CSE 333 – SECTION 2

Programming Tools

Questions, Comments, Concerns

- Do you have any?
- Exercises going ok?
- Lectures make sense?
- Homework 1 – Should have started by now!

Exercises!

- Comments
  - Program Comments – Author, copyright, problem description at the top
  - Function Comments – Near the prototype/declaration in header files; local functions are a more complex story, but near the prototype works for those too.
- `clint` or `cpplint` errors
- Valgrind errors
- Check for error codes/return values and handle them correctly!

GNU Debugger (gdb)

- Use it!
- Run your C program using gcc with the –g flag along with the other relevant flags
- Refer to the gdb common commands card linked on the course website
- Explore the –tui option with gdb

Demo

gdb demo: [*buggy.c*]

Valgrind

- Use of uninitialized memory
- Reading/writing memory after it has been freed
- Reading/writing of the end of malloc'd blocks
- Reading/writing inappropriate areas on the stack
- Memory leaks (where pointers to malloc'd blocks are lost forever)
- Mismatched use of malloc/new/new[] vs free/delete/delete[]
- These errors usually lead to crashes.

Reading uninitialized memory

Code

```
1  #include "stdlib.h"
2  int main(int argc, char *argv[]) {
3    int *x;
4    *x = 4; // XXX Using x before initialized.
5    return EXIT_SUCCESS;
6  }
```

Valgrind Output

```
==2205== Use of uninitialised value of size 8
==2205==    at 0x4004AB: main (error.c:4)
```

Illegal reads and writes

```c
#include "stdlib.h"
#include "stdio.h"
int main(int argc, char *argv[]) {
  int *x = (int*)malloc(sizeof(int));
  x += 2; // x now points to invalid memory (some random location).
  printf("%d\n", *x); // XXX Reading to an invalid location of memory.
  *x = 4;             // XXX Writing to an invalid location of memory.
  free(x-2);
  printf("%d\n", *((int*)3838338)); // XXX And even worse read.
  return EXIT_SUCCESS;
}
```

```
==3023== Invalid read of size 4
==3023==    at 0x400592: main (error.c:6)
==3023==  Address 0x51d2048 is 4 bytes after a block of size 4 alloc'd
==3023==    at 0x4C2A93D: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==3023==    by 0x400584: main (error.c:4)
==3023==
==3023== Invalid write of size 4
==3023==    at 0x4005A9: main (error.c:7)
==3023==  Address 0x51d2048 is 4 bytes after a block of size 4 alloc'd
==3023==    at 0x4C2A93D: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==3023==    by 0x400584: main (error.c:4)
==3023==
==3023== Invalid read of size 4
==3023==    at 0x4005C4: main (error.c:9)
==3023==  Address 0x3a9182 is not stack'd, malloc'd or (recently) free'd
```

Memory leaks

Code

```
1  #include "stdlib.h"
2  #include "stdio.h"
3  int main(int argc, char *argv[]) {
4    int *x = (int*)malloc(sizeof(int));
5    *x = 4;
6    printf("%d\n", *x);
7    return EXIT_SUCCESS; // XXX Oh no!  We didn't free x.
8  }
```

Valgrind Output

```
==3093== HEAP SUMMARY:
==3093==     in use at exit: 4 bytes in 1 blocks
==3093==   total heap usage: 1 allocs, 0 frees, 4 bytes allocated
==3093==
==3093== 4 bytes in 1 blocks are definitely lost in loss record 1 of 1
==3093==    at 0x4C2A93D: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==3093==    by 0x400544: main (error.c:3)
==3093==
==3093== LEAK SUMMARY:
==3093==    definitely lost: 4 bytes in 1 blocks
==3093==    indirectly lost: 0 bytes in 0 blocks
==3093==      possibly lost: 0 bytes in 0 blocks
==3093==    still reachable: 0 bytes in 0 blocks
==3093==         suppressed: 0 bytes in 0 blocks
```

Demo

Valgrind demo: [*leaky.c*]

Section Exercise 1

- Memory Diagrams (Handout 1)

Section Exercise 2

- Clean up buggy code *imsobuggy.c* (Handout 2)