VIVA Answers - VIVEK MUNNAA D

1.Using GBD, list the commands for each question and list down the leaks in the provided c program “valgrind_prgm_1” file using valgrind and fix the code.

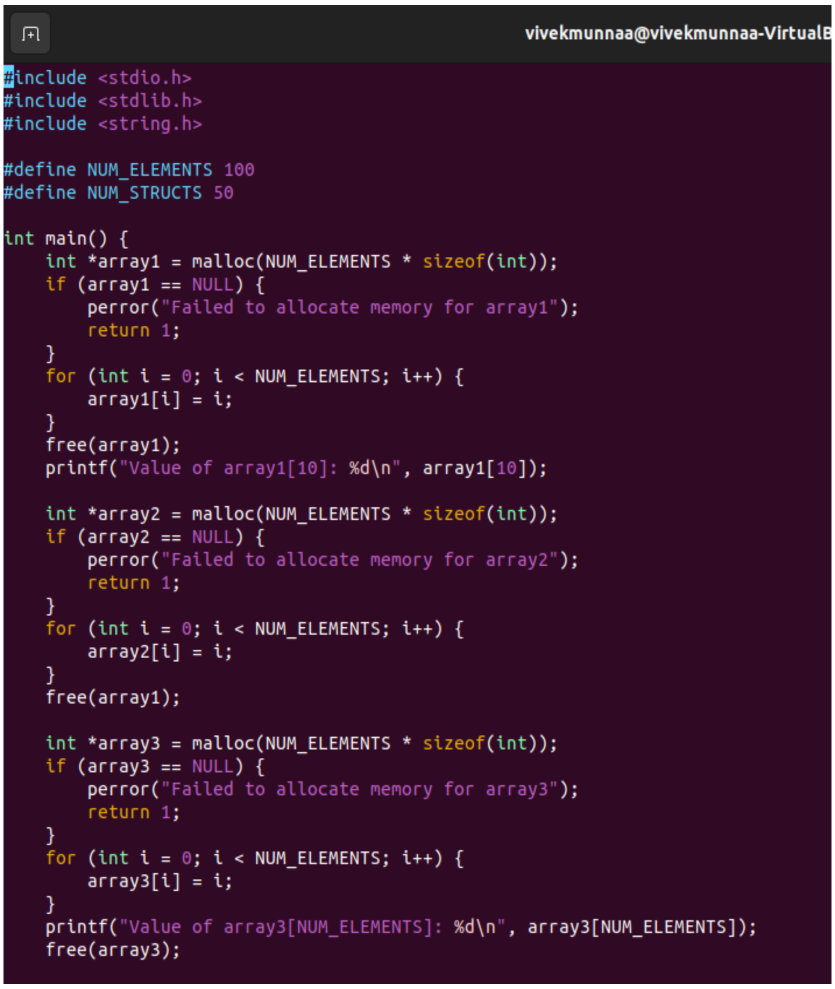
2.How do you list the source code of the main function in GDB?

3.How do you run the program inside GDB?

4.How do you display the current contents of the CPU registers in GDB?

5.How do you display information about all breakpoints in GDB?

-> **VALGRIND EXECUTION:**

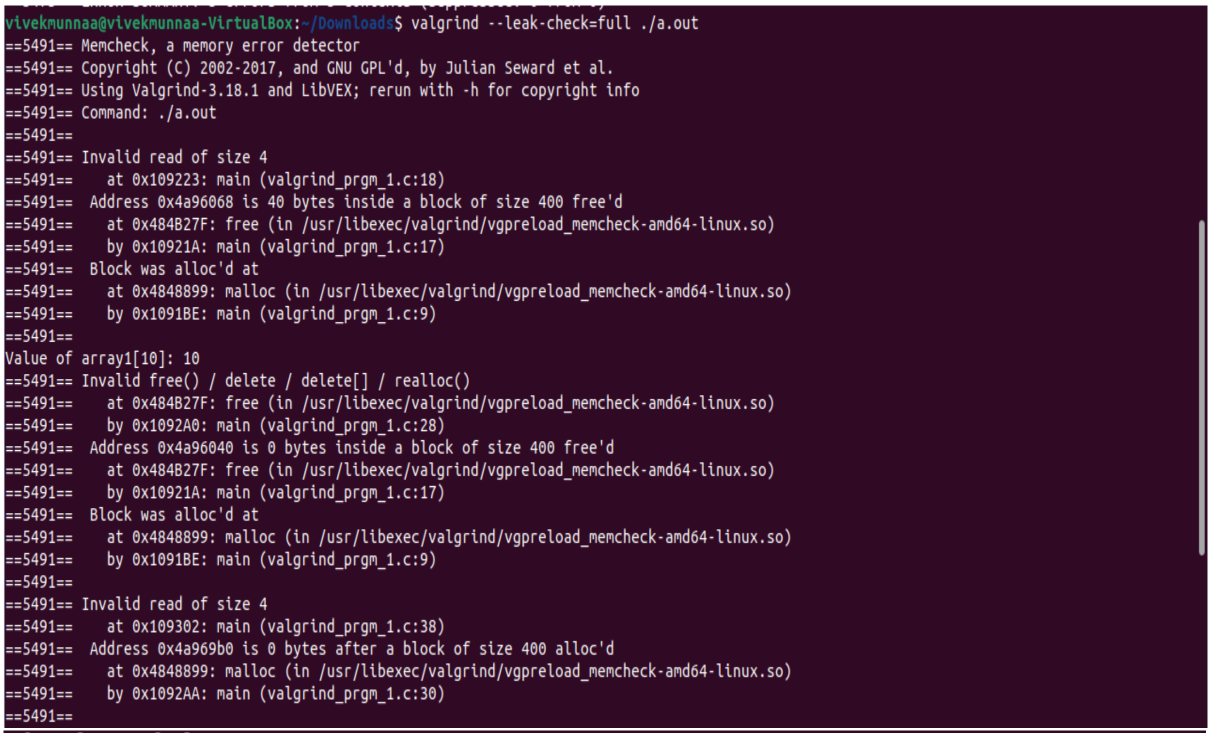


```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define NUM_ELEMENTS 100
#define NUM_STRUCTS 50

int main() {
    int *array1 = malloc(NUM_ELEMENTS * sizeof(int));
    if (array1 == NULL) {
        perror("Failed to allocate memory for array1");
        return 1;
    }
    for (int i = 0; i < NUM_ELEMENTS; i++) {
        array1[i] = i;
    }
    free(array1);
    printf("Value of array1[10]: %d\n", array1[10]);

    int *array2 = malloc(NUM_ELEMENTS * sizeof(int));
    if (array2 == NULL) {
        perror("Failed to allocate memory for array2");
        return 1;
    }
    for (int i = 0; i < NUM_ELEMENTS; i++) {
        array2[i] = i;
    }
    free(array1);

    int *array3 = malloc(NUM_ELEMENTS * sizeof(int));
    if (array3 == NULL) {
        perror("Failed to allocate memory for array3");
        return 1;
    }
    for (int i = 0; i < NUM_ELEMENTS; i++) {
        array3[i] = i;
    }
    printf("Value of array3[NUM_ELEMENTS]: %d\n", array3[NUM_ELEMENTS]);
    free(array3);
```

