15-122 Bootcamp: Valgrind

Summer 2023

Today's Agenda

_01

What is Valgrind?

_02

Valgrind Exercises

materials

tinyurl.com/n23valgrind

_01
What is Valgrind?

materials

tinyurl.com/n23valgrind

Valgrind Basics

- Super powerful set of tools to check C and C++ programs for errors
- We use the Valgrind to help us with memory errors **and** memory leaks in C
- Valgrind output gets scary — always start at **the first error!**
- For more information on leaks: `valgrind --leak-check=full ./a.out`
- Compiling your file: use `-g` flag for line numbers

materials

tinyurl.com/n23valgrind

```
% gcc -g myfile.c
% valgrind --leak-check=full ./a.out
```

Common Errors: Invalid Read

- **Invalid read**: accessing memory that was not allocated
- Usually off-by-one array indices or pointer arithmetic with wrong types

```
Invalid read of size 4
   at 0x4005C6: f2 (example_file.c:14)
   by 0x4005FE: main (example_file.c:21)
Address 0x5205048 is 4 bytes after a block of size 4 alloc'd
   at 0x4C29F73: malloc (vg_replace_malloc.c:309)
   by 0x40058E: f1 (example_file.c:4)
   by 0x4005B4: f2 (example_file.c:11)
   by 0x4005F#: main (example_file.c:21)
```

```
1  #include <stdlib.h>
2
3  int *f1() {
4      int *ip = malloc(sizeof(int));
5
6      *ip = 3;
7      return ip;
8  }
9
10 int f2() {
11     int *internal = f1();
12
13     int left = internal[0];
14     int right = internal[2];
15     free(internal);
16
17     return left + right / 2;
18 }
19
20 int main() {
21     int i = f2();
22     return i;
23 }
```

Common Errors: Invalid Write

- **Invalid write**: writing/initializing memory that was not allocated
- Usually off-by-one array indices or pointer arithmetic with wrong types

```
Invalid write of size 4
   at 0×4005E7: inner (example_file.c:12)
   by 0×40065E: main (example_file.c:21)
Address 0×5205044 is 0 bytes after a block of size 4 alloc'd
   at 0×4C29F73: malloc (vg_replace_malloc.c:309)
   by 0×40058E: f1 (example_file.c:5)
   by 0×400644: main (example_file.c:19)
```

```
1  #include <stdlib.h>
2  #include <stdio.h>
3
4  int *f1() {
5      int *ip = malloc(sizeof(int));
6      return ip;
7  }
8
9  int inner(int *p) {
10     printf("inner: %d\n", *p);
11     if(*p <= 3) return *p;
12     p[1] = p[0] / 2;
13     int *ip = f1();
14     *ip -= p[1] - 1;
15     return *p + inner(ip);
16 }
17
18 int main() {
19     int *p = f1();
20     *p = 10;
21     int i = inner(p);
22     return i;
23 }
```

Common Errors: Invalid Free

- **Invalid free**: trying to free memory that is not allocated
- Freeing pointers that were already freed or not returned by an `alloc` function

```
Invalid free() / delete / delete[] / realloc()
   at 0×4C2B06D: free (vg_replace_malloc.c:540)
   by 0×4005E9: f2 (example_file.c:15)
   by 0×40060C: main (example_file.c:21)
Address 0×5205040 is 0 bytes inside a block of size 4 free'd
   at 0×4C2B06D: free (vg_replace_malloc.c:540)
   by 0×4005DD: f2 (example_file.c:14)
   by 0×40060C: main (example_file.c:21)
```

```
1  #include <stdlib.h>
2
3  int *f1() {
4      int *ip = malloc(sizeof(int));
5      *ip = 3;
6      return ip;
7  }
8
9  int f2() {
10     int *internal = f1();
11     void *other = (void*)internal;
12     int result = *internal;
13     int *result2 = &result;
14     free(internal);
15     free(other);
16     free(result2);
17     return result;
18 }
19
20 int main() {
21     int i = f2();
22     return i;
23 }
```

omitted: Valgrind will also tell you where this block was allocated (line 4)

Common Errors: Uninitialized Values

- **Uninitialized values:** reading allocated memory without initializing
- Use flag `--track-origins=yes` to see where value was allocated

```
Uninitialised value was created by a heap
allocation
  at 0×4C29F73: malloc (vg_replace_malloc.c:309)
  by 0×40058E: f1 (example_file.c:4)
  by 0×4005AA: f2 (example_file.c:9)
  by 0×4005EC: main (example_file.c:21)
```

```
1  #include <stdlib.h>
2
3  int *f1() {
4      int *ip = malloc(sizeof(int));
5      return ip;
6  }
7
8  int f2() {
9      int *internal = f1();
10     int other = 3;
11
12     if (*internal < 5) {
13         other = *internal;
14     }
15
16
17     return other;
18 }
19
20 int main() {
21     int i = f2();
22     return i;
23 }
```

without `--track-origins=yes`, we only get a `Conditional jump or move depends on uninitialised value(s)` error (line 12) — this still shows up with the flag on but we have omitted it

Leaking Memory

- **Leaked memory:** forgetting to free allocated memory
- Rule of thumb: always free everything you allocate
- Use flag `--leak-check=full` to get more details

```
4 bytes in 1 blocks are definitely lost in loss
record 1 of 1
   at 0×4C29E63: malloc (vg_replace_malloc.c:309)
   by 0×40053E: f1 (example_file.c:4)
   by 0×400572: f2 (example_file.c:11)
   by 0×400590: main (example_file.c:17)
```

```
1  #include <stdlib.h>
2
3  int *f1() {
4      int *ip = malloc(sizeof(int));
5
6      *ip = 3;
7      return ip;
8  }
9
10 int f2() {
11     int *internal = f1();
12
13     return *internal;
14 }
15
16 int main() {
17     int i = f2();
18     return i;
19 }
```


_02
Valgrind Exercises

Basic Exercises

- Run both files using the gcc compilation commands in `README.txt`
- Both files should have relatively small fixes! File 2 has a memory leak, see if you can catch it

Classic Exercise

- This file involves a **LOT** of errors in the Valgrind output! Try not to get too discouraged and start at the very top and work your way down

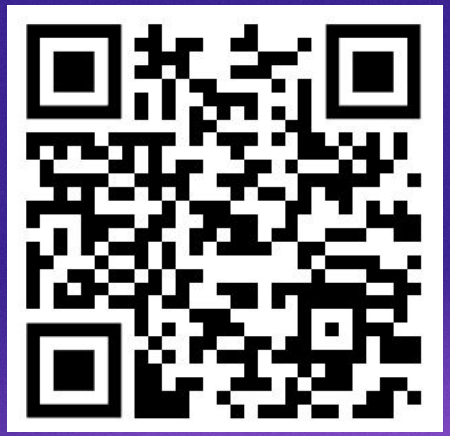
Challenge Exercise

- We try to write a program that stores and prints Pascal triangles, but it has lots of memory-related issues... Help us fix it! (The two functions you should look at are `generate` and the `main` function)

THANK YOU!

PLEASE FILL OUT THE FEEDBACK FORM!

We want to help future 122 students succeed — thanks for being the guinea pigs for the 122 Bootcamp series this summer :)



tinyurl.com/valgrindFB