Objectives

- Memory faults
- Valgrind
- Types of memory faults
  - Memory leaks
  - "Conditional jump depends on uninitialized value"
  - Invalid read/write – after free(…)
  - Invalid read/write – buffer overread / buffer overflow
  - Double free
  - Dereferencing NULL
- Debugging segmentation faults

Rules for heap memory

1. free(…) what you malloc(…).
   - Free the buffer when you no longer need it.
2. Don't read from uninitialized memory
   - Tip: Allocate the memory right before you are going to initialize it.
3. Don't access memory that's not “yours”.
   - Stay within the bounds of your buffer.
   - Do not access buffer after it has been free'd.
4. Don't try to read or write at NULL
   - NULL is an address that is not yours. (Numerically, it is the 0 address.)
   - We often use NULL to indicate that something "empty" or "does not exist."

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

char* repeat_char(char char_to_repeat, size_t num_times) {
    // Ex: repeat_char('*', 3) ⇒ "***"

    // Allocate buffer sufficient for 'num_times' characters plus '\0'
    char* s = malloc(sizeof(*s) * (num_times + 1));

    // Fill buffer with 'char_to_repeat'
    for(size_t i = 0; i < num_times; i++) {
        s[i] = char_to_repeat;
    }

    // Add null terminator
    s[num_times] = '\0';

    return s;
}

void print_string(char* s) {
    // Equivalent to:  printf("%s", s);
    for(int i = 0; s[i] != '\0'; i++) {
        fputc(s[i], stdout);
    }
}

int main(int argc, char* argv[]) {
    char* s = repeat_char('@', 3); // ⇒ "@@@"
    print_string(s); // same as printf("%s", s)
    free(s);
    return EXIT_SUCCESS;
}
```

Output:
@@@

Correct

```
 1  char* repeat_char(char ch, size_t num_times) {
 2      char* s = malloc(sizeof(*s) * (num_times + 1));
 3      for(size_t i = 0; i < num_times; i++) {
 4          s[i] = ch;
 5      }
 6      s[num_times] = '\0';
 7      return s;
 8  }
 9
10  void print_string(char* s) {
11      for(int i = 0; s[i] != '\0'; i++) {
12          fputc(s[i], stdout);
13      }
14  }
15
16  int main(int argc, char* argv[]) {
17      char* s = repeat_char('@', 3); // ⇒ "@@@"
18      print_string(s); // same as printf("%s", s)
19      free(s);
20      return EXIT_SUCCESS;
21  }
```

Valgrind

- Valgrind is tool that detects memory faults

```
$ gcc -o test_join_strings test_join_strings.c join_strings.c

$ valgrind ./test_join_strings
```

Correct » Valgrind output

```
==127197== Memcheck, a memory error detector
==127197== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==127197== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==127197== Command: ./correct
==127197==
@@@==127197==
==127197== HEAP SUMMARY:
==127197==     in use at exit: 0 bytes in 0 blocks
==127197==   total heap usage: 1 allocs, 1 frees, 4 bytes allocated
==127197==
==127197== All heap blocks were freed -- no leaks are possible
==127197==
==127197== For lists of detected and suppressed errors, rerun with: -s
==127197== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

No memory leaks! 😎

malloc(…) was called 1 time

free(…) was called 1 time

malloc(…) allocated a total of 4 bytes across all calls

Memory leak

```
 1  char* repeat_char(char ch, size_t num_times) {
 2      char* s = malloc(sizeof(*s) * (num_times + 1));
 3      for(size_t i = 0; i < num_times; i++) {
 4          s[i] = ch;
 5      }
 6      s[num_times] = '\0';
 7      return s;
 8  }
 9
10  void print_string(char* s) {
11      for(int i = 0; s[i] != '\0'; i++) {
12          fputc(s[i], stdout);
13      }
14  }
15
16  int main(int argc, char* argv[]) {
17      char* s = repeat_char('@', 3);
18      print_string(s);
19
20      // ▶ Whoops... forgot to free(…) ◀
21
22      return EXIT_SUCCESS;
23  }  // ▶▶▶ !!! MEMORY LEAK … 1 block … 4 bytes !!! ◀◀◀
```

