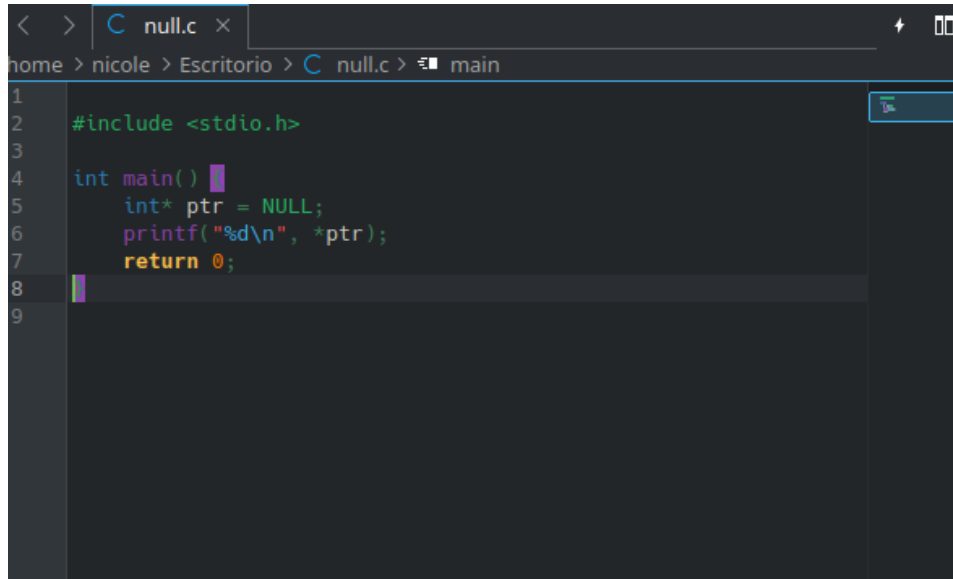


Estudiante: Nicole Araya Ballesterero

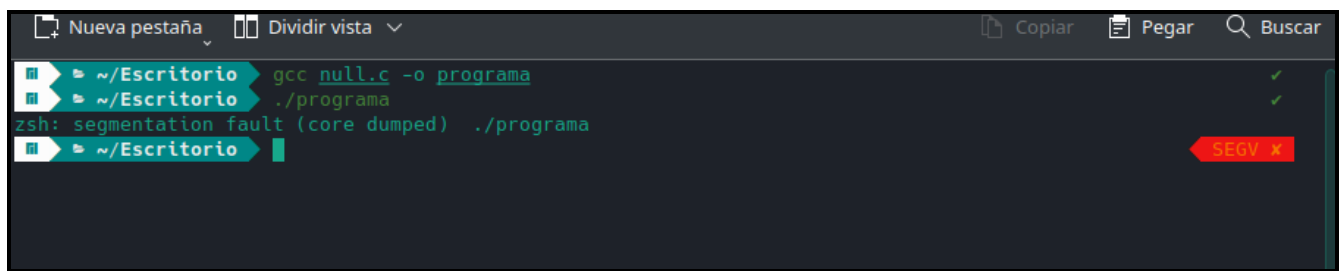
Questions

1. First, write a simple program called null.c that creates a pointer to an integer, sets it to NULL, and then tries to dereference it. Compile this into an executable called null. What happens when you run this program?



```
1 #include <stdio.h>
2
3
4 int main()
5 {
6     int* ptr = NULL;
7     printf("%d\n", *ptr);
8     return 0;
9 }
```

Hubo un fallo en la segmentación por estar en null.



```
Nueva pestaña  Dividir vista  Copiar  Pegar  Buscar
~/Escritorio gcc null.c -o programa ✓
~/Escritorio ./programa ✓
zsh: segmentation fault (core dumped) ./programa
~/Escritorio
```

2. Next, compile this program with symbol information included (with the -g flag). Doing so let's put more information into the executable, enabling the debugger to access more useful information about variable names and the like. Run the program under the debugger by typing gdb null and then, once gdb is running, typing run. What does gdb show you?

Muestra la ubicación exacta donde se produjo la violacion de segmento y la pila de llamadas hasta donde se produjo el error.

