Interlude : Memory API

① `int *ptr = NULL;`

`printf("value: %d\n", *ptr)`

`gcc -o null null.c` ⇒ compile

② `gcc -g -o null null.c`

includes debug symbols in the executable, 'gdb' tool can use

gdb not supported for mac arm processor. Using lldb

→ `lldb null`

→ `run`

```
Process 86046 stopped
* thread #1, queue = 'com.apple.main-thread', stop reason = EXC_BAD_ACCESS (code=1, address=0x0)
    frame #0: 0x0000000100003f2c null`main at null.c:11:25
   8
   9       // Attempt to dereference the NULL pointer
   10      printf("Dereferencing NULL pointer...\n");
-> 11      printf("Value: %d\n", *ptr); // This will cause a segmentation fault
   12
   13      return 0;
   14  }
Target 0: (null) stopped.
```

③ Valgrind again not supported for macOS arm yet.

Mine "leaks" tool

No leaks detected, rightly since null pointer deference no allocated memory

```
null(91653) MallocStackLogging: could not tag MSL-related memory as no_footprint, so those
null(91653) MallocStackLogging: recording malloc (and VM allocation) stacks using lite mod
Pointer is set to NULL.
Dereferencing NULL pointer...
```

④ Udb - no error

```
Process 91528 exited with status = 0 (0x00000000)
```

leaks --atExit -- ./leaky_memory

```
STACK OF 9 INSTANCES OF 'ROOT LEAK: <malloc in leak_memory>':
3   dyld                                0x191650274 start + 2840
2   leaky_program                       0x102033f78 main + 48
1   leaky_program                       0x102033f38 leak_memory + 20
0   libsystem_malloc.dylib              0x19180a7cc _malloc_zone_malloc_instrumented_or_legacy + 148
====
    9 (144 bytes) << TOTAL >>
      1 (16 bytes) ROOT LEAK: <malloc in leak_memory 0x12df040f0> [16]
      1 (16 bytes) ROOT LEAK: <malloc in leak_memory 0x12df04100> [16]
      1 (16 bytes) ROOT LEAK: <malloc in leak_memory 0x12df04110> [16]
      1 (16 bytes) ROOT LEAK: <malloc in leak_memory 0x12df04120> [16]
      1 (16 bytes) ROOT LEAK: <malloc in leak_memory 0x12df04130> [16]
      1 (16 bytes) ROOT LEAK: <malloc in leak_memory 0x12df04140> [16]
      1 (16 bytes) ROOT LEAK: <malloc in leak_memory 0x12df04150> [16]
      1 (16 bytes) ROOT LEAK: <malloc in leak_memory 0x12df04160> [16]
      1 (16 bytes) ROOT LEAK: <malloc in leak_memory 0x12df04170> [16]
```

```
Binary Images:
       0x102030000 -        0x102033fff +leaky_program (0) <DBF997CC-4806-37EA-AEE3-6ADEEC0725F9> /Users/*/leaky_program
       0x10205c000 -        0x10205cffb  libLeaksAtExit.dylib (64566.82.1) <B2BBF4F8-1A34-3754-B74D-672A05F9A387> /usr/lib/libLeaksAtExit.dylib
       0x1915f8000 -        0x191649ca3  libobjc.A.dylib (928.3) <219C0324-7A4F-338E-A395-3D023B289A7F> /usr/lib/libobjc.A.dylib
```

With single malloc on macOS shows 0 leak. Will try with valgrind on linux.

