Background
The Valgrind tool suite provides a number of debugging and profiling tools that help you make your programs faster and more correct. The most popular of these tools is called Memcheck. It can detect many memory-related errors that are common in C and C++ programs and that can lead to crashes and unpredictable behavior. This lab is meant to familiarize you with Valgrind so that you can use it for future projects while debugging. Submit this worksheet at the end of the recitation along with the tarball.

Introduction
1. Login via SSH to thoth.cs.pitt.edu
2. Change directories to the one we created for the work for this class

```
cp ~wahn/public/cs449/valgrind/quick-start.c ./
gcc -g quick-start.c -o ./quick-start
valgrind --leak-check=yes ./quick-start
```

Part 1: Errors detected by Valgrind
   Focus on Section 4.2: Explanation of error messages from MemCheck.

```
cp ~wahn/public/cs449/valgrind/very-buggy.c ./
gcc -g very-buggy.c -o ./very-buggy
valgrind --leak-check=full --track-origins=yes ./very-buggy
```

5. The URL given above describes 8 types of errors discovered by MemCheck. Write how many errors of each type valgrind discovers for the above example code.

   Illegal read / Illegal write errors ___  Use of uninitialised values ___
   Uninitialised values in system calls ___  Illegal frees ___
   Inappropriate deallocation function ___  Overlapping source and destination blocks ___
   Fishy argument values ___  Memory leak detection ___
6. Make reasonable modifications to very-buggy.c as to remove all memory errors. If, buffers overlap, they should be extended. If values are not initialized, they should be initialized to 0. If there is a leak the memory should be freed appropriately, etc. Once you are done, valgrind should not display any errors when run.

Part 2: Memory Leak Analysis

7. Copy over leak.c to the work directory and follow below directions

```bash
cp ~wahn/public/cs449/valgrind/leak.c ./
gcc -g leak.c -o ./leak
valgrind --leak-check=full --show-reachable=yes ./leak
```

8. At below, draw a pictorial diagram of the data structure created by the program. On the diagram, mark the nodes leaked through direct leaks with the letter ‘D’ and the nodes leaked through indirect leaks with the letter ‘I’.

9. Make modifications to leak.c to insert appropriate free() calls to remove all direct and indirect memory leaks. Once you are done, valgrind should not display any errors when run.

What to Hand In

Hand in this worksheet to the TA and submit your modified very-buggy.c and leak.c source files.

```bash
tar zcvf USERNAME_lab_valgrind.tar.gz very-buggy.c leak.c
cp USERNAME_lab_valgrind.tar.gz ~wahn/submit/449/RECITATION_CLASS_NUMBER/
```