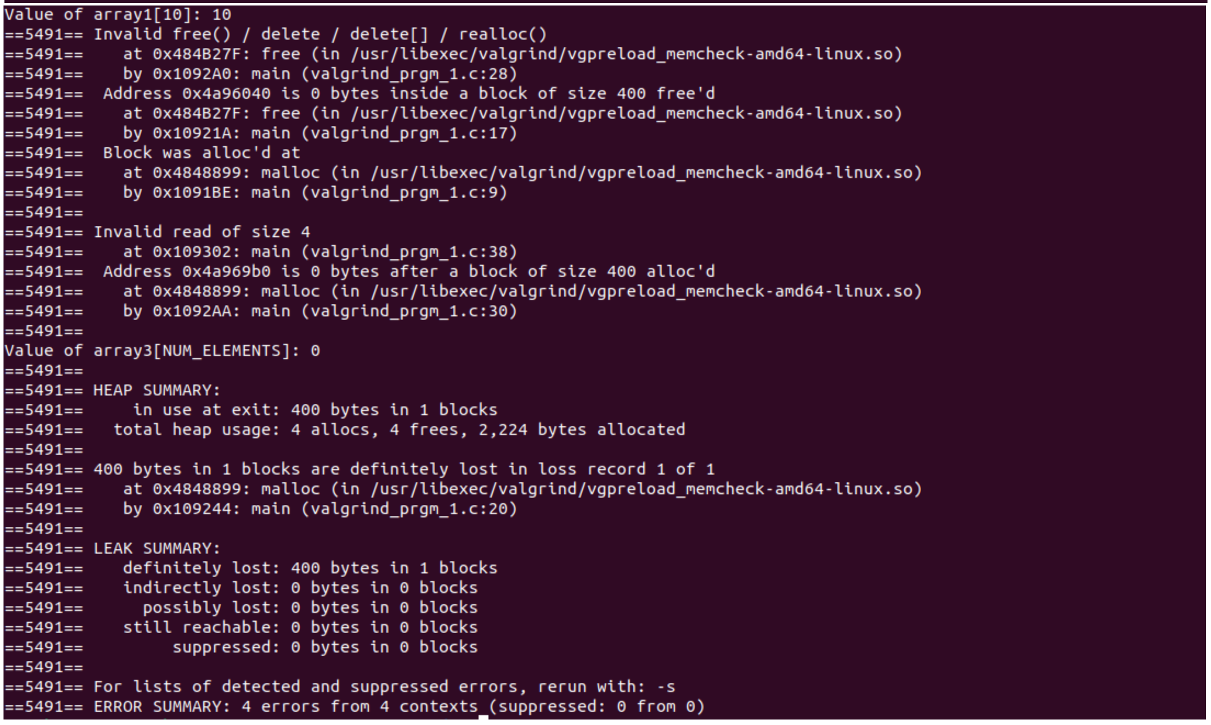
I created a file **valgrind_prgm_1.c** and compiled it with the command:

gcc -g valgrind_pgrm_1.c, Then ran it using the valgrind command :

Valgrind –leak-check=full ./a.out



```
vivekmunnaa@vivekmunnaa-VirtualBox:~/Downloads$ valgrind --leak-check=full ./a.out
==5491== Memcheck, a memory error detector
==5491== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==5491== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==5491== Command: ./a.out
==5491==
==5491== Invalid read of size 4
==5491==    at 0x109223: main (valgrind_prgm_1.c:18)
==5491==  Address 0x4a96068 is 40 bytes inside a block of size 400 free'd
==5491==    at 0x484B27F: free (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==5491==    by 0x10921A: main (valgrind_prgm_1.c:17)
==5491==  Block was alloc'd at
==5491==    at 0x4848899: malloc (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==5491==    by 0x1091BE: main (valgrind_prgm_1.c:9)
==5491==
Value of array1[10]: 10
==5491== Invalid free() / delete / delete[] / realloc()
==5491==    at 0x484B27F: free (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==5491==    by 0x1092A0: main (valgrind_prgm_1.c:28)
==5491==  Address 0x4a96040 is 0 bytes inside a block of size 400 free'd
==5491==    at 0x484B27F: free (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==5491==    by 0x10921A: main (valgrind_prgm_1.c:17)
==5491==  Block was alloc'd at
==5491==    at 0x4848899: malloc (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==5491==    by 0x1091BE: main (valgrind_prgm_1.c:9)
==5491==
==5491== Invalid read of size 4
==5491==    at 0x109302: main (valgrind_prgm_1.c:38)
==5491==  Address 0x4a969b0 is 0 bytes after a block of size 400 alloc'd
==5491==    at 0x4848899: malloc (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==5491==    by 0x1092AA: main (valgrind_prgm_1.c:30)
==5491==
```

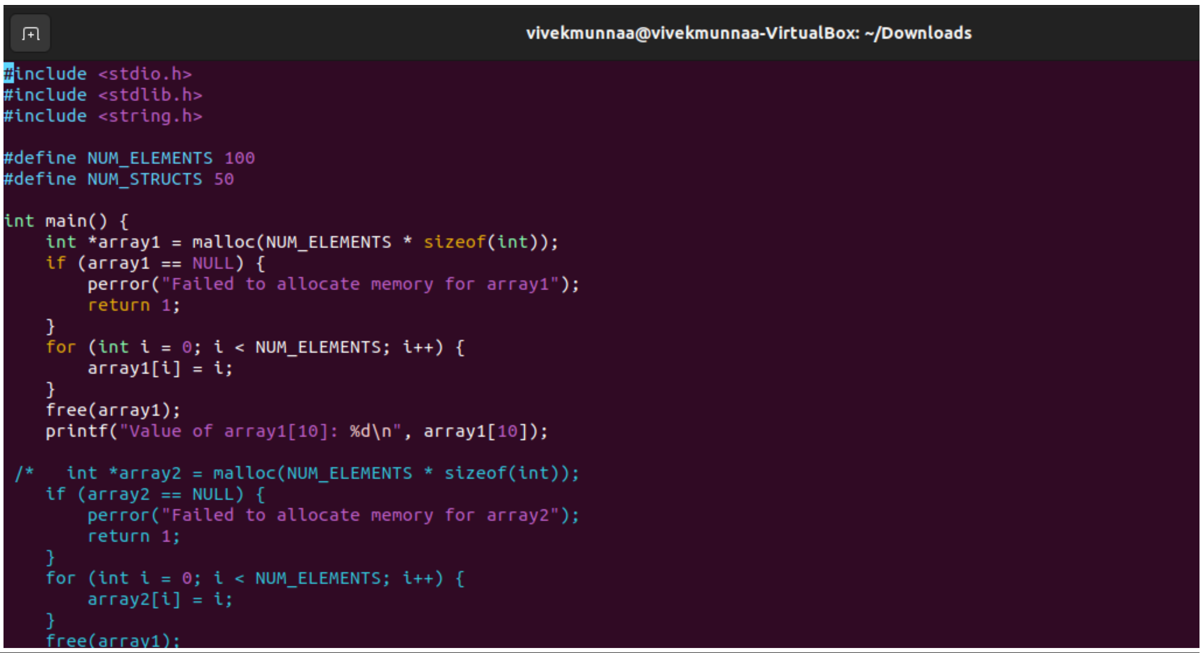


```
Value of array1[10]: 10
==5491== Invalid free() / delete / delete[] / realloc()
==5491==    at 0x484B27F: free (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==5491==    by 0x1092A0: main (valgrind_prgm_1.c:28)
==5491==  Address 0x4a96040 is 0 bytes inside a block of size 400 free'd
==5491==    at 0x484B27F: free (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==5491==    by 0x10921A: main (valgrind_prgm_1.c:17)
==5491==  Block was alloc'd at
==5491==    at 0x4848899: malloc (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==5491==    by 0x1091BE: main (valgrind_prgm_1.c:9)
==5491==
==5491== Invalid read of size 4
==5491==    at 0x109302: main (valgrind_prgm_1.c:38)
==5491==  Address 0x4a969b0 is 0 bytes after a block of size 400 alloc'd
==5491==    at 0x4848899: malloc (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==5491==    by 0x1092AA: main (valgrind_prgm_1.c:30)
==5491==
Value of array3[NUM_ELEMENTS]: 0
==5491==
==5491== HEAP SUMMARY:
==5491==     in use at exit: 400 bytes in 1 blocks
==5491==   total heap usage: 4 allocs, 4 frees, 2,224 bytes allocated
==5491==
==5491== 400 bytes in 1 blocks are definitely lost in loss record 1 of 1
==5491==    at 0x4848899: malloc (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==5491==    by 0x109244: main (valgrind_prgm_1.c:20)
==5491==
==5491== LEAK SUMMARY:
==5491==    definitely lost: 400 bytes in 1 blocks
==5491==    indirectly lost: 0 bytes in 0 blocks
==5491==      possibly lost: 0 bytes in 0 blocks
==5491==    still reachable: 0 bytes in 0 blocks
==5491==         suppressed: 0 bytes in 0 blocks
==5491==
==5491== For lists of detected and suppressed errors, rerun with: -s
==5491== ERROR SUMMARY: 4 errors from 4 contexts (suppressed: 0 from 0)
```

