Daniyal Khan

3765942

CS2263

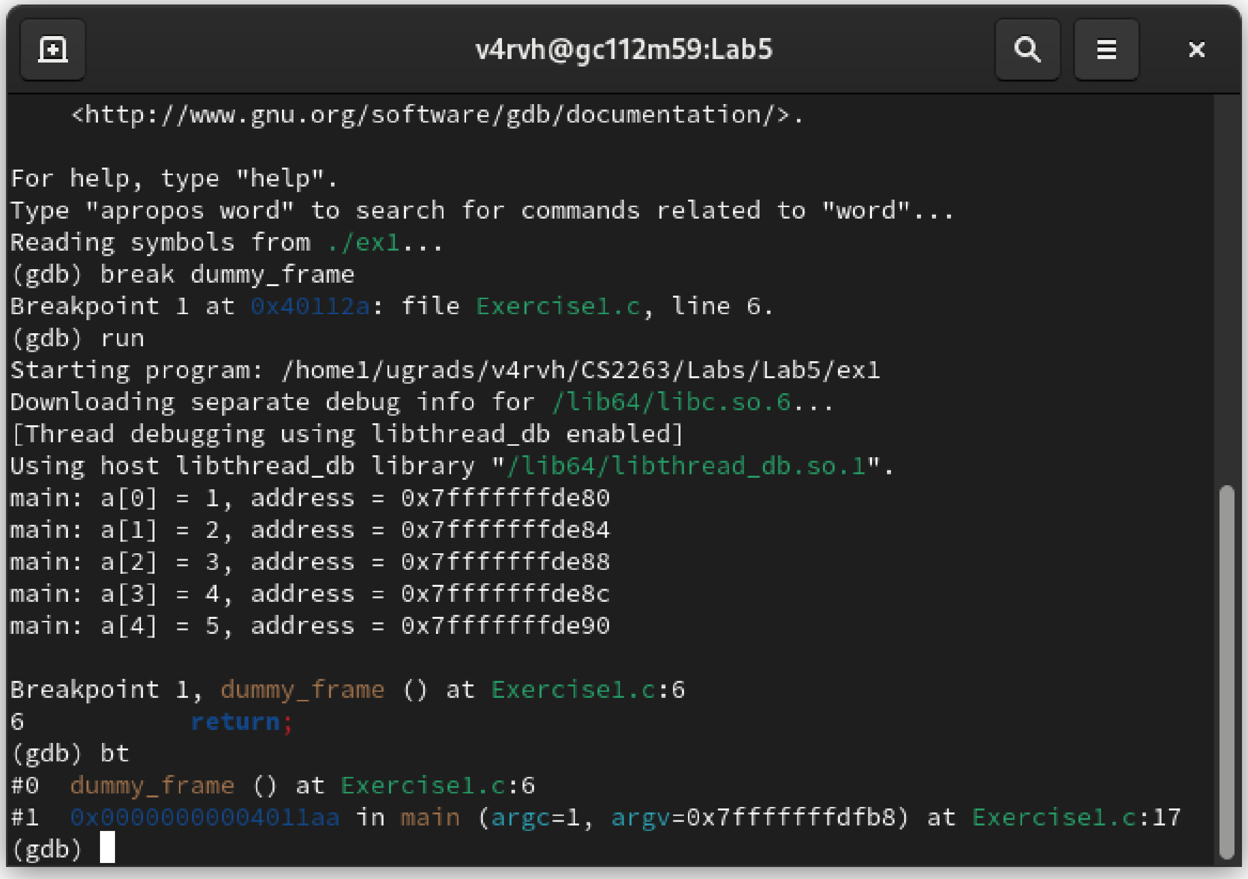
Lab 5

Exercise 1:

1.1:

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

void dummy_frame()
{
return;
}

int main(int argc, char **argv)
{
int i;
int a[] = {1, 2, 3, 4, 5};

for (i = 0; i < 5; i++)
printf("main: a[%d] = %d, address = %p\n", i, a[i], (void *)&a[i]);