Con Backtrace vemos la pila de llamadas nada más, en este caso la función main.

```

Program received signal SIGSEGV, Segmentation fault.
0x00005555555514d in main () at null.c:6
6      printf("%d\n", *ptr);
(gdb) backtrace
#0  0x00005555555514d in main () at null.c:6
(gdb) █

```

3. Finally, use the valgrind tool on this program. We'll use the memcheck tool that is a part of valgrind to analyze what happens. Run this by typing in the following: `valgrind --leak-check=yes null`. What happens when you run this? Can you interpret the output from the tool?

La herramienta me dice que se ha detectado una lectura no válida de tamaño 4 bytes en la línea 6 del programa, en la función main. También, la dirección 0x0 no ha sido asignada mediante malloc ni una pila. Esto significa que se está intentando acceder a una dirección de memoria que no está permitida. Y el programa se ha bloqueado debido a una señal SIGSEGV (violación de segmento).

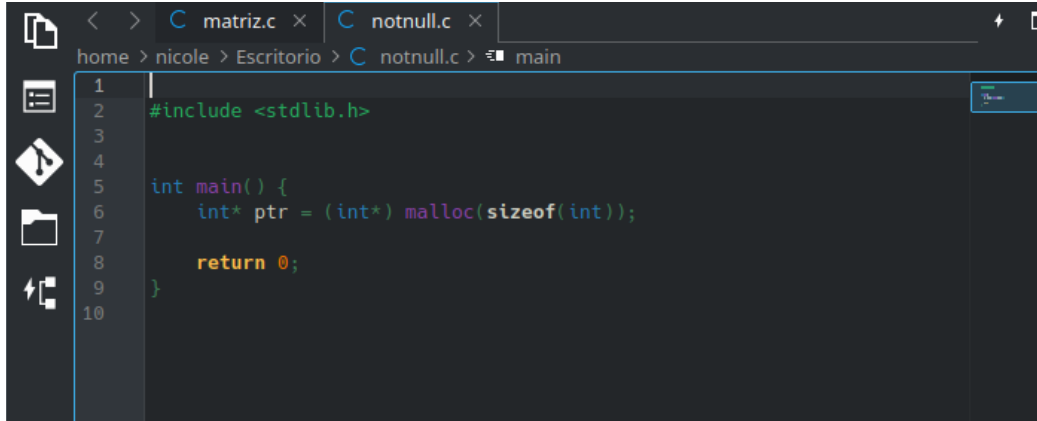
La herramienta valgrind ha detectado un error de violación de memoria en el programa. Este error se produce cuando se intenta leer una dirección de memoria no permitida.

```

~ /Escritorio valgrind --leak-check=yes ./programa
==16750== Memcheck, a memory error detector
==16750== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==16750== Using Valgrind-3.20.0 and LibVEX; rerun with -h for copyright info
==16750== Command: ./programa
==16750==
==16750== Invalid read of size 4
==16750==   at 0x109140: main (null.c:6)
==16750==   Address 0x0 is not stack'd, malloc'd or (recently) free'd
==16750==
==16750== Process terminating with default action of signal 11 (SIGSEGV): dumping core
==16750== Access not within mapped region at address 0x0
==16750==   at 0x109140: main (null.c:6)
==16750== If you believe this happened as a result of a stack
==16750== overflow in your program's main thread (unlikely but
==16750== possible), you can try to increase the size of the
==16750== main thread stack using the --main-stacksize= flag.
==16750== The main thread stack size used in this run was 8388608.
==16750==
==16750== HEAP SUMMARY:
==16750==   in use at exit: 0 bytes in 0 blocks
==16750==   total heap usage: 0 allocs, 0 frees, 0 bytes allocated
==16750==
==16750== All heap blocks were freed -- no leaks are possible
==16750==
==16750== For lists of detected and suppressed errors, rerun with: -s
==16750== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
zsh: segmentation fault (core dumped) valgrind --leak-check=yes ./programa
~ /Escritorio █