Memory leak » Valgrind output

```
==127158== Memcheck, a memory error detector
==127158== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==127158== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==127158== Command: ./memory_leak
==127158==
@@@==127158==
==127158== HEAP SUMMARY:
==127158==     in use at exit: 4 bytes in 1 blocks
==127158==   total heap usage: 1 allocs, 0 frees, 4 bytes allocated
==127158==
==127158== 4 bytes in 1 blocks are definitely lost in loss record 1 of 1
==127158==    at 0x4C29F73: malloc (vg_replace_malloc.c:309)
==127158==    by 0x4005E5: repeat_char (memory_leak.c:2)
==127158==    by 0x400699: main (memory_leak.c:17)
==127158==
==127158== LEAK SUMMARY:
==127158==    definitely lost: 4 bytes in 1 blocks
==127158==    indirectly lost: 0 bytes in 0 blocks
==127158==      possibly lost: 0 bytes in 0 blocks
==127158==    still reachable: 0 bytes in 0 blocks
==127158==         suppressed: 0 bytes in 0 blocks
==127158==
==127158== For lists of detected and suppressed errors, rerun with: -s
==127158== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

Conditional jump … depends on uninitialized value

```
 1  char* repeat_char(char ch, size_t num_times) {
 2      char* s = malloc(sizeof(*s) * (num_times + 1));
 3      for(size_t i = 0; i < num_times; i++) {
 4          s[i] = ch;
 5      }
 6      // ▶ Whoops... forgot to write null terminator ('\0') ◀
 7      return s;
 8  }
 9
10  void print_string(char* s) {                //
11      for(int i = 0; s[i] != '\0'; i++) { //  ▶▶▶ !!! CONDITIONAL JUMP… ◀◀◀
12          fputc(s[i], stdout);            //      depends on unintialized value
13      }                                   //
14  }
15
16  int main(int argc, char* argv[]) {
17      char* s = repeat_char('@', 3);
18      print_string(s);
19      free(s);
20      return EXIT_SUCCESS;
21  }
```

Conditional jump … depends on uninitialized value » Valgrind output

```
==127233== Memcheck, a memory error detector
==127233== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==127233== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==127233== Command: ./conditional_jump
==127233==
==127233== Conditional jump or move depends on uninitialised value(s)
==127233==    at 0x4006B9: print_string (conditional_jump.c:11)
==127233==    by 0x4006EB: main (conditional_jump.c:18)
==127233==  Uninitialised value was created by a heap allocation
==127233==    at 0x4C29F73: malloc (vg_replace_malloc.c:309)
==127233==    by 0x400635: repeat_char (conditional_jump.c:2)
==127233==    by 0x4006DB: main (conditional_jump.c:17)
==127233==
@@==127233==
==127233== HEAP SUMMARY:
==127233==     in use at exit: 0 bytes in 0 blocks
==127233==   total heap usage: 1 allocs, 1 frees, 4 bytes allocated
==127233==
==127233== All heap blocks were freed -- no leaks are possible
==127233==
==127233== For lists of detected and suppressed errors, rerun with: -s
==127233== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

Invalid read … after free

```
 1  char* repeat_char(char ch, size_t num_times) {
 2      char* s = malloc(sizeof(*s) * (num_times + 1));
 3      for(size_t i = 0; i < num_times; i++) {
 4          s[i] = ch;
 5      }
 6      s[num_times] = '\0';
 7      return s;
 8  }
 9
10  void print_string(char* s) {
11      for(int i = 0; s[i] != '\0'; i++) { // ▶▶▶ INVALID READ … free'd ◀◀◀
12          fputc(s[i], stdout);            // ▶▶▶ INVALID READ … free'd ◀◀◀
13      }
14  }
15
16  int main(int argc, char* argv[]) {
17      char* s = repeat_char('@', 3);
18      free(s);  // ▶ Whoops... should free(…) only after done using buffer ◀
19      print_string(s);
20      return EXIT_SUCCESS;
21  }
```

