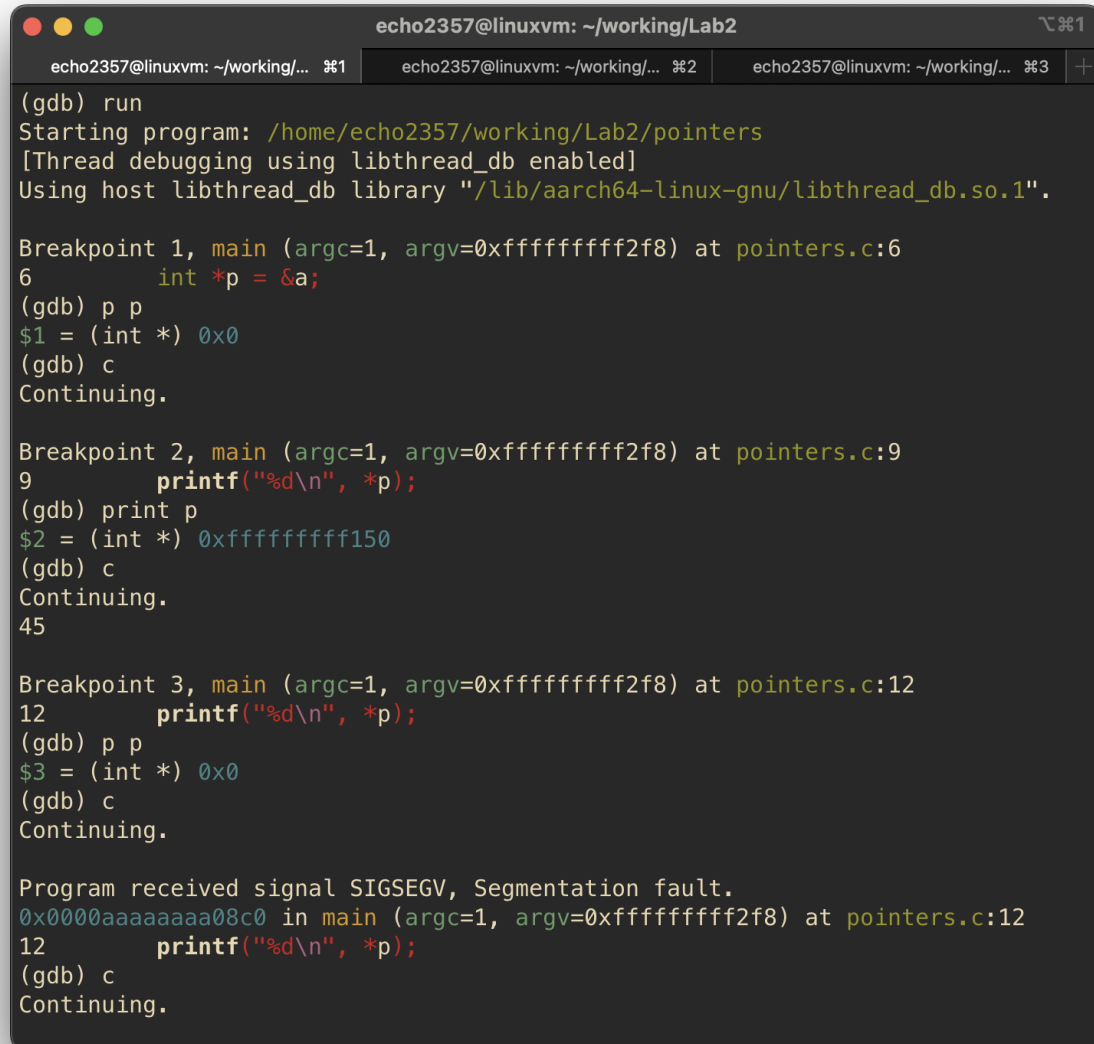


Lab 02

Echo Wang A01347203

Question 1

Line 12 - A segmentation fault happens when we try to read from a null pointer (0x0)



```
echo2357@linuxvm: ~/working/Lab2
echo2357@linuxvm: ~/working/... %1 | echo2357@linuxvm: ~/working/... %2 | echo2357@linuxvm: ~/working/... %3 | +
(gdb) run
Starting program: /home/echo2357/working/Lab2/pointers
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/aarch64-linux-gnu/libthread_db.so.1".

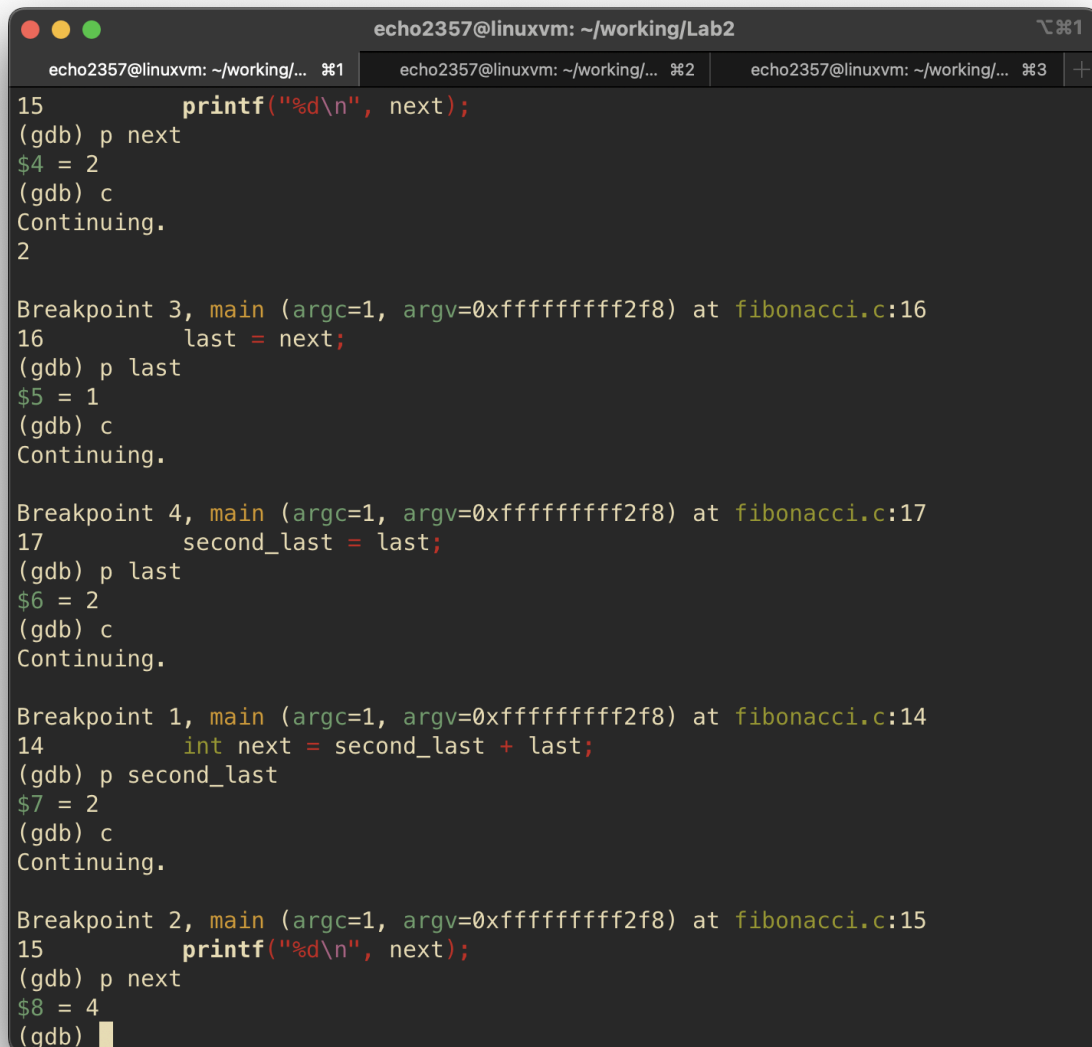
Breakpoint 1, main (argc=1, argv=0xffffffff2f8) at pointers.c:6
6      int *p = &a;
(gdb) p p
$1 = (int *) 0x0
(gdb) c
Continuing.

Breakpoint 2, main (argc=1, argv=0xffffffff2f8) at pointers.c:9
9      printf("%d\n", *p);
(gdb) print p
$2 = (int *) 0xffffffff150
(gdb) c
Continuing.
45

Breakpoint 3, main (argc=1, argv=0xffffffff2f8) at pointers.c:12
12     printf("%d\n", *p);
(gdb) p p
$3 = (int *) 0x0
(gdb) c
Continuing.

