CSE 333 – Section 2: Structs and Debugging

In this class, it is very helpful to be comfortable with `gdb` and `valgrind` as debugging tools. `gdb` allows you to see the source code and has many useful commands for analyzing your program; `valgrind` catches many types of runtime memory errors.

Plenty of debugging resources can be found on the CSE351 GDB page and the CSE333 Debugging Page: https://courses.cs.washington.edu/courses/cse351/23sp/debugging/ and https://courses.cs.washington.edu/courses/cse333/23sp/debug/

Starting gdb

For gdb to work with your C/C++ program, you must compile it using the "-g" flag!
To start up gdb, run the following command (the `-tui` flag is optional and enables a text UI).

```
$ gdb -tui <program file name>
```

Some essential gdb commands

If you want to know more, ask a TA or investigate the resources at the top of the page.

Setting Breakpoints and Continuing

- break <where> — Set a new breakpoint
- info breakpoints — Prints information about the set breakpoints
- continue — Continue normal execution

Controlling Program Execution

- run <command_line_args> — Run the program with provided command_line_args
- next — Go to next instruction, but don't five into functions
- step — Go to next instruction, and dive into functions
- finish — Continue until current function returns
- quit — close gdb

Examining the Current Program

- list — Shows the current or given source context
- backtrace — Shows the call stack
- up — Moves up a stack frame
- down — Moves down a stack frame
- print <expression> — Prints content of variable/memory location/register

Starting valgrind

Note that valgrind only analyzes the code reached during a specific execution of your program.
Run the following command:

```
$ valgrind --leak-check=full ./<program file name>
```

Section Code

Download full code:
wget https://courses.cs.washington.edu/courses/cse333/23sp/sections/02/code/wordcount.c
wget https://courses.cs.washington.edu/courses/cse333/23sp/sections/02/code/Makefile

```
typedef struct word_st {
  char* word;
  int   count;  // The number of times the word appears
} WordCount;

// Increment the count by 1
void IncreaseCount(WordCount wc) {
  wc.count += 1;
}

// Capitalize the first letter in the word
void CapitalizeWord(WordCount* wc_ptr) {
  wc_ptr->word[0] &= ~0x20;
}

// Return a new WordCount with the letters of word in reverse
// order and a count of 0.  Returns NULL on allocation failure.
WordCount ReverseWord(WordCount* wc_ptr) {
  WordCount* rev = (WordCount*) malloc(sizeof(WordCount));
  rev->word = NULL;
  strcpy(rev->word, wc_ptr->word);

  char ch;
  int L = 0, R = strlen(rev->word);
  while (L < R) {
    ch = rev->word[L];
    rev->word[L] = rev->word[R];
    rev->word[R] = ch;
    L++; R--;
  }

  return *rev;
}
```

```c
int main(int argc, char* argv[]) {
  char comp[] = "computer";
  WordCount comp_count = {comp, 5};
  WordCount* comp_ptr = &comp_count;

  // expecting "1. computer, 5"
  printf("1. %s, %d\n", comp_ptr->word, comp_ptr->count);
  IncreaseCount(*comp_ptr);
  // expecting "2. computer, 6"
  printf("2. %s, %d\n", comp_ptr->word, comp_ptr->count);
  CapitalizeWord(comp_ptr);
  // expecting "3. Computer, 6"
  printf("3. %s, %d\n", comp_ptr->word, comp_ptr->count);
  *comp_ptr = ReverseWord(comp_ptr);
  // expecting "4. retupmoC, 0"
  printf("4. %s, %d\n", comp_ptr->word, comp_ptr->count);

  return EXIT_SUCCESS;
}
```

Exercise 1

Draw a memory diagram for the execution of the code above up to the call to `strcpy()` in `ReverseWord()`. Make sure to distinguish between local variables on the Stack- and Heap-allocated memory.

Exercise 2

Feel free to make a few code changes based on your findings in Exercise 1. However, the rest of your time for this exercise should be spent in gdb and valgrind and NOT staring at the code. Find and fix all of the remaining logical and memory errors in the code and try to document/associate each fix with the tool features or output that led you there.

Please use the space below for documenting your errors fixed and tooling assistance.

Exercise 3

Fix any remaining style issues with the code in `wordcount.c`.