Invalid read … after free » Valgrind output

```
…
==127069==
==127069== Invalid read of size 1
==127069==    at 0x4006C2: print_string (invalid_read-after_free.c:11)
==127069==    by 0x400705: main (invalid_read-after_free.c:19)
==127069==  Address 0x5205040 is 0 bytes inside a block of size 4 free'd
==127069==    at 0x4C2B06D: free (vg_replace_malloc.c:540)
==127069==    by 0x4006F9: main (invalid_read-after_free.c:18)
==127069==  Block was alloc'd at
==127069==    at 0x4C29F73: malloc (vg_replace_malloc.c:309)
==127069==    by 0x400635: repeat_char (invalid_read-after_free.c:2)
==127069==    by 0x4006E9: main (invalid_read-after_free.c:17)
==127069==
==127069== Invalid read of size 1
==127069==    at 0x4006A1: print_string (invalid_read-after_free.c:12)
==127069==    by 0x400705: main (invalid_read-after_free.c:19)
==127069==  Address 0x5205040 is 0 bytes inside a block of size 4 free'd
==127069==    at 0x4C2B06D: free (vg_replace_malloc.c:540)
==127069==    by 0x4006F9: main (invalid_read-after_free.c:18)
==127069==  Block was alloc'd at
==127069==    at 0x4C29F73: malloc (vg_replace_malloc.c:309)
==127069==    by 0x400635: repeat_char (invalid_read-after_free.c:2)
==127069==    by 0x4006E9: main (invalid_read-after_free.c:17)
==127069==
…
```

Invalid write … after free

```
 1  char* repeat_char(char ch, size_t num_times) {
 2      char* s = malloc(sizeof(*s) * (num_times + 1));
 3      for(size_t i = 0; i < num_times; i++) {
 4          s[i] = ch;
 5      }
 6      s[num_times] = '\0';
 7      return s;
 8  }
 9
10  void print_string(char* s) {
11      for(int i = 0; s[i] != '\0'; i++) {
12          fputc(s[i], stdout);
13      }
14  }
15
16  int main(int argc, char* argv[]) {
17      char* s = repeat_char('@', 3);
18      print_string(s);
19      free(s);
20
21      s[1] = '_'; // change to "@_@" // ▶▶▶ !!! INVALID WRITE … free'd !!! ◀◀◀
22
23      return EXIT_SUCCESS;
24  }
```

Invalid write … after free » Valgrind output

```
…
==127145==
==127145== Invalid write of size 1
==127145==    at 0x40070E: main (invalid_write.after_free.c:21)
==127145==  Address 0x5205041 is 1 bytes inside a block of size 4 free'd
==127145==    at 0x4C2B06D: free (vg_replace_malloc.c:540)
==127145==    by 0x400705: main (invalid_write.after_free.c:19)
==127145==  Block was alloc'd at
==127145==    at 0x4C29F73: malloc (vg_replace_malloc.c:309)
==127145==    by 0x400635: repeat_char (invalid_write.after_free.c:2)
==127145==    by 0x4006E9: main (invalid_write.after_free.c:17)
==127145==
…
```

Invalid read / write … ▒ bytes after

```
 1  char* repeat_char(char ch, size_t num_times) {
 2      char* s = malloc(sizeof(*s) * num_times);  // ▶ Whoops... no room for '\0' ◀
 3      for(size_t i = 0; i < num_times; i++) {
 4          s[i] = ch;
 5      }
 6      s[num_times] = '\0';  // ▶▶▶ !!! INVALID WRITE … 0 bytes after !!! ◀◀◀
 7      return s;             //     AKA "buffer overflow"
 8  }
 9
10  void print_string(char* s) {
11      for(int i = 0; s[i] != '\0'; i++) { // ▶▶▶ !! INVALID READ … 0 bytes after !! ◀◀◀
12          fputc(s[i], stdout);            //     (when s==2)  AKA "buffer overread"
13      }
14  }
15
16  int main(int argc, char* argv[]) {
17      char* s = repeat_char('@', 3);
18      print_string(s);
19      free(s);
20      return EXIT_SUCCESS;
21  }
```

