reachable: 0 bytes in 0 blocks
==30121==         suppressed: 0 bytes in 0 blocks
==30121==
==30121== For counts of detected and suppressed errors, rerun with: -v
==30121== ERROR SUMMARY: 2 errors from 2 contexts (suppressed: 0 from 0)
dasol@dasol-VirtualBox:~$ 
```

It told me that what my code wants to be free is invalid, so it has problem to free specific pointer. I think it is impossible to free only specific pointer. Also, the code has leak of memory at heap memory. Generally, memory that allocated by malloc() stored in heap memory. If program forgets to free it before exiting, memory in heap memory didn't not delete unlike stack memory. In this case, I tried only specific pointer, so the rest of memories are not free. Therefore, program has leak of memory problem.

9. Spend more time and read about using gdb and valgrind. Knowing your tools is critical; spend the time and learn how to become an expert debugger in the UNIX and C environment.

I think valgrind is very powerful tool, it makes easy for debugging. It told programmer that which problems are occurred and the address of problem. I want to learn how to use it and make good use of it.