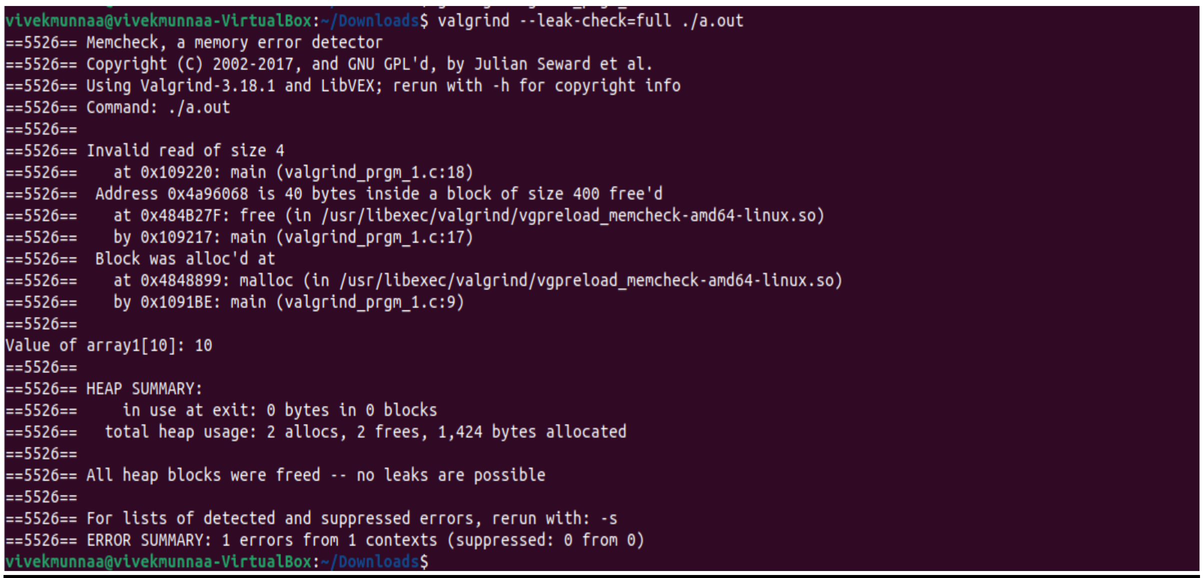
This the output I got initially with 4 error and 4 contexts for compiling and evaluating the entire program.

For making it easier to identify errors, I compiled the program and executed Valgrind for each ARRAY involved separately, commenting out the other arrays.

<u>ARRAY 1:</u>



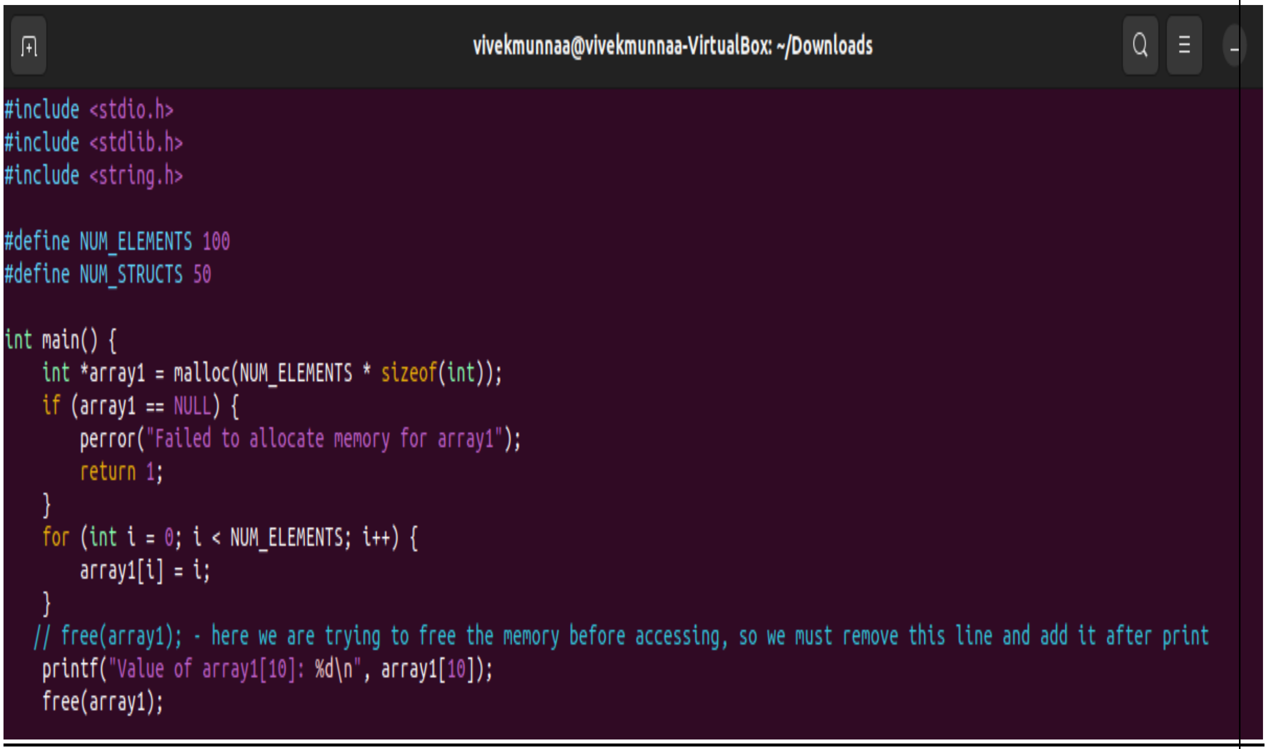
```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define NUM_ELEMENTS 100
#define NUM_STRUCTS 50

int main() {
    int *array1 = malloc(NUM_ELEMENTS * sizeof(int));
    if (array1 == NULL) {
        perror("Failed to allocate memory for array1");
        return 1;
    }
    for (int i = 0; i < NUM_ELEMENTS; i++) {
        array1[i] = i;
    }
    free(array1);
    printf("Value of array1[10]: %d\n", array1[10]);

 /*   int *array2 = malloc(NUM_ELEMENTS * sizeof(int));
    if (array2 == NULL) {
        perror("Failed to allocate memory for array2");
        return 1;
    }
    for (int i = 0; i < NUM_ELEMENTS; i++) {
        array2[i] = i;
    }
    free(array1);
```



```
vivekmunnaa@vivekmunnaa-VirtualBox:~/Downloads$ valgrind --leak-check=full ./a.out
==5526== Memcheck, a memory error detector
==5526== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==5526== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==5526== Command: ./a.out
==5526==
==5526== Invalid read of size 4
==5526==    at 0x109220: main (valgrind_prgm_1.c:18)
==5526==  Address 0x4a96068 is 40 bytes inside a block of size 400 free'd
==5526==    at 0x484B27F: free (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==5526==    by 0x109217: main (valgrind_prgm_1.c:17)
==5526==  Block was alloc'd at
==5526==    at 0x4848899: malloc (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==5526==    by 0x1091BE: main (valgrind_prgm_1.c:9)
==5526==
Value of array1[10]: 10
==5526==
==5526== HEAP SUMMARY:
==5526==     in use at exit: 0 bytes in 0 blocks
==5526==   total heap usage: 2 allocs, 2 frees, 1,424 bytes allocated
==5526==
==5526== All heap blocks were freed -- no leaks are possible
==5526==
==5526== For lists of detected and suppressed errors, rerun with: -s
==5526== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
vivekmunnaa@vivekmunnaa-VirtualBox:~/Downloads$
```

- Valgrind reported 1 error.
- The context given was "Invalid read of size 4" – this means the code is trying to access a variable of size 4 which is not initialized.
- Also, It reports the error is in line number 18.