Invalid read / write … ▒ bytes after » Valgrind output

```
==127104==
==127104== Invalid write of size 1
==127104==    at 0x40066B: repeat_char (invalid_read.invalid_write.0_bytes_after.c:6)
==127104==    by 0x4006E5: main (invalid_read.invalid_write.0_bytes_after.c:17)
==127104==  Address 0x5205043 is 0 bytes after a block of size 3 alloc'd
==127104==    at 0x4C29F73: malloc (vg_replace_malloc.c:309)
==127104==    by 0x400631: repeat_char (invalid_read.invalid_write.0_bytes_after.c:2)
==127104==    by 0x4006E5: main (invalid_read.invalid_write.0_bytes_after.c:17)
==127104==
==127104== Invalid read of size 1
==127104==    at 0x4006BE: print_string (invalid_read.invalid_write.0_bytes_after.c:11)
==127104==    by 0x4006F5: main (invalid_read.invalid_write.0_bytes_after.c:18)
==127104==  Address 0x5205043 is 0 bytes after a block of size 3 alloc'd
==127104==    at 0x4C29F73: malloc (vg_replace_malloc.c:309)
==127104==    by 0x400631: repeat_char (invalid_read.invalid_write.0_bytes_after.c:2)
==127104==    by 0x4006E5: main (invalid_read.invalid_write.0_bytes_after.c:17)
==127104==
```

Double free

```
 1  char* repeat_char(char ch, size_t num_times) {
 2      char* s = malloc(sizeof(*s) * (num_times + 1));
 3      for(size_t i = 0; i < num_times; i++) {
 4          s[i] = ch;
 5      }
 6      s[num_times] = '\0';
 7      free(s);  // ▶ Whoops... should not free(…) here ◀
 8      return s;
 9  }
10
11  void print_string(char* s) {
12      for(int i = 0; s[i] != '\0'; i++) {
13          fputc(s[i], stdout);
14      }
15  }
16
17  int main(int argc, char* argv[]) {
18      char* s = repeat_char('@', 3);
19      print_string(s);
20      free(s);  // ▶▶▶ !!! DOUBLE FREE !!! ◀◀◀
21      return EXIT_SUCCESS;
22  }
```

Double free » CRASH 

```
$ ./double_free
*** Error in `./double_free': double free or corruption (fasttop): 0x0000000001cca010 ***
======= Backtrace: =========
/lib64/libc.so.6(+0x81329)[0x7f0f7f477329]
./double_free[0x400712]
/lib64/libc.so.6(__libc_start_main+0xf5)[0x7f0f7f418555]
./double_free[0x400519]
======= Memory map: ========
00400000-00401000 r-xp 00000000 00:206 21514542733                      …/double_free
00600000-00601000 r--p 00000000 00:206 21514542733                      …/double_free
00601000-00602000 rw-p 00001000 00:206 21514542733                      …/double_free
01cca000-01ceb000 rw-p 00000000 00:00 0                                 [heap]
7f0f78000000-7f0f78021000 rw-p 00000000 00:00 0
7f0f78021000-7f0f7c000000 ---p 00000000 00:00 0
7f0f7f1df000-7f0f7f1f6000 r-xp 00000000 00:5f 2362745505                 …/libgcc_s.so.1
7f0f7f1f6000-7f0f7f3f5000 ---p 00017000 00:5f 2362745505                 …/libgcc_s.so.1
7f0f7f3f5000-7f0f7f3f6000 rw-p 00016000 00:5f 2362745505                 …/libgcc_s.so.1
7f0f7f5ba000-7f0f7f7b9000 ---p 001c4000 fd:00 100673250                  …/libc-2.17.so
7f0f7f7b9000-7f0f7f7bd000 r--p 001c3000 fd:00 100673250                  …/libc-2.17.so
7f0f7f7bd000-7f0f7f7bf000 rw-p 001c7000 fd:00 100673250                  …/libc-2.17.so
7f0f7f7bf000-7f0f7f7c4000 rw-p 00000000 00:00 0
7f0f7f7c4000-7f0f7f7e6000 r-xp 00000000 fd:00 100673242                  …/ld-2.17.so
7f0f7f9a3000-7f0f7f9a6000 rw-p 00000000 00:00 0
7f0f7f9e3000-7f0f7f9e5000 rw-p 00000000 00:00 0
7f0f7f9e5000-7f0f7f9e6000 r--p 00021000 fd:00 100673242                  …/ld-2.17.so
7f0f7f9e6000-7f0f7f9e7000 rw-p 00022000 fd:00 100673242                  …/ld-2.17.so
7f0f7f9e7000-7f0f7f9e8000 rw-p 00000000 00:00 0
7ffcad53d000-7ffcad55f000 rw-p 00000000 00:00 0                         [stack]
7ffcad58e000-7ffcad590000 r-xp 00000000 00:00 0                         [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                 [vsyscall]
Aborted (core dumped)
```