```

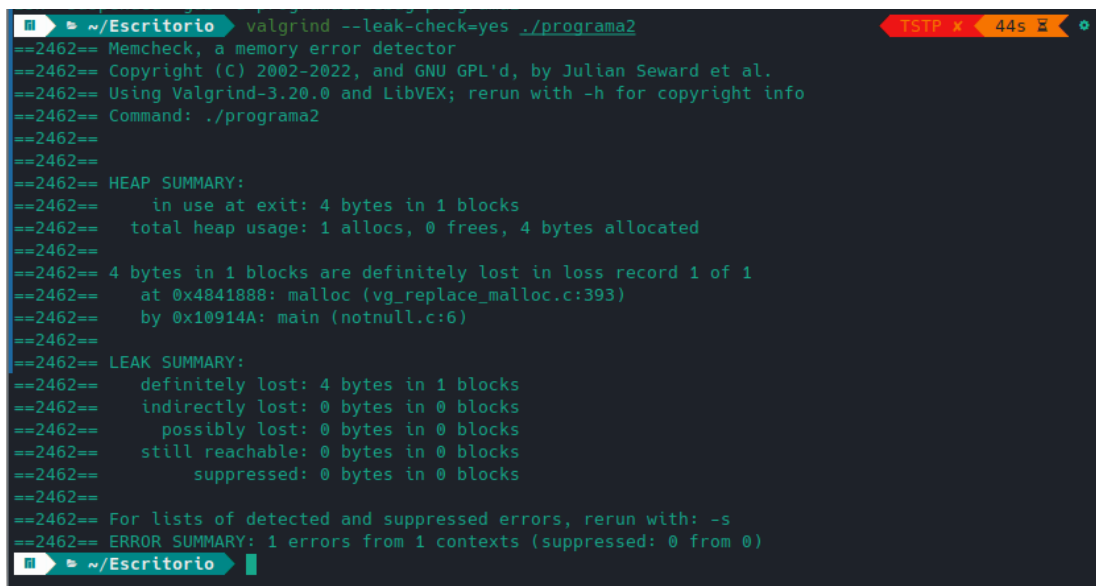
4. Write a simple program that allocates memory using malloc() but forgets to free it before exiting. What happens when this program runs? Can you use gdb to find any problems with it? How about valgrind (again with the --leak-check=yes flag)?



```
1  
2 #include <stdlib.h>  
3  
4  
5 int main() {  
6     int* ptr = (int*) malloc(sizeof(int));  
7  
8     return 0;  
9 }  
10
```

La memoria asignada por malloc() no se libera antes de salir, por lo que hay una fuga de memoria, esto se liberará hasta que se reinicie el sistema.

Use valgrind y nos dice que hay una pérdida de 4 bits en malloc() en la función main, ya que no fue liberado



```
~/Escritorio valgrind --leak-check=yes ./programa2 TSTP 44s  
==2462== Memcheck, a memory error detector  
==2462== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.  
==2462== Using Valgrind-3.20.0 and LibVEX; rerun with -h for copyright info  
==2462== Command: ./programa2  
==2462==  
==2462== HEAP SUMMARY:  
==2462==   in use at exit: 4 bytes in 1 blocks  
==2462==   total heap usage: 1 allocs, 0 frees, 4 bytes allocated  
==2462==  
==2462== 4 bytes in 1 blocks are definitely lost in loss record 1 of 1  
==2462==    at 0x4841888: malloc (vg_replace_malloc.c:393)  
==2462==    by 0x10914A: main (notnull.c:6)  
==2462==  
==2462== LEAK SUMMARY:  
==2462==    definitely lost: 4 bytes in 1 blocks  
==2462==    indirectly lost: 0 bytes in 0 blocks  
==2462==    possibly lost: 0 bytes in 0 blocks  
==2462==    still reachable: 0 bytes in 0 blocks  
==2462==    suppressed: 0 bytes in 0 blocks  
==2462==  
==2462== For lists of detected and suppressed errors, rerun with: -s  
==2462== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

5. Write a program that creates an array of integers called data of size 100 using malloc; then, set data[100] to zero. What happens when you run this program? What happens when you run this program using valgrind? Is the program correct?

El programa no da ningún error ni fuga de memoria, ya que se está liberando.

Los índices de una matriz van de 0 el último índice válido es 99. Por lo que acceder a `data[100]` está yendo más allá del tamaño asignado de la matriz.