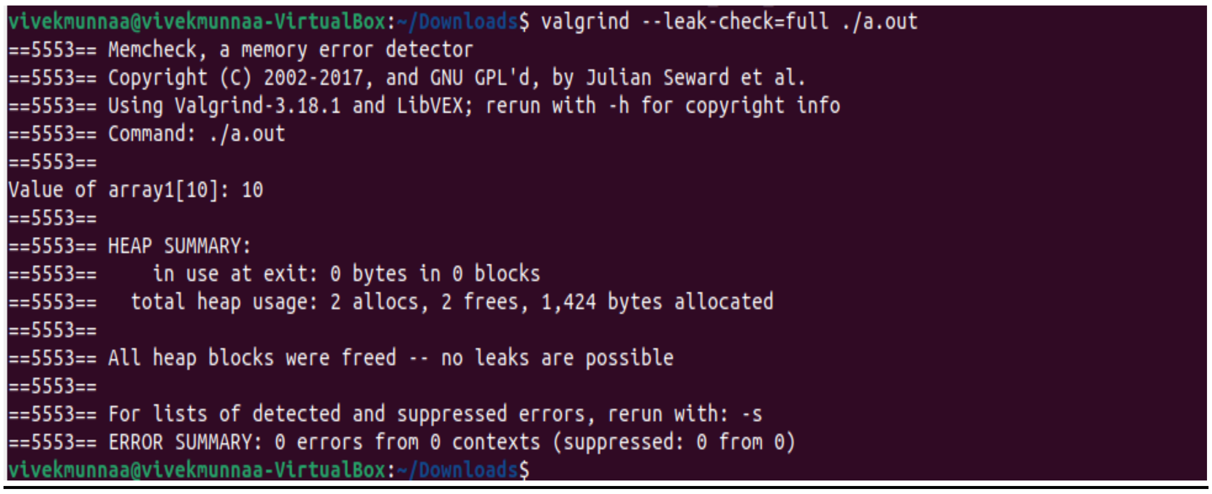
Solution:



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

#define NUM_ELEMENTS 100
#define NUM_STRUCTS 50

int main() {
    int *array1 = malloc(NUM_ELEMENTS * sizeof(int));
    if (array1 == NULL) {
        perror("Failed to allocate memory for array1");
        return 1;
    }
    for (int i = 0; i < NUM_ELEMENTS; i++) {
        array1[i] = i;
    }
   // free(array1); - here we are trying to free the memory before accessing, so we must remove this line and add it after print
    printf("Value of array1[10]: %d\n", array1[10]);
    free(array1);
```

- The error was due to freeing the allocated memory before print.
- We can rectify the error by freeing after print statement.
- (Or) by Commenting out the print statement and leaving free(array1) at the same place.

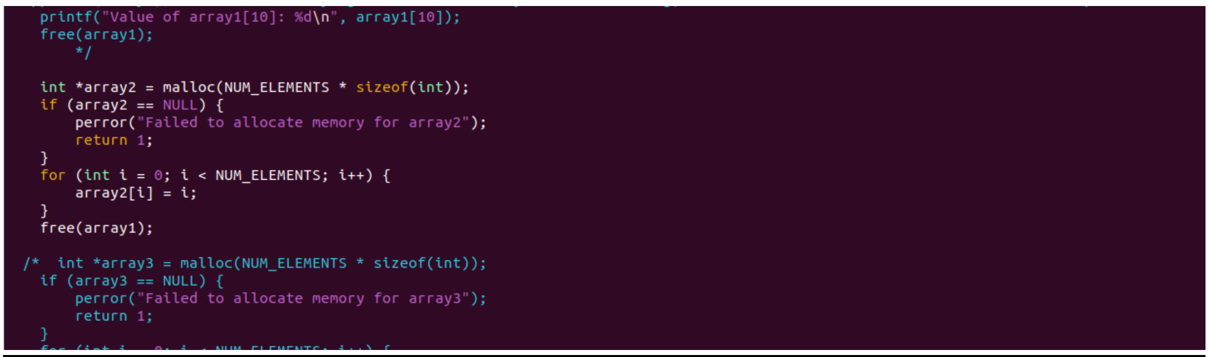
OUTPUT AFTER CHANGES:



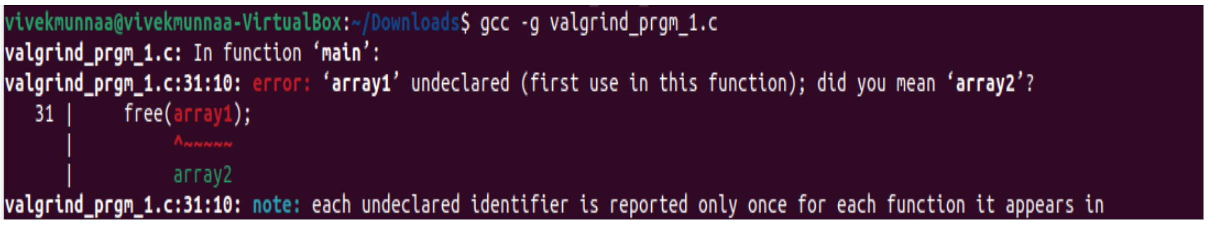
```
vivekmunnaa@vivekmunnaa-VirtualBox:~/Downloads$ valgrind --leak-check=full ./a.out
==5553== Memcheck, a memory error detector
==5553== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==5553== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==5553== Command: ./a.out
==5553==
Value of array1[10]: 10
==5553==
==5553== HEAP SUMMARY:
==5553==     in use at exit: 0 bytes in 0 blocks
==5553==   total heap usage: 2 allocs, 2 frees, 1,424 bytes allocated
==5553==
==5553== All heap blocks were freed -- no leaks are possible
==5553==
==5553== For lists of detected and suppressed errors, rerun with: -s
==5553== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
vivekmunnaa@vivekmunnaa-VirtualBox:~/Downloads$
```

- Valgrind reported no errors for array1 after the changes
- And I also got the o/p: Value of array1[10]:10

ARRAY2:



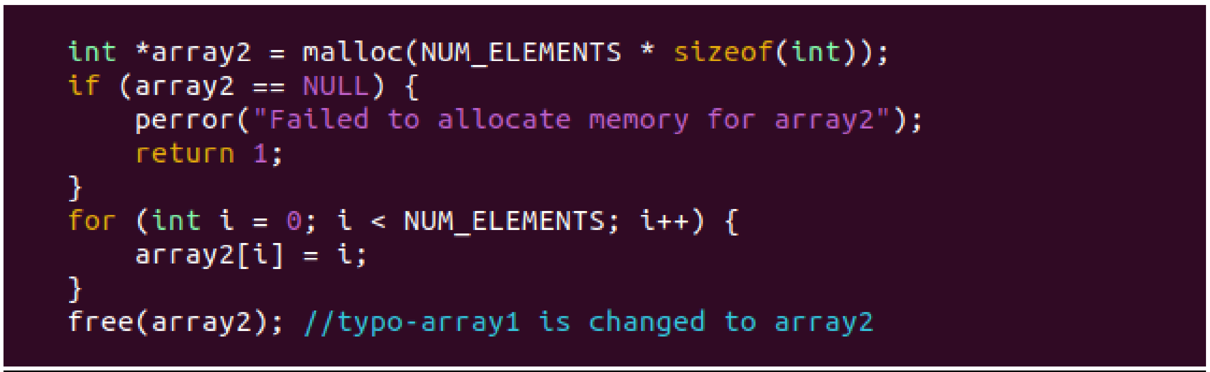
```
    printf("Value of array1[10]: %d\n", array1[10]);
    free(array1);
        */

    int *array2 = malloc(NUM_ELEMENTS * sizeof(int));
    if (array2 == NULL) {
        perror("Failed to allocate memory for array2");
        return 1;
    }
    for (int i = 0; i < NUM_ELEMENTS; i++) {
        array2[i] = i;
    }
    free(array1);