Double free » Valgrind output

```
==127171== Memcheck, a memory error detector
==127171== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==127171== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==127171== Command: ./double_free
==127171==
==127171== Invalid read of size 1
==127171==    at 0x4006CE: print_string (double_free.c:12)
==127171==    by 0x400705: main (double_free.c:19)
==127171==  Address 0x5205040 is 0 bytes inside a block of size 4 free'd
==127171==    at 0x4C2B06D: free (vg_replace_malloc.c:540)
==127171==    by 0x40067D: repeat_char (double_free.c:7)
==127171==    by 0x4006F5: main (double_free.c:18)
==127171==  Block was alloc'd at
==127171==    at 0x4C29F73: malloc (vg_replace_malloc.c:309)
==127171==    by 0x400635: repeat_char (double_free.c:2)
==127171==    by 0x4006F5: main (double_free.c:18)
==127171==
==127171== Invalid read of size 1
==127171==    at 0x4006AD: print_string (double_free.c:13)
==127171==    by 0x400705: main (double_free.c:19)
==127171==  Address 0x5205040 is 0 bytes inside a block of size 4 free'd
==127171==    at 0x4C2B06D: free (vg_replace_malloc.c:540)
==127171==    by 0x40067D: repeat_char (double_free.c:7)
==127171==    by 0x4006F5: main (double_free.c:18)
==127171==  Block was alloc'd at
==127171==    at 0x4C29F73: malloc (vg_replace_malloc.c:309)
==127171==    by 0x400635: repeat_char (double_free.c:2)
==127171==    by 0x4006F5: main (double_free.c:18)
==127171==
==127171== Invalid free() / delete / delete[] / realloc()
==127171==    at 0x4C2B06D: free (vg_replace_malloc.c:540)
==127171==    by 0x400711: main (double_free.c:20)
==127171==  Address 0x5205040 is 0 bytes inside a block of size 4 free'd
==127171==    at 0x4C2B06D: free (vg_replace_malloc.c:540)
==127171==    by 0x40067D: repeat_char (double_free.c:7)
==127171==    by 0x4006F5: main (double_free.c:18)
==127171==  Block was alloc'd at
==127171==    at 0x4C29F73: malloc (vg_replace_malloc.c:309)
==127171==    by 0x400635: repeat_char (double_free.c:2)
==127171==    by 0x4006F5: main (double_free.c:18)
==127171==
@@==127171==
==127171== HEAP SUMMARY:
==127171==     in use at exit: 0 bytes in 0 blocks
==127171==   total heap usage: 1 allocs, 2 frees, 4 bytes allocated
==127171==
==127171== All heap blocks were freed -- no leaks are possible
==127171==
==127171== For lists of detected and suppressed errors, rerun with: -s
==127171== ERROR SUMMARY: 8 errors from 3 contexts (suppressed: 0 from 0)
```

Double free » Valgrind output

```
==127…
==127171== Invalid read of size 1
==127==    at 0x4006CE: print_string (double_free.c:12)
==127==    by 0x400705: main (double_free.c:19)
==127==  Address 0x5205040 is 0 bytes inside a block of size 4 free'd
==127==    at 0x4C2B06D: free (vg_replace_malloc.c:540)
==127==    by 0x40067D: repeat_char (double_free.c:7)
==127==    by 0x4006F5: main (double_free.c:18)
==127==  Block was alloc'd at
==127==    at 0x4C29F73: malloc (vg_replace_malloc.c:309)
==127==    by 0x400635: repeat_char (double_free.c:2)
==127==    by 0x4006F5: main (double_free.c:18)
==127==
==127171== Invalid read of size 1
==127==    at 0x4006AD: print_string (double_free.c:13)
==127==    by 0x400705: main (double_free.c:19)
==127==  Address 0x5205040 is 0 bytes inside a block of size 4 free'd
==127==    at 0x4C2B06D: free (vg_replace_malloc.c:540)
==127==    by 0x40067D: repeat_char (double_free.c:7)
==127==    by 0x4006F5: main (double_free.c:18)
==127==  Block was alloc'd at
@@==
==127==    at 0x4C29F73: malloc (vg_replace_malloc.c:309)
==127==    by 0x400635: repeat_char (double_free.c:2)
==127==    by 0x4006F5: main (double_free.c:18)
==127==
==127…
```