```
~/Escritorio gcc -g matriz.c -o programa3
~/Escritorio valgrind --leak-check=yes ./programa3
==3652== Memcheck, a memory error detector
==3652== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==3652== Using Valgrind-3.20.0 and LibVEX; rerun with -h for copyright info
==3652== Command: ./programa3
==3652==
==3652== HEAP SUMMARY:
==3652==   in use at exit: 0 bytes in 0 blocks
==3652==   total heap usage: 1 allocs, 1 frees, 400 bytes allocated
==3652==
==3652== All heap blocks were freed -- no leaks are possible
==3652==
==3652== For lists of detected and suppressed errors, rerun with: -s
==3652== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
~/Escritorio
```

6. Create a program that allocates an array of integers (as above), frees them, and then tries to print the value of one of the elements of the array. Does the program run? What happens when you use valgrind on it?

```
matriz-2.c
home > nicole > Escritorio > C matriz-2.c > main
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main() {
5      int *data = malloc(sizeof(int) * 100);
6      if (data == NULL) {
7          printf("Error: no se pudo asignar memoria\n");
8          return 1;
9      }
10
11     // inicializar todos los elementos a cero
12     for (int i = 0; i < 100; i++) {
13         data[i] = 0;
14     }
15
16     // liberar la memoria asignada
17     free(data);
18
19     // intentar imprimir un elemento de la matriz
20     printf("%d\n", data[0]);
21
22     return 0;
23 }
24
25
```

Muestra un "Invalid read" de tamaño 4. Además, valgrind indica que la dirección 0x4a61040 se encuentra dentro de un bloque de tamaño 400 que ya ha sido liberado.

Entonces el valor que muestra al ser imprimido es un número aleatorio ya que fue liberada la memoria, también podría bloquearse el programa.

```
~/Escritorio gcc -g matriz-2.c -o programa4
~/Escritorio ./programa4
1484866412
~/Escritorio valgrind --leak-check=yes ./programa4
==4578== Memcheck, a memory error detector
==4578== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==4578== Using Valgrind-3.20.0 and LibVEX; rerun with -h for copyright info
==4578== Command: ./programa4
==4578==
==4578== Invalid read of size 4
==4578== at 0x1091D9: main (matriz-2.c:20)
==4578== Address 0x4a61040 is 0 bytes inside a block of size 400 free'd
==4578== at 0x484426F: free (vg_replace_malloc.c:884)
==4578== by 0x1091D4: main (matriz-2.c:17)
==4578== Block was alloc'd at
==4578== at 0x4841888: malloc (vg_replace_malloc.c:393)
==4578== by 0x10917A: main (matriz-2.c:5)
==4578==
0
==4578==
==4578== HEAP SUMMARY:
==4578== in use at exit: 0 bytes in 0 blocks
==4578== total heap usage: 2 allocs, 2 frees, 1,424 bytes allocated
==4578==
==4578== All heap blocks were freed -- no leaks are possible
==4578==
==4578== For lists of detected and suppressed errors, rerun with: -s
==4578== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
~/Escritorio
```

7. Now pass a funny value to free (e.g., a pointer in the middle of the array you allocated above). What happens? Do you need tools to find this type of problem?

```
C matriz-2.c x
home > nicole > Escritorio > C matriz-2.c > main
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main() {
5     int *data = malloc(sizeof(int) * 100);
6     int *ptr=&data[50];
7
8     if (data == NULL) {
9         printf("Error: no se pudo asignar memoria\n");
10        return 1;
11    }
12
13
14    for (int i = 0; i < 100; i++) {
15        data[i] = 0;
16    }
17
18    free(ptr);
19
20    printf("%d\n", data[0]);
21
22    return 0;
23 }
```

Se usó AddressSanitizer para ver los errores.

Se produjo un error de uso después de liberar la memoria asignada a un puntero en medio de la matriz.

```

~/Escritorio gcc -fsanitize=address matriz-2.c -o programa4
matriz-2.c: En la función 'main':
matriz-2.c:18:5: aviso: 'free' called on pointer 'data' with nonzero offset 200 [-Wfree-nonheap-object]
18 |     free(ptr);
    |     ^~~~~~
matriz-2.c:5:17: nota: returned from 'malloc'
5 |     int *data = malloc(sizeof(int) * 100);
    |                   ^~~~~~~~~~~~~~~~~~~~~~