  /*  int *array3 = malloc(NUM_ELEMENTS * sizeof(int));
    if (array3 == NULL) {
        perror("Failed to allocate memory for array3");
        return 1;
    }
```



```
vivekmunnaa@vivekmunnaa-VirtualBox:~/Downloads$ gcc -g valgrind_prgm_1.c
valgrind_prgm_1.c: In function ‘main’:
valgrind_prgm_1.c:31:10: error: ‘array1’ undeclared (first use in this function); did you mean ‘array2’?
   31 |     free(array1);
      |          ^~~~~~
      |          array2
valgrind_prgm_1.c:31:10: note: each undeclared identifier is reported only once for each function it appears in
```

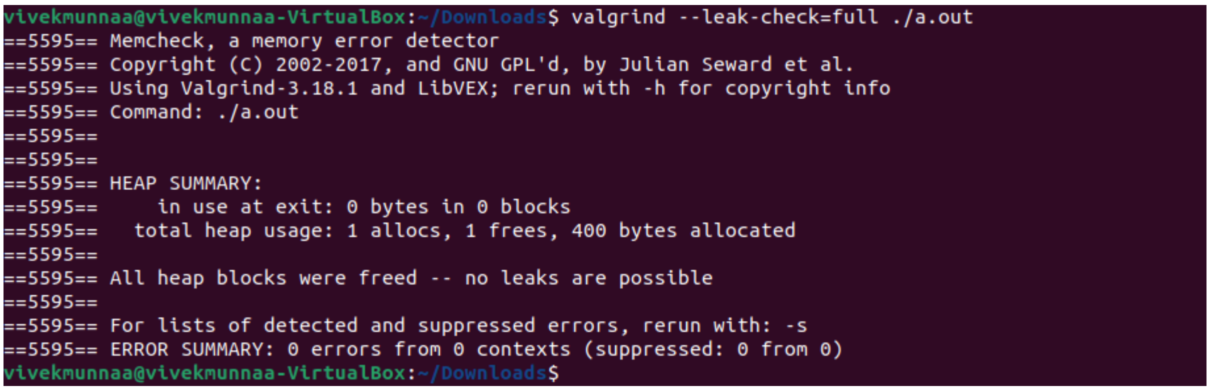
- I got an error while compiling, stating that in line number 31: array1 was undeclared.
- Here instead of freeing array2, the program tried to free array1

Solution:



```
    int *array2 = malloc(NUM_ELEMENTS * sizeof(int));
    if (array2 == NULL) {
        perror("Failed to allocate memory for array2");
        return 1;
    }
    for (int i = 0; i < NUM_ELEMENTS; i++) {
        array2[i] = i;
    }
    free(array2); //typo-array1 is changed to array2
```

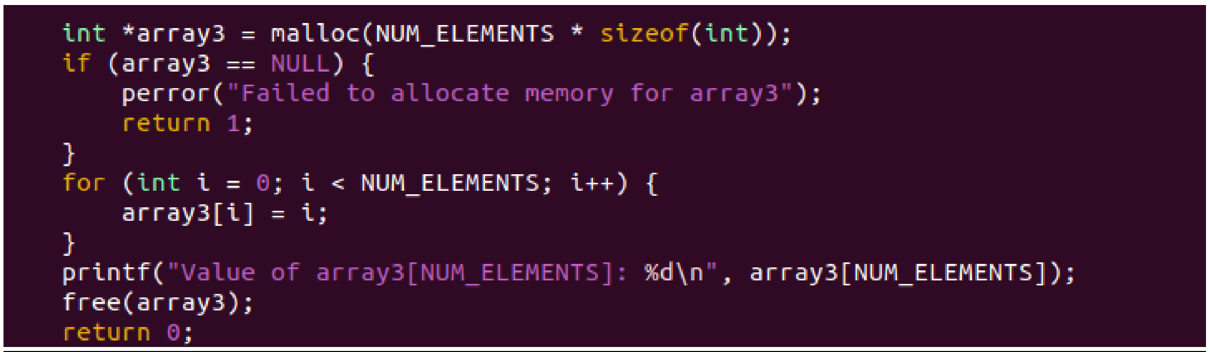
- I changed free(array1) to free(array2)



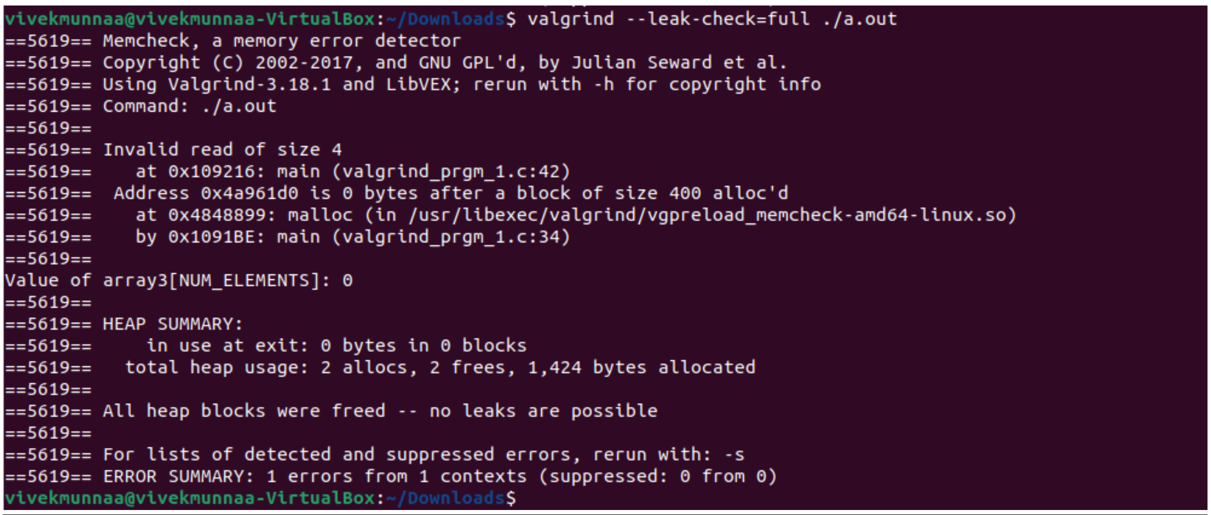
```
vivekmunnaa@vivekmunnaa-VirtualBox:~/Downloads$ valgrind --leak-check=full ./a.out
==5595== Memcheck, a memory error detector
==5595== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==5595== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==5595== Command: ./a.out
==5595==
==5595==
==5595== HEAP SUMMARY:
==5595==     in use at exit: 0 bytes in 0 blocks
==5595==   total heap usage: 1 allocs, 1 frees, 400 bytes allocated
==5595==
==5595== All heap blocks were freed -- no leaks are possible
==5595==
==5595== For lists of detected and suppressed errors, rerun with: -s
==5595== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
vivekmunnaa@vivekmunnaa-VirtualBox:~/Downloads$
```

- Then ran valgrind command, which then reported 0 errors.

ARRAY3:



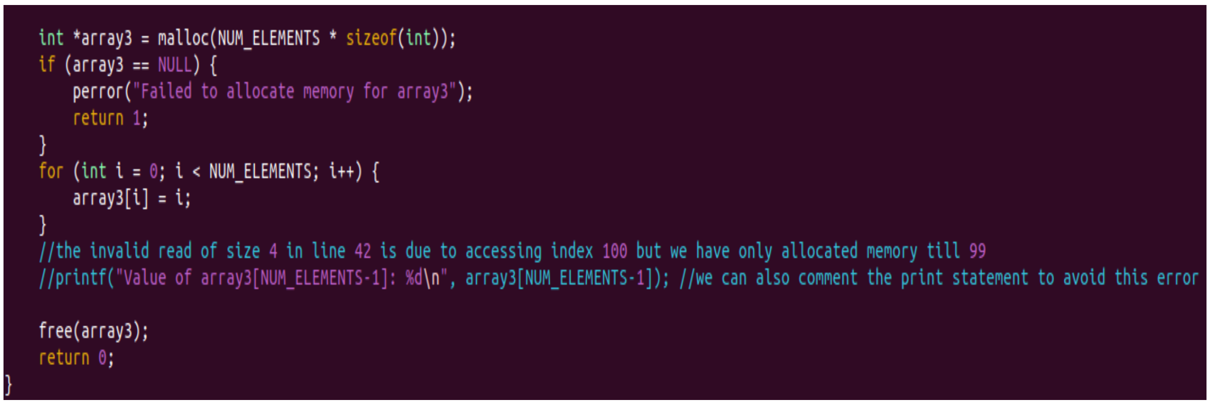
```
    int *array3 = malloc(NUM_ELEMENTS * sizeof(int));
    if (array3 == NULL) {
        perror("Failed to allocate memory for array3");
        return 1;
    }
    for (int i = 0; i < NUM_ELEMENTS; i++) {
        array3[i] = i;
    }
    printf("Value of array3[NUM_ELEMENTS]: %d\n", array3[NUM_ELEMENTS]);
    free(array3);
    return 0;
```