Double free » Valgrind output

```
==127171== Memcheck, a memory error detector
==127171== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==127171== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==127171== Command: ./double_free
==127171==
==127171== Invalid read of size 1
==127171==    at 0x4006CE: print_string (double_free.c:12)
==127171==    by 0x400705: main (double_free.c:19)
==127171==  Address 0x5205040 is 0 bytes inside a block of size 4 free'd
==127171==    at 0x4C2B06D: free (vg_replace_malloc.c:540)
==127171==    by 0x40067D: repeat_char (double_free.c:7)
==127171==    by 0x4006F5: main (double_free.c:18)
==127
==127
…
==127171== Invalid free() / delete / delete[] / realloc()
==127
==127171==    at 0x4C2B06D: free (vg_replace_malloc.c:540)
==127
==127171==    by 0x400711: main (double_free.c:20)
==127
==127171==  Address 0x5205040 is 0 bytes inside a block of size 4 free'd
==127
==127171==    at 0x4C2B06D: free (vg_replace_malloc.c:540)
==127
==127171==    by 0x40067D: repeat_char (double_free.c:7)
==127
==127171==    by 0x4006F5: main (double_free.c:18)
==127
==127171==  Block was alloc'd at
==127
==127171==    at 0x4C29F73: malloc (vg_replace_malloc.c:309)
==127
==127171==    by 0x400635: repeat_char (double_free.c:2)
==127
==127171==    by 0x4006F5: main (double_free.c:18)
==127
==127
…
==127
==127171==    by 0x4006F5: main (double_free.c:18)
==127171==
@@@==127171==
==127171== HEAP SUMMARY:
==127171==     in use at exit: 0 bytes in 0 blocks
==127171==   total heap usage: 1 allocs, 2 frees, 4 bytes allocated
==127171==
==127171== All heap blocks were freed -- no leaks are possible
==127171==
==127171== For lists of detected and suppressed errors, rerun with: -s
==127171== ERROR SUMMARY: 8 errors from 3 contexts (suppressed: 0 from 0)
```

Double free » Valgrind output

```
==127171== Memcheck, a memory error detector
==127171== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==127171== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==127171== Command: ./double_free
==127171==
==127171== Invalid read of size 1
==127171==    at 0x4006CE: print_string (double_free.c:12)
==127171==    by 0x400705: main (double_free.c:19)
==127171==  Address 0x5205040 is 0 bytes inside a block of size 4 free'd
==127171==    at 0x4C2B06D: free (vg_replace_malloc.c:540)
==127171==    by 0x40067D: repeat_char (double_free.c:7)
==127171==    by 0x4006F5: main (double_free.c:18)
==127171==  Block was alloc'd at
==127171==    at 0x4C29F73: malloc (vg_replace_malloc.c:309)
==127171==    by 0x400635: repeat_char (double_free.c:2)
==127171==    by 0x4006F5: main (double_free.c:18)
==127171==
==127171== Invalid read of size 1
==127171==    at 0x4006AD: print_string (double_free.c:13)
==127171==    by 0x400705: main (double_free.c:19)
==127171==  Address 0x5205040 is 0 bytes inside a block of size 4 free'd
==127171==    at 0x4C2B06D: free (vg_replace_malloc.c:540)
==127171==    by 0x40067D: repeat_char (double_free.c:7)
==127171==    by 0x4006F5: main (double_free.c:18)
==127171==  Block was alloc'd at
==127171==    at 0x4C29F73: malloc (vg_replace_malloc.c:309)
==127171==    by 0x400635: repeat_char (double_free.c:2)
==127171==    by 0x4006F5: main (double_free.c:18)
==127171==
==127171== Invalid free() / delete / delete[] / realloc()
==127171==    at 0x4C2B06D: free (vg_replace_malloc.c:540)
==127171==    by 0x400711: main (double_free.c:20)
==127171==  Address 0x5205040 is 0 bytes inside a block of size 4 free'd
==127171==    at 0x4C2B06D: free (vg_replace_malloc.c:540)
==127171==    by 0x40067D: repeat_char (double_free.c:7)
==127171==    by 0x4006F5: main (double_free.c:18)
==127171==  Block was alloc'd at
==127171==
==127171==
==127171==
==127171==
@@@==12717
==127171==
==127171==
==127171==
==127171==
==127171==
==127171==
==127171==
==127171==
```

```
…
==127171== HEAP SUMMARY:
==127171==     in use at exit: 0 bytes in 0 blocks
==127171==   total heap usage: 1 allocs, 2 frees, 4 bytes allocated
==127171==
==127171== All heap blocks were freed -- no leaks are possible
==127171==

…
```