dummy_frame();
return EXIT_SUCCESS;
}
```

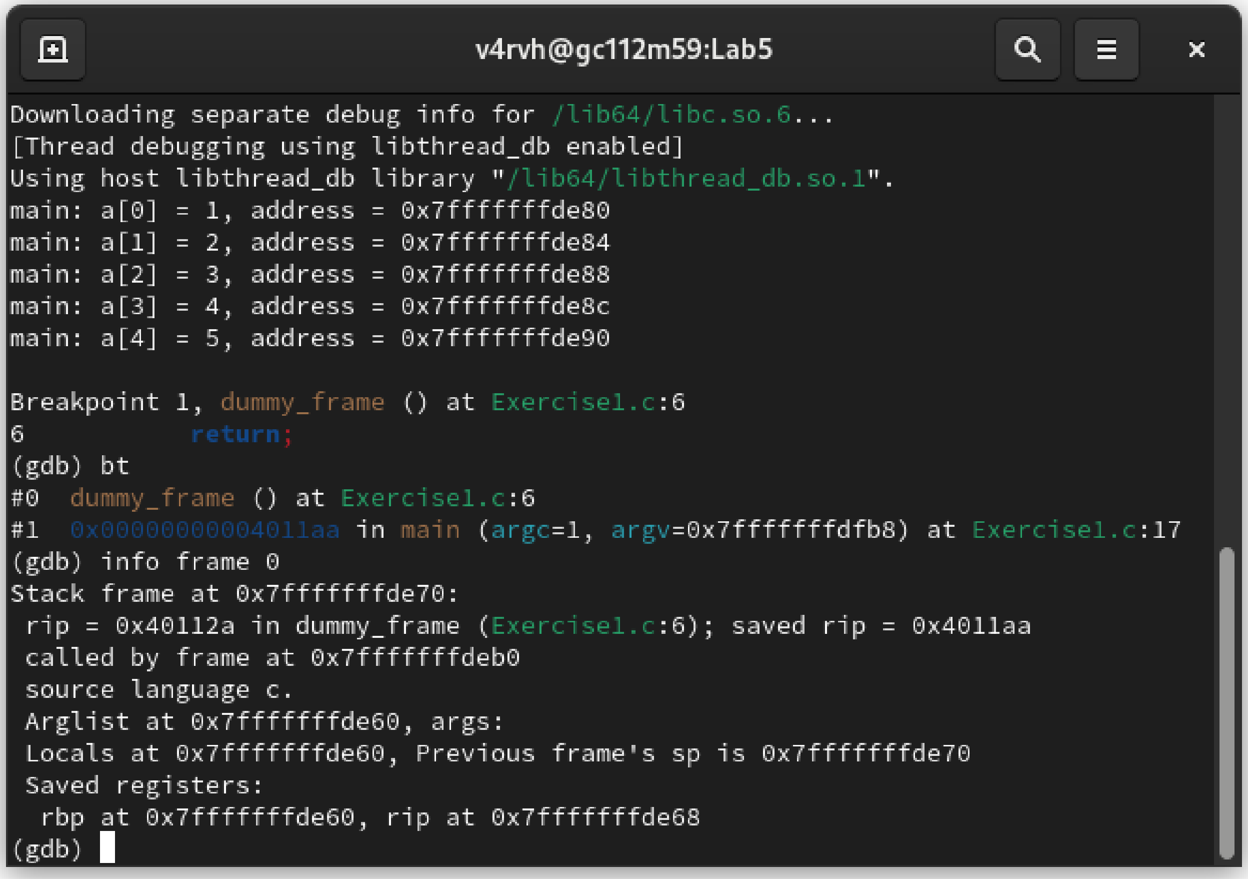
Backtrace:


v4rvh@gc112m59:Lab5

```
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./ex1...
(gdb) break dummy_frame
Breakpoint 1 at 0x40112a: file Exercise1.c, line 6.
(gdb) run
Starting program: /home1/ugrads/v4rvh/CS2263/Labs/Lab5/ex1
Downloading separate debug info for /lib64/libc.so.6...
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib64/libthread_db.so.1".
main: a[0] = 1, address = 0x7fffffffde80
main: a[1] = 2, address = 0x7fffffffde84
main: a[2] = 3, address = 0x7fffffffde88
main: a[3] = 4, address = 0x7fffffffde8c
main: a[4] = 5, address = 0x7fffffffde90

Breakpoint 1, dummy_frame () at Exercise1.c:6
6           return;
(gdb) bt
#0  dummy_frame () at Exercise1.c:6
#1  0x00000000004011aa in main (argc=1, argv=0x7fffffffdfb8) at Exercise1.c:17
(gdb) 
```

There are 2 frames on the memory stack.

Frame 0 info:



```
Downloading separate debug info for /lib64/libc.so.6...
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib64/libthread_db.so.1".
main: a[0] = 1, address = 0x7fffffffde80
main: a[1] = 2, address = 0x7fffffffde84
main: a[2] = 3, address = 0x7fffffffde88
main: a[3] = 4, address = 0x7fffffffde8c
main: a[4] = 5, address = 0x7fffffffde90