```
vivekmunnaa@vivekmunnaa-VirtualBox:~/Downloads$ valgrind --leak-check=full ./a.out
==5619== Memcheck, a memory error detector
==5619== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==5619== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==5619== Command: ./a.out
==5619==
==5619== Invalid read of size 4
==5619==    at 0x109216: main (valgrind_prgm_1.c:42)
==5619==  Address 0x4a961d0 is 0 bytes after a block of size 400 alloc'd
==5619==    at 0x4848899: malloc (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==5619==    by 0x1091BE: main (valgrind_prgm_1.c:34)
==5619==
Value of array3[NUM_ELEMENTS]: 0
==5619==
==5619== HEAP SUMMARY:
==5619==     in use at exit: 0 bytes in 0 blocks
==5619==   total heap usage: 2 allocs, 2 frees, 1,424 bytes allocated
==5619==
==5619== All heap blocks were freed -- no leaks are possible
==5619==
==5619== For lists of detected and suppressed errors, rerun with: -s
==5619== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
vivekmunnaa@vivekmunnaa-VirtualBox:~/Downloads$
```

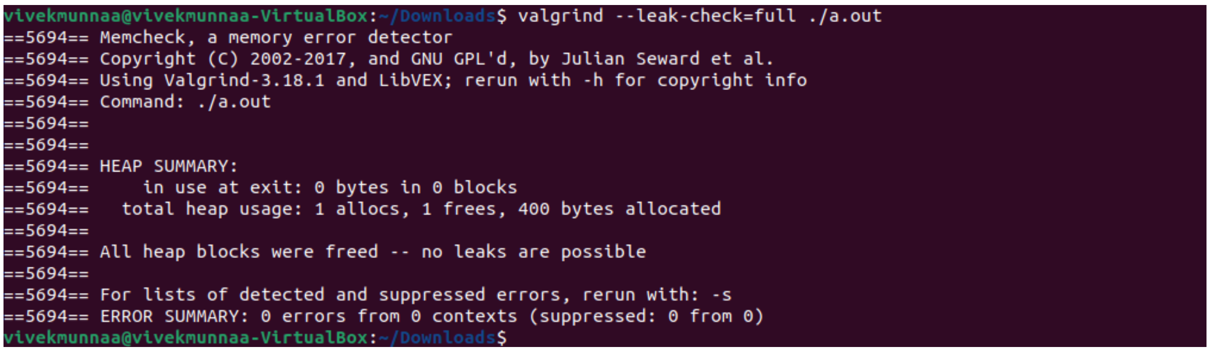
- Valgrind reported 1 error.
- The context given was “Invalid read of size 4”
- This means the code was trying to read 4 bytes of uninitialized memory.
- Also the line number which was specified in the valgrind report was 42.

Solution:



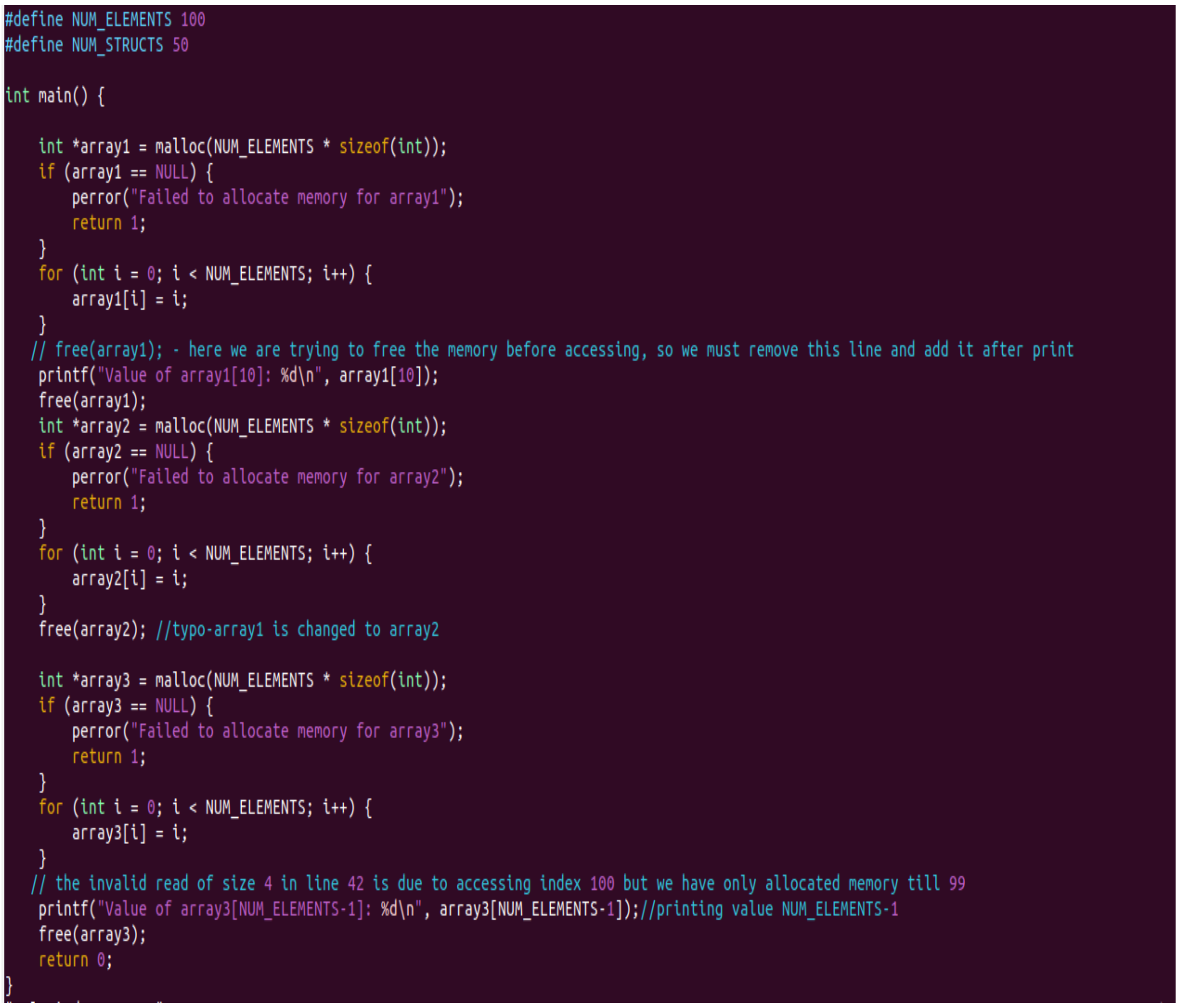
```
    int *array3 = malloc(NUM_ELEMENTS * sizeof(int));
    if (array3 == NULL) {
        perror("Failed to allocate memory for array3");
        return 1;
    }
    for (int i = 0; i < NUM_ELEMENTS; i++) {
        array3[i] = i;
    }
    //the invalid read of size 4 in line 42 is due to accessing index 100 but we have only allocated memory till 99
    //printf("Value of array3[NUM_ELEMENTS-1]: %d\n", array3[NUM_ELEMENTS-1]); //we can also comment the print statement to avoid this error