Segmentation fault - dereferencing NULL

```
 1  char* repeat_char(char ch, size_t num_times) {
 2      char* s = malloc(sizeof(*s) * (num_times + 1));
 3      for(size_t i = 0; i < num_times; i++) {
 4          s[i] = ch;
 5      }
 6      s[num_times] = '\0';
 7      return s;
 8  }
 9
10  void print_string(char* s) {
11      for(int i = 0; s[i] != '\0'; i++) {// ▶▶▶ SEGMENTATION FAULT !!! ◀◀◀
12          fputc(s[i], stdout);
13      }
14  }
15
16  int main(int argc, char* argv[]) {
17      char* s = repeat_char('@', 3);
18      free(s);
19      s = NULL;
20      print_string(s); // ▶ Whoops… will attempt to access memory at NULL ◀
21      return EXIT_SUCCESS;
22  }
```

PAUSE (memory faults)

NULL

- We set address variables to NULL to signify:
  - empty
  - does not exist
  - none
  - nothing to refer to
- NULL is an address that is off-limits.
  - NULL is a symbolic constant defined as 0 in stdlib.h.
  - Address 0 is in a reserved section of memory we are not allowed to access.
- Dereferencing NULL causes a Segmentation Fault
  - "Dereference" means to try to access (read or write) a location in memory.
  - `*a` and `a[0]` are both ways of dereferencing the address `a`.
  - "Segmentation fault" is a crash that happens if you try to read (or write) a location in memory where you are not allowed to read (or write).

*UN*PAUSE (memory faults)

Segmentation fault - dereferencing NULL - CRASH 

```
$ ./segmentation_fault.null
Segmentation fault (core dumped)
```

Segmentation fault - dereferencing NULL » Valgrind output

```
…
==127213== Command: ./segmentation_fault.null
==127213==
==127213== Invalid read of size 1
==127213==    at 0x4006C2: print_string (segmentation_fault.null.c:11)
==127213==    by 0x40070D: main (segmentation_fault.null.c:20)
==127213==  Address 0x0 is not stack'd, malloc'd or (recently) free'd
==127213==
==127213==
==127213== Process terminating with default action of signal 11 (SIGSEGV)
==127213==  Access not within mapped region at address 0x0
==127213==    at 0x4006C2: print_string (segmentation_fault.null.c:11)
==127213==    by 0x40070D: main (segmentation_fault.null.c:20)
==127213==  If you believe this happened as a result of a stack
==127213==  overflow in your program's main thread (unlikely but
==127213==  possible), you can try to increase the size of the
==127213==  main thread stack using the --main-stacksize= flag.
==127213==  The main thread stack size used in this run was 8388608.
==127213==
…
```

How to debug a segmentation fault

- Run in GDB
  - You do not need to create any breakpoints.
- When it crashes, inspect the expression for anything equivalent to any of these:
  - `*a` *// write at address a*
  - `a[▒]`
  - `a -> ▒` *// The -> operator will be covered soon.*
- Use **`p`** (⇔ **`print`**) to view an expression
  - Ex: `(gdb) p a`
  - Ex: `(gdb) print a`