Breakpoint 1, dummy_frame () at Exercise1.c:6
6           return;
(gdb) bt
#0  dummy_frame () at Exercise1.c:6
#1  0x00000000004011aa in main (argc=1, argv=0x7fffffffdfb8) at Exercise1.c:17
(gdb) info frame 0
Stack frame at 0x7fffffffde70:
 rip = 0x40112a in dummy_frame (Exercise1.c:6); saved rip = 0x4011aa
 called by frame at 0x7fffffffdeb0
 source language c.
 Arglist at 0x7fffffffde60, args:
 Locals at 0x7fffffffde60, Previous frame's sp is 0x7fffffffde70
 Saved registers:
  rbp at 0x7fffffffde60, rip at 0x7fffffffde68
(gdb) 
```

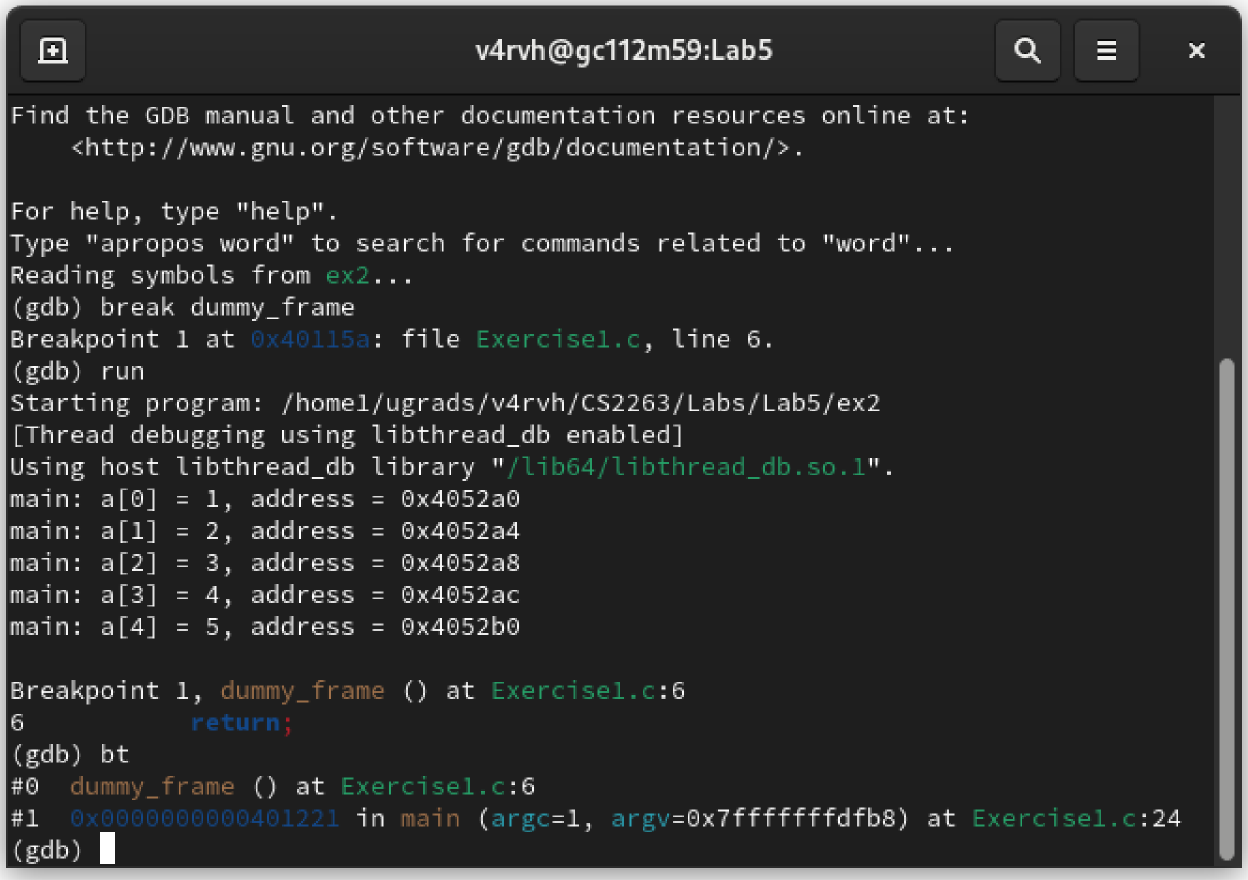
main()'s stack frame lies above dummy_frame()'s (since stack grows downward).
So the boundaries for main() are from:
0x7fffffffde70 (top of main frame)
up to the called by frame at (0x7fffffffdeb0)

Looking at the addresses of the array elements, they do fall inside the range of the main function frame as in the screenshot.

1.2:

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

void dummy_frame()
{
return;
}

int main(int argc, char **argv)
{
int i;
int *a = (int *)malloc(5 * sizeof(int)); // Heap allocation

if (a == NULL) {
fprintf(stderr, "Memory allocation failed\n");
return EXIT_FAILURE;
}

for (i = 0; i < 5; i++) {
a[i] = i + 1;
printf("main: a[%d] = %d, address = %p\n", i, a[i], (void *)&a[i]);
}

dummy_frame();

free(a); // Clean up
return EXIT_SUCCESS;
}
```