    free(array3);
    return 0;
}
```

- On line 42, the program was printing array3[NUM_ELEMENTS] initially, this is the reason we got the error as we are accessing index 100 which is out of bound as we have allocated memory only from 0-99.
- So, to rectify it we have changed the print statement to print **array3[NUM_ELEMENTS-1]**
- We can also comment out this line to avoid this error.



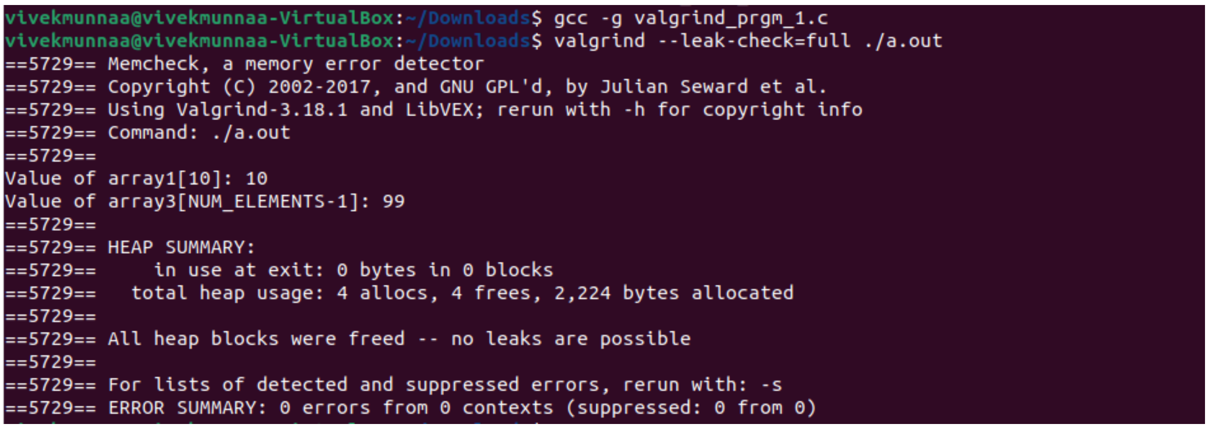
```
vivekmunnaa@vivekmunnaa-VirtualBox:~/Downloads$ valgrind --leak-check=full ./a.out
==5694== Memcheck, a memory error detector
==5694== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==5694== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==5694== Command: ./a.out
==5694==
==5694==
==5694== HEAP SUMMARY:
==5694==     in use at exit: 0 bytes in 0 blocks
==5694==   total heap usage: 1 allocs, 1 frees, 400 bytes allocated
==5694==
==5694== All heap blocks were freed -- no leaks are possible
==5694==
==5694== For lists of detected and suppressed errors, rerun with: -s
==5694== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
vivekmunnaa@vivekmunnaa-VirtualBox:~/Downloads$
```

> after this I tried executing valgrind for the entire program with all three arrays:



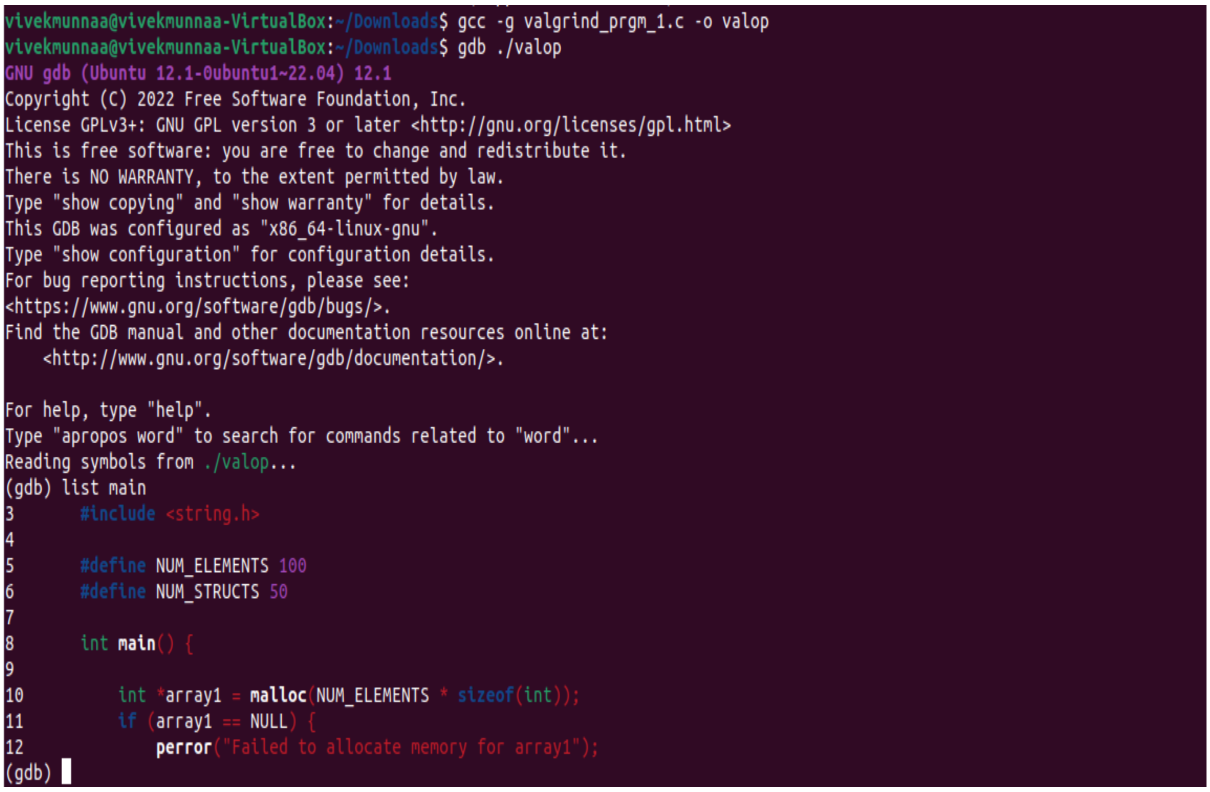
```c
#define NUM_ELEMENTS 100
#define NUM_STRUCTS 50

int main() {

    int *array1 = malloc(NUM_ELEMENTS * sizeof(int));
    if (array1 == NULL) {
        perror("Failed to allocate memory for array1");
        return 1;
    }
    for (int i = 0; i < NUM_ELEMENTS; i++) {
        array1[i] = i;
    }
   // free(array1); - here we are trying to free the memory before accessing, so we must remove this line and add it after print
    printf("Value of array1[10]: %d\n", array1[10]);
    free(array1);
    int *array2 = malloc(NUM_ELEMENTS * sizeof(int));
    if (array2 == NULL) {
        perror("Failed to allocate memory for array2");
        return 1;
    }
    for (int i = 0; i < NUM_ELEMENTS; i++) {
        array2[i] = i;
    }
    free(array2); //typo-array1 is changed to array2

    int *array3 = malloc(NUM_ELEMENTS * sizeof(int));
    if (array3 == NULL) {
        perror("Failed to allocate memory for array3");
        return 1;
    }
    for (int i = 0; i < NUM_ELEMENTS; i++) {
        array3[i] = i;
    }
   // the invalid read of size 4 in line 42 is due to accessing index 100 but we have only allocated memory till 99
    printf("Value of array3[NUM_ELEMENTS-1]: %d\n", array3[NUM_ELEMENTS-1]);//printing value NUM_ELEMENTS-1
    free(array3);
    return 0;
}
```

> **FINAL OUTPUT:**



