

# CSE 333 – SECTION 2

---

Memory Management

# Questions, Comments, Concerns

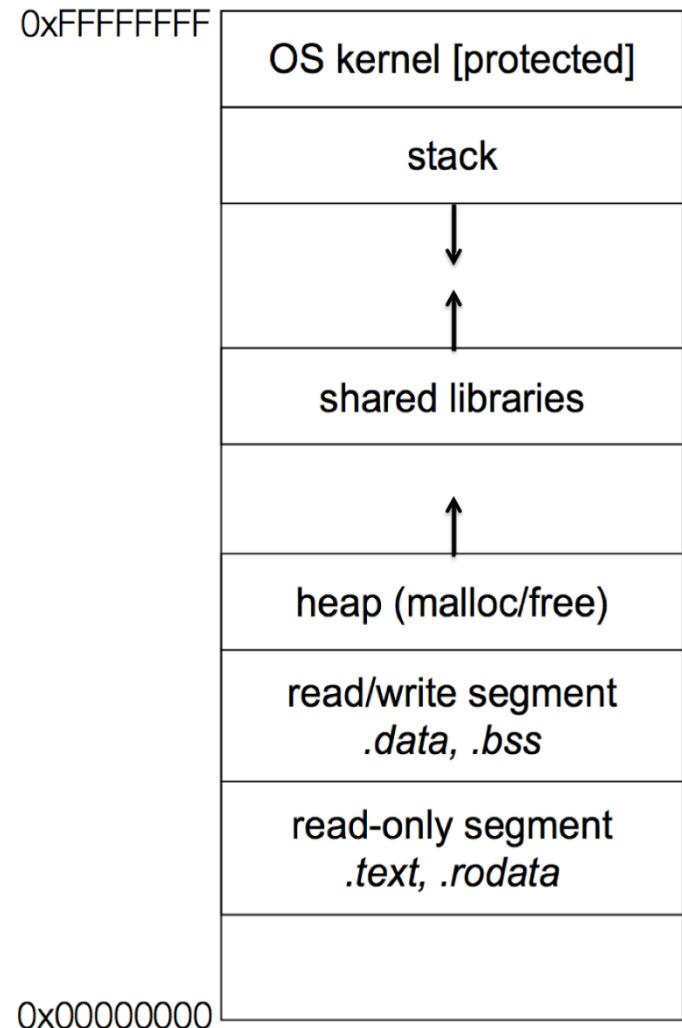
- Do you have any?
- Exercises going ok?
- Lectures make sense?
- Homework 1 – START NOW! (if you haven't already)

# Debugging with gdb

- Just like in CSE 351, gdb is your friend
- Unlike CSE 351, we will be debugging C/C++ code, not assembly
  - Instead of `n(ext)i` and `s(tep)i`, use `n(ext)` and `s(tep)`
- Your first instinct for bug fixing should be gdb, not `printf`
- If you want something a little more friendly, use `gdb -tui`
  - It's pretty darn helpful!
- Demo: [`buggy.c`]

# Memory Management

- Heap
  - Large pool of unused memory
  - `malloc()` allocates chunks of this memory
  - `free()` deallocates memory and reclaims space
- Stack and stack frame
  - Stores temporary/local variables
  - Each function has its own stack frame
- Lifetime on heap vs. Lifetime on stack



# Memory Errors

- Use of uninitialized memory
- Reading/writing memory after it has been freed – Dangling pointers
- Reading/writing to the end of malloc'd blocks
- Reading/writing to inappropriate areas on the stack
- Memory leaks where pointers to malloc'd blocks are lost
- Mismatched use of malloc/new/new[] vs free/delete/delete[]

**Valgrind is your friend!!**

# Buggy Code

```
1  #include "stdio.h"
2  #include "stdlib.h"
3
4  // Returns an array containing [n, n+1, ..., m-1, m]. If n > m, then the
5  // array returned is []. If an error occurs, NULL is returned.
6  int *RangeArray(int n, int m) {
7      int length = m-n+1;
8
9      // Heap-allocate the array needed to return.
10     int *array = (int*)malloc(sizeof(int)*length);
11
12     // Initialize the elements.
13     for (int i = 0; i <= length; ++i)
14         array[i] = i+n;
15
16     return array;
17 }
18
19 // Accepts two integers as arguments
20 int main(int argc, char *argv[]) {
21     if (argc != 3) return EXIT_FAILURE;
22     int n = atoi(argv[1]), m = atoi(argv[2]); // Parse cmd-line args.
23     int *nums = RangeArray(n, m);
24
25     // Print the resulting array.
26     for (int i = 0; i <= (m-n+1); ++i)
27         printf("%d ", nums[i]);
28     puts("");
29
30     return EXIT_SUCCESS;
31 }
```

```
==22891== Command: ./warmup 1 10
==22891==
==22891== Invalid write of size 4
==22891==   at 0x400616: RangeArray (warmup.c:14)
==22891==   by 0x400683: main (warmup.c:22)
==22891== Address 0x51d2068 is 0 bytes after a block of size 40 alloc'd
==22891==   at 0x4C2A93D: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==22891==   by 0x4005EC: RangeArray (warmup.c:10)
==22891==   by 0x400683: main (warmup.c:22)
==22891==
==22891== Invalid read of size 4
==22891==   at 0x4006A5: main (warmup.c:26)
==22891== Address 0x51d2068 is 0 bytes after a block of size 40 alloc'd
==22891==   at 0x4C2A93D: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==22891==   by 0x4005EC: RangeArray (warmup.c:10)
==22891==   by 0x400683: main (warmup.c:22)
==22891==
1 2 3 4 5 6 7 8 9 10 11
==22891==
==22891== HEAP SUMMARY:
==22891==   in use at exit: 40 bytes in 1 blocks
==22891== total heap usage: 1 allocs, 0 frees, 40 bytes allocated
==22891==
==22891== 40 bytes in 1 blocks are definitely lost in loss record 1 of 1
==22891==   at 0x4C2A93D: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==22891==   by 0x4005EC: RangeArray (warmup.c:10)
==22891==   by 0x400683: main (warmup.c:22)
==22891==
==22891== LEAK SUMMARY:
==22891==   definitely lost: 40 bytes in 1 blocks
==22891==   indirectly lost: 0 bytes in 0 blocks
==22891==   possibly lost: 0 bytes in 0 blocks
==22891==   still reachable: 0 bytes in 0 blocks
==22891==   suppressed: 0 bytes in 0 blocks
==22891==
==22891== For counts of detected and suppressed errors, rerun with: -v
==22891== ERROR SUMMARY: 3 errors from 3 contexts (suppressed: 3 from 3)
```