Backtrace:

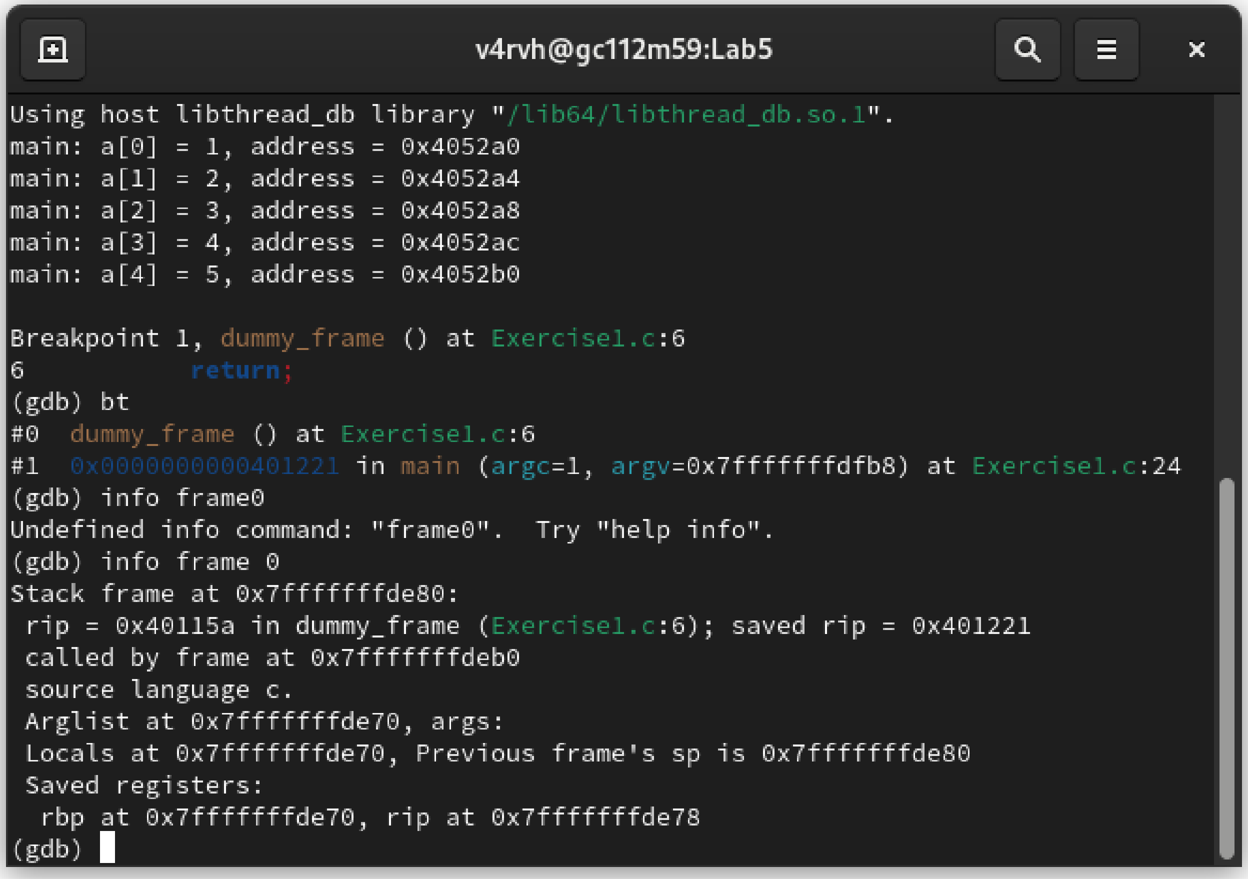


v4rvh@gc112m59:Lab5

```
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 ex2...
(gdb) break dummy_frame
Breakpoint 1 at 0x40115a: file Exercise1.c, line 6.
(gdb) run
Starting program: /home1/ugrads/v4rvh/CS2263/Labs/Lab5/ex2
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib64/libthread_db.so.1".
main: a[0] = 1, address = 0x4052a0
main: a[1] = 2, address = 0x4052a4
main: a[2] = 3, address = 0x4052a8
main: a[3] = 4, address = 0x4052ac
main: a[4] = 5, address = 0x4052b0

Breakpoint 1, dummy_frame () at Exercise1.c:6
6           return;
(gdb) bt
#0  dummy_frame () at Exercise1.c:6
#1  0x0000000000401221 in main (argc=1, argv=0x7fffffffdfb8) at Exercise1.c:24
(gdb)
```

There are 2 frames on the memory stack.