(5) Program is incorrect, it is accessing index not there, should give out-of-bounds error But C program doesn't throw any error.

```
./array-out

data[100] = 0
```

```
leaks Report Version: 4.0, multi-line stacks
Process 98120: 184 nodes malloced for 15 KB
Process 98120: 1 leak for 416 total leaked bytes.

STACK OF 1 INSTANCE OF 'ROOT LEAK: <malloc in main>':
2   dyld                                0x191650274 start + 2840
1   array-out                           0x10041bef4 main + 24  array-out.c:6
0   libsystem_malloc.dylib              0x19180a7cc _malloc_zone_malloc_instrumented_or_legacy + 148
====
    1 (416 bytes) ROOT LEAK: <malloc in main 0x11d7040f0> [416]



Binary Images:
       0x100418000 -        0x10041bfff +array-out (0) <9CA2DECD-8DDF-3676-9B6B-A5EDB2300B4B> /Users/*/array-out
       0x100444000 -        0x100444ffb  libLeaksAtExit.dylib (64566.82.1) <B2BBF4F8-1A34-3754-B74D-672A05F9A387> /usr/lib/libLeaksAtExit.dylib
       0x1915f8000 -        0x191649ca3  libobjc.A.dylib (928.3) <219C0324-7A4F-338E-A395-3D023B289A7F> /usr/lib/libobjc.A.dylib
       0x19164a000 -        0x1916cc7b3  dyld (1.0.0 - 1235.2) <6BEAFDE4-B011-3E47-8AAE-4D7B6E4BB7E8> /usr/lib/dyld
```

(6) Program runs, prints 0 value for it. Again no errors. This time memory is freed so no leaks.
But obviously the program is incorrect.

```
./array-free-access
data[0] = 0
```

```
leaks Report Version: 4.0, multi-line stacks
Process 99435: 183 nodes malloced for 14 KB
Process 99435: 0 leaks for 0 total leaked bytes.
```

(7) No need of any tool, this time no execution the program throws an error.
In this one we are running free with pointer to some middle element of array.

```
./array-free-middle
array-free-middle(1035,0x1f6ae7840) malloc: *** error for object 0x121e04828: pointer being freed was not allocated
array-free-middle(1035,0x1f6ae7840) malloc: *** set a breakpoint in malloc_error_break to debug
[1]    1035 abort      ./array-free-middle
```

(8) Realloc() for vector struct

Compared to linked-list
- Access is O(1) instead of O(n) in linked-list where you need to traverse the list.
- Insertion O(1) for both, since

for vector realloc happens not always since we allocate double of current capacity when it runs out.
For linked-list new node at the end.

- Vector allocates contiguous memory.
  May waste space if vector becomes smaller than the allocated capacity.
  Linked-list no waste since memory is allocated per-node.
  Uses more memory per node, stores a pointer to the next node.

No leaks (If we free memory)

```
leaks Report Version: 4.0, multi-line stacks
Process 4301: 183 nodes malloced for 14 KB
Process 4301: 0 leaks for 0 total leaked bytes.
```

```
leaks Report Version: 4.0, multi-line stacks
Process 4647: 184 nodes malloced for 14 KB
Process 4647: 1 leak for 80 total leaked bytes.

STACK OF 1 INSTANCE OF 'ROOT LEAK: <realloc in vector_add>':
4   dyld                                0x191650274 start + 2840
3   vector-realloc                      0x100723f04 main + 64  vector-realloc.c:61
2   vector-realloc                      0x100723d70 vector_add + 136  vector-realloc.c:23
1   libsystem_malloc.dylib              0x19180b4dc _realloc + 468
0   libsystem_malloc.dylib              0x19180acc0 _malloc_zone_realloc + 144
====
    1 (80 bytes) ROOT LEAK: <realloc in vector_add 0x13c7040f0> [80]



Binary Images:
       0x100720000 -        0x100723fff +vector-realloc (0) <6146970E-2653-3B02-A608-0C14E2976A6D> /Users/*/vector-realloc
       0x10074c000 -        0x10074cffb  libLeaksAtExit.dylib (64566.82.1) <B2BBF4F8-1A34-3754-B74D-672A05F9A387> /usr/lib/libLeaksAtExit.dylib
       0x1915f8000 -        0x191649ca3  libobjc.A.dylib (928.3) <219C0324-7A4F-338E-A395-3D023B289A7F> /usr/lib/libobjc.A.dylib
```

(9) If you observe the value of leaks tool output, it shows values like 16 bytes, 80 bytes, 416 bytes those are the nearest multiple for 16 byte.

Since most modern memory allocators align with specific boundary, its 16 bytes on macOS.

Minimum block size = 16 bytes