# Code Fix

```
#include "stdio.h"
#include "stdlib.h"

// Returns an array of [n, n+1, ..., m-1, m]
// If n > m, then the array returned is [].
// If an error occurs, NULL is returned.
int *RangeArray(int n, int m) {
    int length;
    int *array;

    // XXX We must check this explicitly.
    if (n > m)
        return (int*)malloc(0);

    // Heap-allocate the array needed to return.
    length = m-n+1;
    array = (int*)malloc(sizeof(int)*length);

    // XXX We need to check if malloc'd returned successfully.
    if (array == NULL)
        return NULL;

    // Initialize the elements.
    // XXX We had an off-by-one error here.
    for (int i = 0; i < length; ++i)
        array[i] = i+n;

    return array;
}
```



# Code Fix

```
int main(int argc, char *argv[]) {
    if (argc != 3) return EXIT_FAILURE;
    int n = atoi(argv[1]), m = atoi(argv[2]);
    int *nums = RangeArray(n, m);

    // XXX Terminate program with failure if RangeArray cannot allocate and initialize the array.
    if (nums == NULL)
        return EXIT_FAILURE;

    // Print the resulting array.
    // XXX We had another off-by-one error here.
    for (int i = 0; i < (m-n+1); ++i)
        printf("%d ", nums[i]);
    puts("");

    // XXX Free storage before terminating.
    free(nums);
    return EXIT_SUCCESS;
}
```

# Basic Valgrind Usage

## Command

```
valgrind ./a.out
```

## Example Output

```
==26428== Memcheck, a memory error detector
==26428== Copyright (C) 2002-2011, and GNU GPL'd, by Julian Seward et al.
==26428== Using Valgrind-3.7.0 and LibVEX; rerun with -h for copyright info
==26428== Command: ./a.out
==26428==
..... LOTS OF ERRORS .....
==26428==
==26428== HEAP SUMMARY:
==26428==   in use at exit: 528 bytes in 22 blocks
==26428==   total heap usage: 22 allocs, 0 frees, 528 bytes allocated
==26428==
==26428== LEAK SUMMARY:
==26428==   definitely lost: 408 bytes in 11 blocks
==26428==   indirectly lost: 120 bytes in 11 blocks
==26428==   possibly lost: 0 bytes in 0 blocks
==26428==   still reachable: 0 bytes in 0 blocks
==26428==   suppressed: 0 bytes in 0 blocks
==26428== Rerun with --leak-check=full to see details of leaked memory
==26428==
==26428== For counts of detected and suppressed errors, rerun with: -v
==26428== Use --track-origins=yes to see where uninitialised values come from
==26428== ERROR SUMMARY: 65 errors from 16 contexts (suppressed: 3 from 3)
```

- Note: Compile your C code with the GCC's `-g` option for debugging information.
- Note: Valgrind accepts tags `--leak-check=full` and `--show-reachable=yes` to output more details.

# Use of uninitialized value

## 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/writes

## 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 += 2; // x now points to invalid memory (some random location).
6      printf("%d\n", *x); // XXX Reading to an invalid location of memory.
7      *x = 4;           // XXX Writing to an invalid location of memory.
8      free(x-2);
9      printf("%d\n", *((int*)3838338)); // XXX And even worse read.
10     return EXIT_SUCCESS;
11 }
```

```
==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
```

# Illegal frees

## Code

```
1  #include "stdlib.h"
2  int main(int argc, char *argv[]) {
3      free((void*) 0xdeadbeef); // XXX free some random address free'd.
4
5      int *x = (int*)malloc(sizeof(int));
6      free(x+4);                // XXX free outside malloc'd block.
7      free(x);
8
9      return EXIT_SUCCESS;
10 }
```

## Valgrind Output

```
==2978== Invalid free() / delete / delete[] / realloc()
==2978==   at 0x4C29A9E: free (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==2978==   by 0x400544: main (error.c:3)
==2978==   Address 0xdeadbeef is not stack'd, malloc'd or (recently) free'd
==2978==
==2978== Invalid free() / delete / delete[] / realloc()
==2978==   at 0x4C29A9E: free (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==2978==   by 0x400562: main (error.c:6)
==2978==   Address 0x51d2050 is 12 bytes after a block of size 4 alloc'd
==2978==   at 0x4C2A93D: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==2978==   by 0x40054E: main (error.c:5)
```

# 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
```

# Section exercise

- Handouts.
- Work with a partner, if you wish.
- Look at the expandable vector code in `imsobuggy.c`.
- First, try to find all the bugs by inspection.
- Then try to use Valgrind on the same code.
- Code is located at <http://courses.cs.washington.edu/courses/cse333/15su/sections/sec2-code/>