Info of frame 0:



```
Using host libthread_db library "/lib64/libthread_db.so.1".
main: a[0] = 1, address = 0x4052a0
main: a[1] = 2, address = 0x4052a4
main: a[2] = 3, address = 0x4052a8
main: a[3] = 4, address = 0x4052ac
main: a[4] = 5, address = 0x4052b0

Breakpoint 1, dummy_frame () at Exercise1.c:6
6           return;
(gdb) bt
#0  dummy_frame () at Exercise1.c:6
#1  0x0000000000401221 in main (argc=1, argv=0x7fffffffdfb8) at Exercise1.c:24
(gdb) info frame0
Undefined info command: "frame0".  Try "help info".
(gdb) info frame 0
Stack frame at 0x7fffffffde80:
 rip = 0x40115a in dummy_frame (Exercise1.c:6); saved rip = 0x401221
 called by frame at 0x7fffffffdeb0
 source language c.
 Arglist at 0x7fffffffde70, args:
 Locals at 0x7fffffffde70, Previous frame's sp is 0x7fffffffde80
 Saved registers:
  rbp at 0x7fffffffde70, rip at 0x7fffffffde78
(gdb)
```

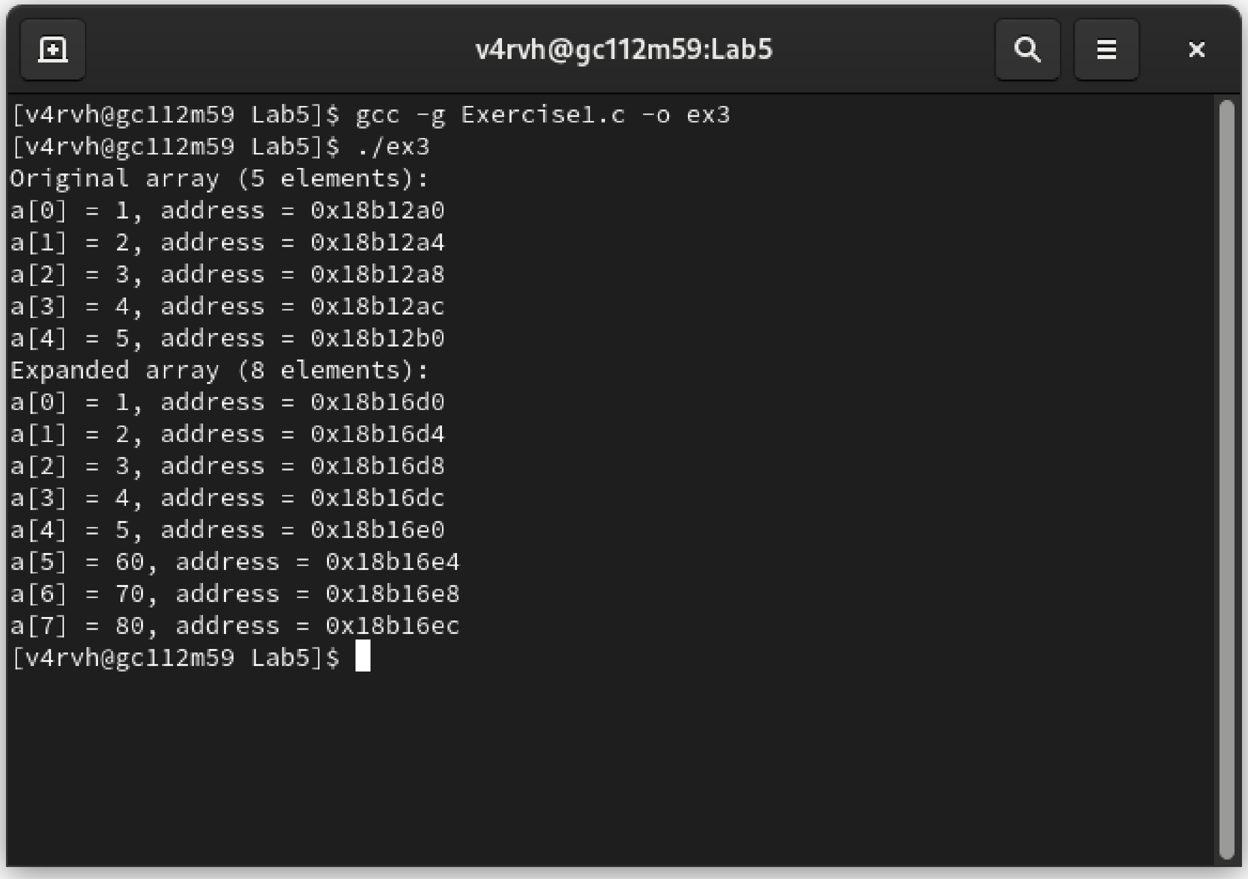
This means main()'s frame lies between 0x7fffffffde80 and 0x7fffffffdeb0.

