CSE 3100 Systems Programming
Lab #4

Welcome to Lab 4! At the start of each lab, you will receive a document like this detailing the tasks you are to complete. Read it carefully. Try your best to finish labs during the lab session. However, labs are not due right after each lab period.

1 Debugging

I've given you a program called `debug.c`. It takes in a string, and a delimiter, and breaks the string into pieces. For example if the string is `jordan_force__uconn`, and the delimiter is _, then *should* be broken into `jordan`, `force`, and `uconn`. Unfortunately, there are some bugs in the program:

```
$ cc -o debug debug.c -g
$ ./debug
Enter string:
jordan-force-uconn
Enter delimiter: -
Num tokens: 2
token: jordan_
token: n
```

There are three bugs here. First, even though the string really contains three tokens, it only counts two. Second, it doesn't even extract the two tokens that it counts correctly.

The third bug has to do with its usage of memory. This can be detected using the tool `valgrind`:

```
[jforce@jforce template]$ valgrind ./debug
==4144== Memcheck, a memory error detector
==4144== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==4144== Using Valgrind-3.16.1 and LibVEX; rerun with -h for copyright info
==4144== Command: ./debug
==4144==
Enter string:
jordan-force-uconn
Enter delimiter: -
==4144== Invalid write of size 1
==4144==    at 0x1091E4: dupString (debug.c:13)
==4144==    by 0x109391: tokenize (debug.c:54)
==4144==    by 0x1094A3: main (debug.c:75)
==4144==  Address 0x4a379d6 is 0 bytes after a block of size 6 alloc'd
==4144==    at 0x483A77F: malloc (vg_replace_malloc.c:307)
==4144==    by 0x1091D4: dupString (debug.c:12)
==4144==    by 0x109391: tokenize (debug.c:54)
==4144==    by 0x1094A3: main (debug.c:75)
==4144==
Num tokens: 2
==4144== Invalid read of size 1
==4144==    at 0x483DCE4: __strlen_sse2 (vg_replace_strmem.c:461)
==4144==    by 0x48D8AB7: __vfprintf_internal (in /usr/lib/libc-2.32.so)
==4144==    by 0x48C3BBE: printf (in /usr/lib/libc-2.32.so)
==4144==    by 0x1094F3: main (debug.c:78)
==4144==  Address 0x4a379d6 is 0 bytes after a block of size 6 alloc'd
```

```
==4144==    at 0x483A77F: malloc (vg_replace_malloc.c:307)
==4144==    by 0x1091D4: dupString (debug.c:12)
==4144==    by 0x109391: tokenize (debug.c:54)
==4144==    by 0x1094A3: main (debug.c:75)
==4144==
token: jordan
token: n
==4144==
==4144== HEAP SUMMARY:
==4144==     in use at exit: 0 bytes in 0 blocks
==4144==   total heap usage: 6 allocs, 6 frees, 2,191 bytes allocated
==4144==
==4144== All heap blocks were freed -- no leaks are possible
==4144==
==4144== For lists of detected and suppressed errors, rerun with: -s
==4144== ERROR SUMMARY: 4 errors from 2 contexts (suppressed: 0 from 0)
```

Here are two helpful hints about these memory bugs:

- The `Invalid write of size 1` is caused by the `dupString` function writing to memory that hasn't been allocated. The fact that it says `Address 0x4a379d6 is 0 bytes after a block...` means that it is writing immediately after a block of memory you have allocated
- The `Invalid read of size 1` is caused by trying to `printf` a string that lacks a null terminator; `printf` iterates through the string until it encounters the null terminator.

These two memory bugs share the same root cause!

1.1 Running

When the bugs are fixed, the output should look like this:

```
$ ./debug
Enter string:
jordan-force-uconn
Enter delimiter: -
numtok: 3
Num tokens: 3
token: jordan
token: force
token: uconn
```

`valgrind` shouldn't detect any memory errors or leaks, either:

```
$ valgrind ./debug
==3178== Memcheck, a memory error detector
==3178== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==3178== Using Valgrind-3.16.1 and LibVEX; rerun with -h for copyright info
==3178== Command: ./debug
==3178==
Enter string:
jordan-force-uconn
Enter delimiter: -
```

```
numtok: 3
Num tokens: 3
token: jordan
token: force
token: uconn
==3178==
==3178== HEAP SUMMARY:
==3178==     in use at exit: 0 bytes in 0 blocks
==3178==   total heap usage: 7 allocs, 7 frees, 2,211 bytes allocated
==3178==
==3178== All heap blocks were freed -- no leaks are possible
==3178==
==3178== For lists of detected and suppressed errors, rerun with: -s
==3178== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

2 Tree

This program builds a *binary search tree* out of a list of words stored in a text file. Then, it prompts the user for a word, and checks if the word they enter is in the list. The `tree.c` file contains an incomplete program to do this. There are several parts that need to be finished:

- The `main` function needs to open the file of words, read and insert them into the tree.
- The `insertIntoTree` function
- The `searchTree` function
- The `destroyTree` function

2.1 Compiling and Running

The program should correctly detect if a word entered by the user is present or absent in the words list. There should be no memory errors or leaks when you run it under `valgrind`:

```
$ cc -o tree tree.c
$ ./tree words.txt
word: jordan
ABSENT
word: thing
ABSENT
word: computer
PRESENT
word:
$ valgrind ./tree words.txt
==3478== Memcheck, a memory error detector
==3478== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==3478== Using Valgrind-3.16.1 and LibVEX; rerun with -h for copyright info
==3478== Command: ./tree words.txt
==3478==
word: jordan
ABSENT
word: computer
PRESENT
```

```
word: ==3478==
==3478== HEAP SUMMARY:
==3478==     in use at exit: 0 bytes in 0 blocks
==3478==   total heap usage: 50 allocs, 50 frees, 7,325 bytes allocated
==3478==
==3478== All heap blocks were freed -- no leaks are possible
==3478==
==3478== For lists of detected and suppressed errors, rerun with: -s
==3478== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Note: Enter `ctrl-d` to exit the program.