Program received signal SIGSEGV, Segmentation fault.
0x0000aaaaaaa08c0 in main (argc=1, argv=0xffffffff2f8) at pointers.c:12
12     printf("%d\n", *p);
(gdb) c
Continuing.
```

Question 2



```
echo2357@linuxvm: ~/working/Lab2
echo2357@linuxvm: ~/working/... %1  echo2357@linuxvm: ~/working/... %2  echo2357@linuxvm: ~/working/... %3  +
15      printf("%d\n", next);
(gdb) p next
$4 = 2
(gdb) c
Continuing.
2

Breakpoint 3, main (argc=1, argv=0xffffffff2f8) at fibonacci.c:16
16      last = next;
(gdb) p last
$5 = 1
(gdb) c
Continuing.

Breakpoint 4, main (argc=1, argv=0xffffffff2f8) at fibonacci.c:17
17      second_last = last;
(gdb) p last
$6 = 2
(gdb) c
Continuing.

Breakpoint 1, main (argc=1, argv=0xffffffff2f8) at fibonacci.c:14
14      int next = second_last + last;
(gdb) p second_last
$7 = 2
(gdb) c
Continuing.

Breakpoint 2, main (argc=1, argv=0xffffffff2f8) at fibonacci.c:15
15      printf("%d\n", next);
(gdb) p next
$8 = 4
(gdb) █
```