```
vivekmunnaa@vivekmunnaa-VirtualBox:~/Downloads$ gcc -g valgrind_prgm_1.c
vivekmunnaa@vivekmunnaa-VirtualBox:~/Downloads$ valgrind --leak-check=full ./a.out
==5729== Memcheck, a memory error detector
==5729== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==5729== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==5729== Command: ./a.out
==5729==
Value of array1[10]: 10
Value of array3[NUM_ELEMENTS-1]: 99
==5729==
==5729== HEAP SUMMARY:
==5729==     in use at exit: 0 bytes in 0 blocks
==5729==   total heap usage: 4 allocs, 4 frees, 2,224 bytes allocated
==5729==
==5729== All heap blocks were freed -- no leaks are possible
==5729==
==5729== For lists of detected and suppressed errors, rerun with: -s
==5729== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

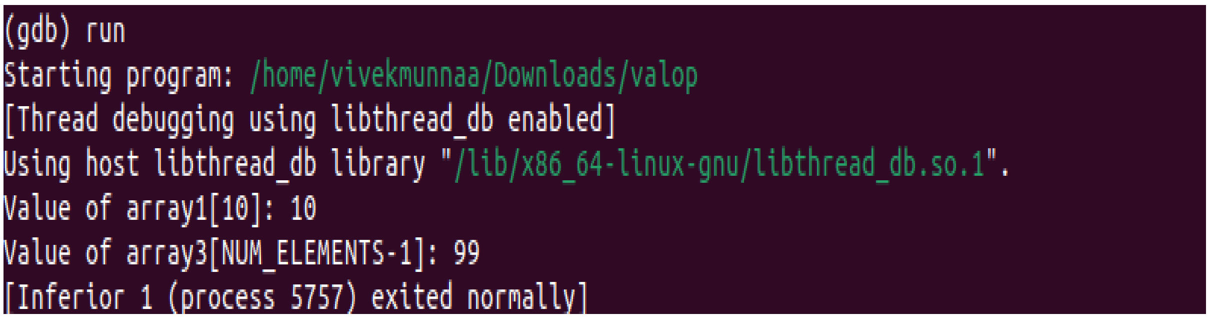
2.How do you list the source code of the main function in GDB?



```
vivekmunnaa@vivekmunnaa-VirtualBox:~/Downloads$ gcc -g valgrind_prgm_1.c -o valop
vivekmunnaa@vivekmunnaa-VirtualBox:~/Downloads$ gdb ./valop
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04) 12.1
Copyright (C) 2022 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:
<https://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 ./valop...
(gdb) list main
3       #include <string.h>
4
5       #define NUM_ELEMENTS 100
6       #define NUM_STRUCTS 50
7
8       int main() {
9
10          int *array1 = malloc(NUM_ELEMENTS * sizeof(int));
11          if (array1 == NULL) {
12              perror("Failed to allocate memory for array1");
(gdb) 
```

- **list main** is the command used.

3.How do you run the program inside GDB?

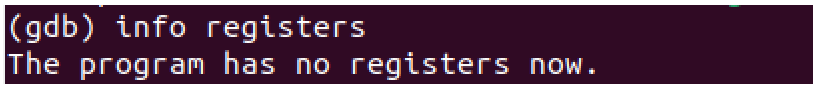


```
(gdb) run
Starting program: /home/vivekmunnaa/Downloads/valop
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Value of array1[10]: 10
Value of array3[NUM_ELEMENTS-1]: 99
[Inferior 1 (process 5757) exited normally]
```

- **run** is the command used.

(Note: The program runs normally as there is no breakpoints set yet and this was run after debugging errors with valgrind so there was no interruption and the program exited normally)

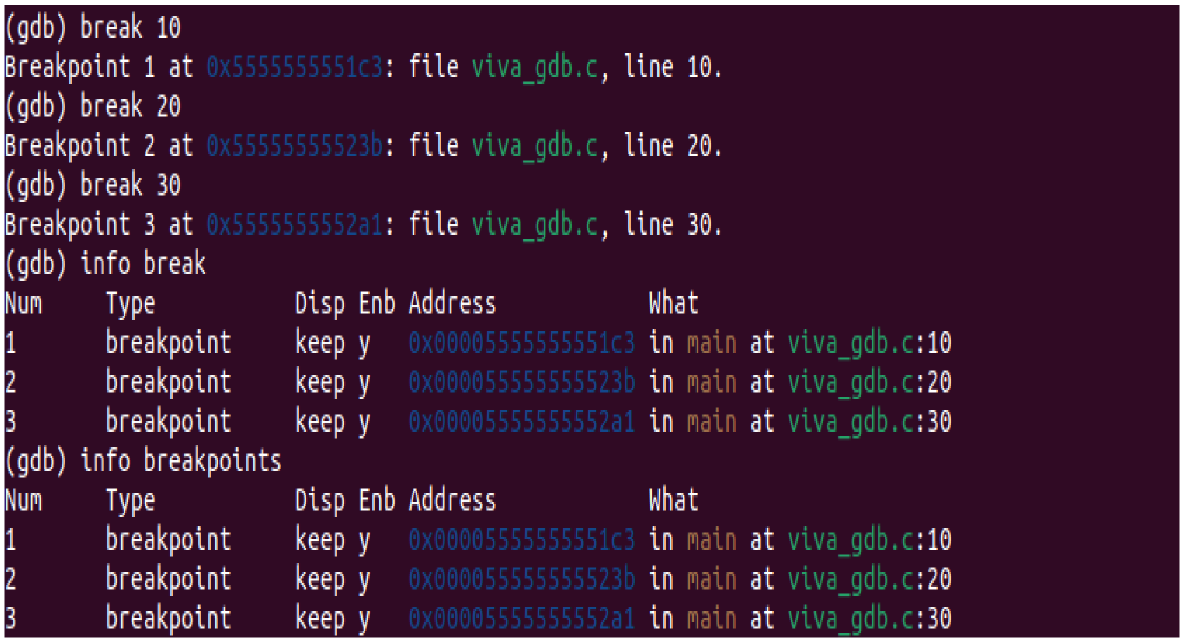
4.How do you display the current contents of the CPU registers in GDB?



```
(gdb) info registers
The program has no registers now.
```

- **info registers** is the command used.

5.How do you display information about all breakpoints in GDB?



```
(gdb) break 10
Breakpoint 1 at 0x5555555551c3: file viva_gdb.c, line 10.
(gdb) break 20
Breakpoint 2 at 0x55555555523b: file viva_gdb.c, line 20.
(gdb) break 30
Breakpoint 3 at 0x5555555552a1: file viva_gdb.c, line 30.
(gdb) info break
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0x00005555555551c3 in main at viva_gdb.c:10
2       breakpoint     keep y   0x000055555555523b in main at viva_gdb.c:20
3       breakpoint     keep y   0x00005555555552a1 in main at viva_gdb.c:30
(gdb) info breakpoints
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0x00005555555551c3 in main at viva_gdb.c:10
2       breakpoint     keep y   0x000055555555523b in main at viva_gdb.c:20
3       breakpoint     keep y   0x00005555555552a1 in main at viva_gdb.c:30
```