CSE333 SECTION 4

Important Dates

- October 27th – Homework 2 Due
- October 29th – Midterm

Survey

- Let us know how the course is going!

- In the feedback section feel free to write about:
  - Improvements we can make in:
    - Lecture
    - Section
    - Assignments (Exercises & Homework)
  - Anything else you can think of related to the course

Assignment Turn In Policy

- Assignments are due at the specified due date
- The Drop-Box will close at some point after the due date
- After that point, submissions will happen by email
- **Might or Might Not be <u>accepted</u>**

GDB - Debugging

- Requirements:
  - System has GDB (Lab Machines/Attu do)
  - Compile code with –g (debugging symbols) symbol
- Running GDB:
  - gdb <file name>
  - run <optional command line arguments>
- Stopping GDB:
  - If the command line is listening, quit or Ctrl-d
  - To halt the current process, Ctrl-c

GDB – Breakpoints

- Setting breakpoints:
  - At a function, break <function name>
  - At an address, break *<address>
  - Breakpoints are issued a number used to identify them
- info breakpoints – get breakpoint identifiers and more
- Deleting breakpoints:
  - Remove a single breakpoint, delete <breakpoint number>
  - Remove all the breakpoints, delete

GDB – Execution

- To execute one statement, s or step
- To go to next breakpoint, continue
- To go to specific breakpoint, until <breakpoint identifier>
- To finish current function, finish

- Calling a function:
  - call <function name>(argument, argument, …, argument)

GDB – Examining Data

- To get the names and values of local variables, info locals
- To get information about the current stack, info stack

- Printing Data
  - Modifiers:
    - /d – decimal
    - /x – hex
    - /t – binary
  - To print the value of a single variable, print <variable name>

Valgrind – Memory Management

- Possible problems:
  - Use uninitialized memory
  - Read/Write after freeing
  - Read/Write outside of memory block
  - Read/Write on inappropriate part of stack
  - Memory leaks
  - Mismatched use of malloc/free

- Running Valgrind:
  - valgrind ./(executable) (If in current directory)
  - Valgrind options
    - --leak-check=full, used to display more information
    - --show-reachable=yes, show if the memory is still reachable

Valgrind - Asides

- Will not work on Macs
- Is also useful for discovering more info about seg faults

Valgrind – Error output

```
int main(int argc, char **argv) {
  int *x;
  *x = 333;
  return EXIT_SUCCESS;
}
```

Valgrind output:

```
Use of uninitialized value of size 8
```

Valgrind – Error output

```
int main(int argc, char **argv) {
  int *x = (int *) malloc(sizeof(int));
  x += 2;
  printf(“My value: %d\n”, *x);
  *x = 4;
  free(x - 2);
  printf(“My value: %d\n”, *x);
  return EXIT_SUCCESS;
}
```

(Continued on next slide)

Valgrind – Error output

(Continued from previous slide)

Valgrind output:

- Invalid read of size 4
- Invalid write of size 4
- Invalid read of size 4

Can you identify the problems?

Illegal Frees

```
int main(int argc, char **argv) {
  free((void *) 0xcafefood);

  int *x = (int *) malloc(sizeof(int));
  free(x + 4);
  free(x);

  return EXIT_SUCCESS;
}
```

(Continued on next slide)

Valgrind – Error output

(Continued from previous slide)

Valgrind output:

- Invalid free() / delete / delete[] / realloc()
- Invalid free() / delete / delete[] / realloc()

Valgrind – Error output

```
int main(int argc, char** argv) {
  int *x = (int *) malloc(sizeof(int));
  *x = 333;
  return EXIT_SUCCESS;
}
```

(Example code)

Some code from lecture

(See lectureProblem/)

Lets run valgrind on our app and ensure that its leak free.

Lecture Code – Valgrind output

```
==5140== 26 bytes in 1 blocks are definitely lost in loss record 1 of 4
==5140==    at 0x4A0645D: malloc (in /usr/lib64/valgrind/vgpreload_memcheck-amd64-linux.so)
==5140==    by 0x3BF8874B07: vasprintf (in /usr/lib64/libc-2.18.so)
==5140==    by 0x3BF8851CA6: asprintf (in /usr/lib64/libc-2.18.so)
==5140==    by 0x4006E9: point_toString (Point.c:20)
==5140==    by 0x400856: main (App.c:10)
==5140==
==5140== 26 bytes in 1 blocks are definitely lost in loss record 2 of 4
==5140==    at 0x4A0645D: malloc (in /usr/lib64/valgrind/vgpreload_memcheck-amd64-linux.so)
==5140==    by 0x3BF8874B07: vasprintf (in /usr/lib64/libc-2.18.so)
==5140==    by 0x3BF8851CA6: asprintf (in /usr/lib64/libc-2.18.so)
==5140==    by 0x4006E9: point_toString (Point.c:20)
==5140==    by 0x40089C: main (App.c:16)
==5140==
==5140== 27 bytes in 1 blocks are definitely lost in loss record 3 of 4
==5140==    at 0x4A0645D: malloc (in /usr/lib64/valgrind/vgpreload_memcheck-amd64-linux.so)
==5140==    by 0x3BF8874B07: vasprintf (in /usr/lib64/libc-2.18.so)
==5140==    by 0x3BF8851CA6: asprintf (in /usr/lib64/libc-2.18.so)
==5140==    by 0x4007D9: vector_toString (Vector.c:20)
==5140==    by 0x400838: main (App.c:10)
==5140==
==5140== 27 bytes in 1 blocks are definitely lost in loss record 4 of 4
==5140==    at 0x4A0645D: malloc (in /usr/lib64/valgrind/vgpreload_memcheck-amd64-linux.so)
==5140==    by 0x3BF8874B07: vasprintf (in /usr/lib64/libc-2.18.so)
==5140==    by 0x3BF8851CA6: asprintf (in /usr/lib64/libc-2.18.so)
==5140==    by 0x4007D9: vector_toString (Vector.c:20)
==5140==    by 0x400847: main (App.c:10)
```

Lecture Code - Problem

- What’s the problem?

Lecture Code - Problem

man asprintf:

- The functions **asprintf**() and **vasprintf**() are analogs of **sprintf**(3) and **vsprintf**(3), except that they allocate a string large enough to hold the output including the terminating null byte ('\0'), and return a pointer to it via the first argument. **This pointer should be passed to free(3) to release the allocated storage when it is no longer needed.**

- asprintf is allocating memory, but we need to free it

Lecture Code - Solutions

- How can we solve this problem?

Lecture Code - Solutions

- Here are two possibilities:
  - Ensure that we free it by individually holding them in variables
  - Use a static global array
  - Can you think of any others?

- What are the downsides of each possibility?

One more example

(See https://courses.cs.washington.edu/courses/cse333/14su/sections/sec2_code/imsobuggy.c)

Can you fix all the problems?