The 2 variables: `last` and `second_last` is set to the same value of `next`, in this iteration, `2`, which is not what we want

We should switch line 16 and 17 to make the logic correct

Also, initializing `second_last` to 0 would make the program more compliant to the definition of fibonacci numbers, as mentioned in the lab :p

Question 3

```
echo2357@linuxvm: ~/working/Lab2 — 133x58
echo2357@linuxvm: ~/working/Lab2 (ssh) %1 echo2357@linuxvm: ~/working/Lab2 (ssh) %2 echo2357@linuxvm: ~/working/Lab2 (ssh) %3 +
echo2357@linuxvm:~/working/Lab2$ valgrind --tool=memcheck --leak-check=yes --show-reachable=yes --num-callers=20 ./memory_bugs
==1493== Memcheck, a memory error detector
==1493== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==1493== Using Valgrind-3.22.0 and LibVEX; rerun with -h for copyright info
==1493== Command: ./memory_bugs
==1493==
==1493== Syscall param write(buf) points to uninitialised byte(s)
==1493==    at 0x49A1D48: write (write.c:26)
==1493==    by 0x1089EB: main (memory_bugs.c:19)
==1493==    Address 0x1fff000b0 is on thread 1's stack
==1493==    in frame #1, created by main (memory_bugs.c:9)
==1493==
y==1493== Invalid write of size 1
==1493==    at 0x108A08: main (memory_bugs.c:26)
==1493==    Address 0x4a7e0a0 is 0 bytes inside a block of size 12 free'd
==1493==    at 0x48882A8: free (in /usr/libexec/valgrind/vgpreload_memcheck-arm64-linux.so)
==1493==    by 0x1089FF: main (memory_bugs.c:23)
==1493==    Block was alloc'd at
==1493==    at 0x4885250: malloc (in /usr/libexec/valgrind/vgpreload_memcheck-arm64-linux.so)
==1493==    by 0x1089F3: main (memory_bugs.c:22)
==1493==
==1493== Invalid read of size 1
==1493==    at 0x108A10: main (memory_bugs.c:29)
==1493==    Address 0x4a7e0a0 is 0 bytes inside a block of size 12 free'd
==1493==    at 0x48882A8: free (in /usr/libexec/valgrind/vgpreload_memcheck-arm64-linux.so)
==1493==    by 0x1089FF: main (memory_bugs.c:23)
==1493==    Block was alloc'd at
==1493==    at 0x4885250: malloc (in /usr/libexec/valgrind/vgpreload_memcheck-arm64-linux.so)
==1493==    by 0x1089F3: main (memory_bugs.c:22)
==1493==
A
==1493== Invalid free() / delete / delete[] / realloc()
==1493==    at 0x48882A8: free (in /usr/libexec/valgrind/vgpreload_memcheck-arm64-linux.so)
==1493==    by 0x108A37: main (memory_bugs.c:35)
==1493==    Address 0x1fff000b0 is on thread 1's stack
==1493==    in frame #1, created by main (memory_bugs.c:9)
==1493==
==1493==
==1493== HEAP SUMMARY:
==1493==    in use at exit: 80 bytes in 2 blocks
==1493==    total heap usage: 4 allocs, 3 frees, 1,116 bytes allocated
==1493==
==1493== 30 bytes in 1 blocks are definitely lost in loss record 1 of 2
==1493==    at 0x4885250: malloc (in /usr/libexec/valgrind/vgpreload_memcheck-arm64-linux.so)
==1493==    by 0x1089BF: main (memory_bugs.c:16)
==1493==
==1493== 50 bytes in 1 blocks are definitely lost in loss record 2 of 2
==1493==    at 0x4885250: malloc (in /usr/libexec/valgrind/vgpreload_memcheck-arm64-linux.so)
==1493==    by 0x108A2B: main (memory_bugs.c:32)
==1493==
==1493== LEAK SUMMARY:
==1493==    definitely lost: 80 bytes in 2 blocks
==1493==    indirectly lost: 0 bytes in 0 blocks
==1493==    possibly lost: 0 bytes in 0 blocks
==1493==    still reachable: 0 bytes in 0 blocks
==1493==    suppressed: 0 bytes in 0 blocks
==1493==
==1493== Use --track-origins=yes to see where uninitialised values come from
```

1. Uninitialized Memory Use at line 16 `write(filenno(stdout), arr, 10)` .The variable `arr` is declared as `int arr[10]` and there's no initialization, so we're not supposed to read from it
2. Invalid Write at line 26 `p = 'A'`, because `*p` has already been freed at line 22
3. Invalid Read at line 29 `printf("%c\n", *p)` because of the same reason above:`*p` has already been freed at line 22
4. Invalid free of `arr` at line 35, because the memory location is statically allocated in the stack with `int arr[10]`, so it cannot be freed
5. Memory leaks at line 16 and line 32: Memory is allocated with `malloc()` calls, but never used nor freed.