~/Escritorio ./programa4
=====
==5033==ERROR: AddressSanitizer: attempting free on address which was not malloc()-ed: 0x61400000108 in thread T0
#0 0x7fed300be672 in __interceptor_free /usr/src/debug/gcc/gcc/libsanitizer/asan/asan_malloc_linux.cpp:52
#1 0x55f17b45127d in main (/home/nicole/Escritorio/programa4+0x127d)
#2 0x7fed2fe3c78f (/usr/lib/libc.so.6+0x2378f)
#3 0x7fed2fe3c849 in __libc_start_main (/usr/lib/libc.so.6+0x23849)
#4 0x55f17b4510f4 in _start (/home/nicole/Escritorio/programa4+0x10f4)

0x61400000108 is located 200 bytes inside of 400-byte region [0x61400000040,0x614000001d0)
allocated by thread T0 here:
#0 0x7fed300bfa89 in __interceptor_malloc /usr/src/debug/gcc/gcc/libsanitizer/asan/asan_malloc_linux.cpp:69
#1 0x55f17b4511da in main (/home/nicole/Escritorio/programa4+0x11da)
#2 0x7fed2fe3c78f (/usr/lib/libc.so.6+0x2378f)

SUMMARY: AddressSanitizer: bad-free /usr/src/debug/gcc/gcc/libsanitizer/asan/asan_malloc_linux.cpp:52 in __interceptor_free
==5033==ABORTING

```

8. Try out some of the other interfaces to memory allocation. For example, create a simple vector-like data structure and related routines that use `realloc()` to manage the vector. Use an array to store the vectors elements; when a user adds an entry to the vector, use `realloc()` to allocate more space for it. How well does such a vector perform? How does it compare to a linked list? Use `valgrind` to help you find bugs.

```

Preg8.c
home > nicole > Escritorio > Preg8.c > main

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  struct vector {
5      int *data;
6      size_t size;
7      size_t capacity;
8  };
9
10 void vector_init(struct vector *v) {
11     v->data = NULL;
12     v->size = 0;
13     v->capacity = 0;
14 }
15
16 void vector_resize(struct vector *v, size_t new_size) {
17     if (new_size > v->capacity) {
18         v->capacity = new_size;
19         v->data = realloc(v->data, v->capacity * sizeof(int));
20     }
21     v->size = new_size;
22 }
23
24 void vector_add(struct vector *v, int value) {
25     size_t new_size = v->size + 1;
26     vector_resize(v, new_size);
27     v->data[new_size - 1] = value;
28 }
29
30 void vector_free(struct vector *v) {
31     free(v->data);
32     v->data = NULL;
33     v->size = 0;
34     v->capacity = 0;
35 }
36
37 int main() {
38     struct vector v;
39     vector_init(&v);
40
41     for (int i = 0; i < 10; i++) {
42         vector_add(&v, i);
43     }
44
45     for (int i = 0; i < v.size; i++) {
46         printf("d ", v.data[i]);
47     }
48     printf("\n");
49     vector_free(&v);
50
51     return 0;
52 }
53
54
55

```

--leak-check=yes para detectar fugas de memoria y --track-origins=yes para rastrear la fuente de valores no inicializados

```
Escritorio : zsh — Konsole
Archivo  Editar  Ver  Marcadores  Complementos  Preferencias  Ayuda
Nueva pestaña  Dividir vista  Copiar

~/Escritorio gcc -g Preg8.c -o programa5
~/Escritorio ./programa5
0 1 2 3 4 5 6 7 8 9
~/Escritorio valgrind --leak-check=yes --track-origins=yes ./programa5
==5610== Memcheck, a memory error detector
==5610== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==5610== Using Valgrind-3.20.0 and LibVEX; rerun with -h for copyright info
==5610== Command: ./programa5
==5610==
0 1 2 3 4 5 6 7 8 9
==5610==
==5610== HEAP SUMMARY:
==5610==    in use at exit: 0 bytes in 0 blocks
==5610==   total heap usage: 11 allocs, 11 frees, 1,244 bytes allocated
==5610==
==5610== All heap blocks were freed -- no leaks are possible
==5610==
==5610== For lists of detected and suppressed errors, rerun with: -s
==5610== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
~/Escritorio
```

9. Spend more time and read about using gdb and valgrind. Knowing your tools is critical; spend the time and learn how to become an expert debugger in the UNIX and C environment.

Okis