No the addresses of the array elements are not within the stack frame of main(). They are located in the heap, as they were allocated using malloc().

Exercise 2:

2.1:

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

void dummy_frame() {
return;
}

int main(int argc, char **argv) {
int i;
int *a;

a = (int *)malloc(5 * sizeof(int));
if (a == NULL) {
fprintf(stderr, "Initial malloc failed.\n");
return EXIT_FAILURE;
}

printf("Original array (5 elements):\n");
for (i = 0; i < 5; i++) {
a[i] = i + 1;
printf("a[%d] = %d, address = %p\n", i, a[i], (void *)&a[i]);
}

int *temp = (int *)realloc(a, 8 * sizeof(int));
if (temp == NULL) {
fprintf(stderr, "Realloc failed.\n");
free(a); // Free original memory if realloc fails
return EXIT_FAILURE;
}
a = temp;


for (i = 5; i < 8; i++) {
a[i] = (i + 1) * 10;
}

printf("Original array (5 elements):\n");
for (i = 0; i < 8; i++) {
printf("a[%d] = %d, address = %p\n", i, a[i], (void *)&a[i]);
}

dummy_frame();
free(a);
return EXIT_SUCCESS;
}
```



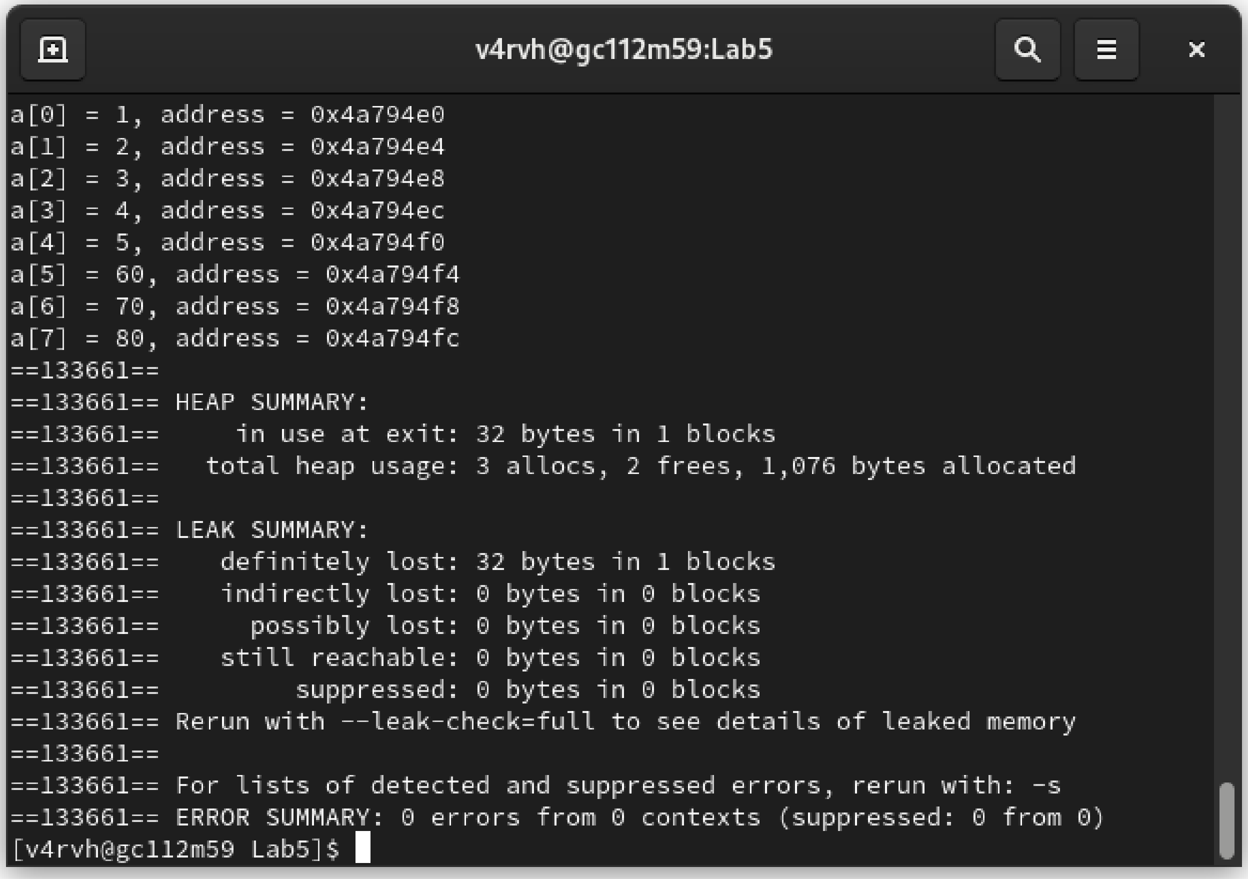
```
[v4rvh@gc112m59 Lab5]$ gcc -g Exercise1.c -o ex3
[v4rvh@gc112m59 Lab5]$ ./ex3
Original array (5 elements):
a[0] = 1, address = 0x18b12a0
a[1] = 2, address = 0x18b12a4
a[2] = 3, address = 0x18b12a8
a[3] = 4, address = 0x18b12ac
a[4] = 5, address = 0x18b12b0
Expanded array (8 elements):
a[0] = 1, address = 0x18b16d0
a[1] = 2, address = 0x18b16d4
a[2] = 3, address = 0x18b16d8
a[3] = 4, address = 0x18b16dc
a[4] = 5, address = 0x18b16e0
a[5] = 60, address = 0x18b16e4
a[6] = 70, address = 0x18b16e8
a[7] = 80, address = 0x18b16ec
[v4rvh@gc112m59 Lab5]$
```

As in the terminal screenshot the address changed, the array was moved in memory

Exercise 3:

3.1:

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

void dummy_frame() {
return;
}

int main(int argc, char **argv) {
int i;
int *a;

a = (int *)malloc(5 * sizeof(int));
if (a == NULL) {
fprintf(stderr, "Initial malloc failed.\n");
return EXIT_FAILURE;
}

printf("Original array (5 elements):\n");
for (i = 0; i < 5; i++) {
a[i] = i + 1;
printf("a[%d] = %d, address = %p\n", i, a[i], (void *)&a[i]);
}

int *temp = (int *)realloc(a, 8 * sizeof(int));
if (temp == NULL) {
fprintf(stderr, "Realloc failed.\n");
free(a); // Free original memory if realloc fails
return EXIT_FAILURE;
}
a = temp;


for (i = 5; i < 8; i++) {
a[i] = (i + 1) * 10;
}

printf("Expanded array (8 elements):\n");
for (i = 0; i < 8; i++) {
printf("a[%d] = %d, address = %p\n", i, a[i], (void *)&a[i]);
}
dummy_frame();
return EXIT_SUCCESS;
}
```

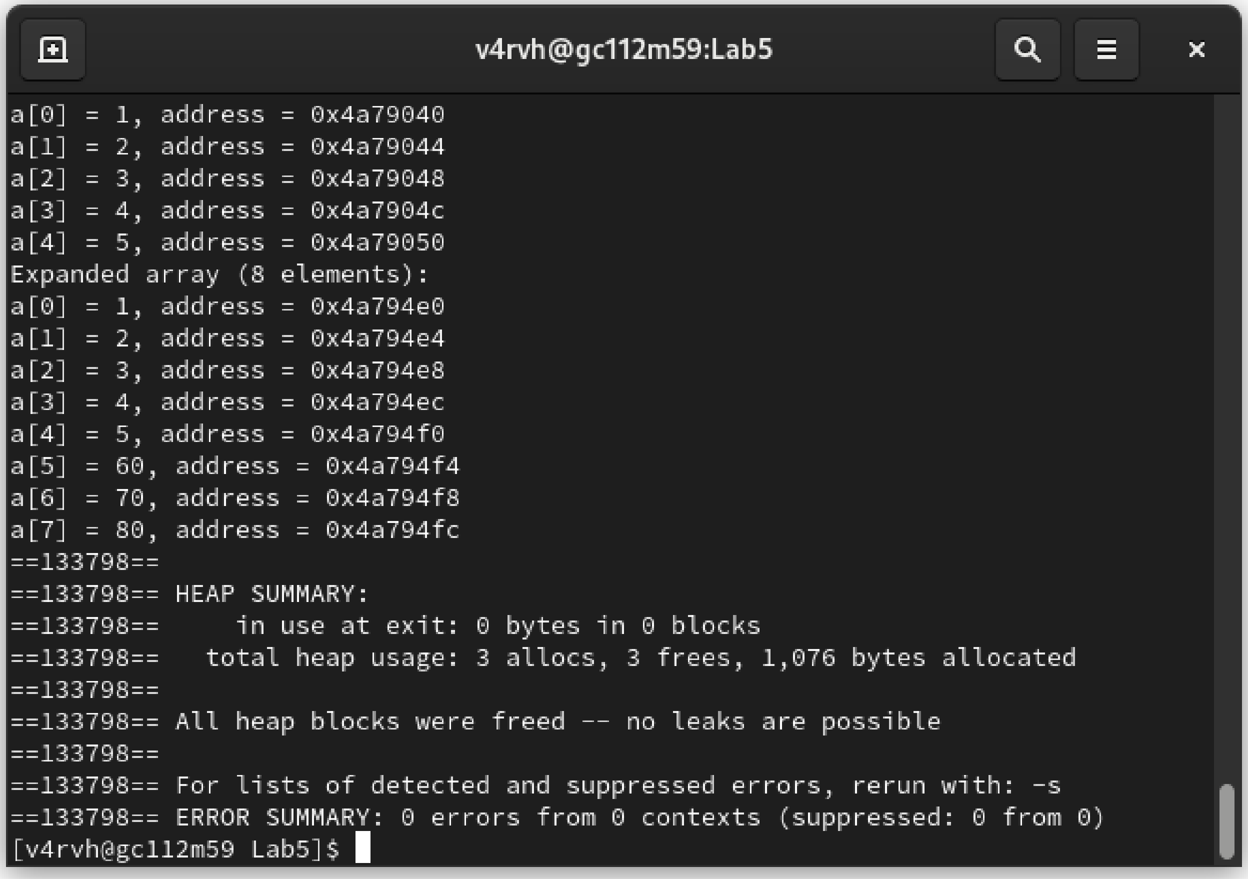

v4rvh@gc112m59:Lab5

```
a[0] = 1, address = 0x4a794e0
a[1] = 2, address = 0x4a794e4
a[2] = 3, address = 0x4a794e8
a[3] = 4, address = 0x4a794ec
a[4] = 5, address = 0x4a794f0
a[5] = 60, address = 0x4a794f4
a[6] = 70, address = 0x4a794f8
a[7] = 80, address = 0x4a794fc
==133661==
==133661== HEAP SUMMARY:
==133661==     in use at exit: 32 bytes in 1 blocks
==133661==   total heap usage: 3 allocs, 2 frees, 1,076 bytes allocated
==133661==
==133661== LEAK SUMMARY:
==133661==    definitely lost: 32 bytes in 1 blocks
==133661==    indirectly lost: 0 bytes in 0 blocks
==133661==      possibly lost: 0 bytes in 0 blocks
==133661==    still reachable: 0 bytes in 0 blocks
==133661==         suppressed: 0 bytes in 0 blocks
==133661== Rerun with --leak-check=full to see details of leaked memory
==133661==
==133661== For lists of detected and suppressed errors, rerun with: -s
==133661== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
[v4rvh@gc112m59 Lab5]$
```

There is a memory leak as in the screenshot.

3.2:

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

void dummy_frame() {
return;
}

int main(int argc, char **argv) {
int i;
int *a;

a = (int *)malloc(5 * sizeof(int));
if (a == NULL) {
fprintf(stderr, "Initial malloc failed.\n");
return EXIT_FAILURE;
}

printf("Original array (5 elements):\n");
for (i = 0; i < 5; i++) {
a[i] = i + 1;
printf("a[%d] = %d, address = %p\n", i, a[i], (void *)&a[i]);
}

int *temp = (int *)realloc(a, 8 * sizeof(int));
if (temp == NULL) {
fprintf(stderr, "Realloc failed.\n");
free(a); // Free original memory if realloc fails
return EXIT_FAILURE;
}
a = temp;


for (i = 5; i < 8; i++) {
a[i] = (i + 1) * 10;
}

printf("Expanded array (8 elements):\n");
for (i = 0; i < 8; i++) {
printf("a[%d] = %d, address = %p\n", i, a[i], (void *)&a[i]);
}

dummy_frame();
free(a);
return EXIT_SUCCESS;
}
```



```
a[0] = 1, address = 0x4a79040
a[1] = 2, address = 0x4a79044
a[2] = 3, address = 0x4a79048
a[3] = 4, address = 0x4a7904c
a[4] = 5, address = 0x4a79050
Expanded array (8 elements):
a[0] = 1, address = 0x4a794e0
a[1] = 2, address = 0x4a794e4
a[2] = 3, address = 0x4a794e8
a[3] = 4, address = 0x4a794ec
a[4] = 5, address = 0x4a794f0
a[5] = 60, address = 0x4a794f4
a[6] = 70, address = 0x4a794f8
a[7] = 80, address = 0x4a794fc
==133798==
==133798== HEAP SUMMARY:
==133798==     in use at exit: 0 bytes in 0 blocks
==133798==   total heap usage: 3 allocs, 3 frees, 1,076 bytes allocated
==133798==
==133798== All heap blocks were freed -- no leaks are possible
==133798==
==133798== For lists of detected and suppressed errors, rerun with: -s
==133798== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
[v4rvh@gc112m59 Lab5]$
```