Use Valgrind to check your program for memory errors and memory leaks. Valgrind can do many more things but we will focus on the memory checking part.

Use Valgrind for testing your singly linked list as follows.

```bash
valgrind --leak-check=yes SimpleTest <n>
```

Run it on the `SimpleTest.c` program without the `freeList` function and then run it again after adding the function.

Valgrind is installed in the lab on all systems. Install it on your Linux system with the command:
```
yum install valgrind
```
Valgrind: Overview

- Mostly known for Memcheck, which helps find many common problems in C/C++ code
- Extremely useful tool for any C/C++ programmer
- Similar to proprietary programs such as Purify, Bounds-Checker, CodeGuard and Insure++
- Supports X86/Linux, AMD64/Linux, PPC32/Linux, PPC64/Linux and X86/Darwin (Mac OS X)
- ARM/Linux and MIPS/Linux ports are in progress, some versions for *BSD
- Available for X86/Linux since ~2003, actively developed
Sample (Bad) Code

We will use the following code for demonstration purposes:

```c
/* sample.c */
#include <stdio.h>
#include <stdlib.h>
#define SIZE 100
int main() {
    int i, sum = 0;
    int *a = malloc(SIZE);
    for(i=0; i < SIZE; ++i) sum += a[i];
    a[26] = 1;
    a = NULL;
    if(sum > 0) printf("Hi!\n");
    return 0;
}
```

Contains many bugs. Compiles without warnings or errors.
Invalid Read

Example

```c
==8990== Invalid read of size 4
==8990== at 0x804844A: main (sample.c:7)
==8990== Address 0x417e08c is 0 bytes after a block of size 100 alloc’d
==8990== at 0x4024C1C: malloc (vg_replace_malloc.c:195)
==8990== by 0x8048430: main (sample.c:6)
```

- We read past the end of the allocated array
- Trying to read from area which we are not allowed to access
- Could result in a SEGFAULT and surely doesn’t do what we want
- Valgrind provides enough details to find the problem.
Invalid Write

Example

==8990== Invalid write of size 4
==8990== at 0x8048463: main (sample.c:8)
==8990== Address 0x417e090 is 4 bytes after a block of size 100 alloc’d
==8990== at 0x4024C1C: malloc (vg_replace_malloc.c:195)
==8990== by 0x8048430: main (sample.c:6)

- Similar to invalid read
- Details provided by valgrind:
  - Location of fault (addresses, line number if debug-information present)
  - Stack-trace to fault (you can get more using --num-callers=30)
  - Relevant blocks details and allocation/de-allocation stack-trace
Memory Leaks

- At the end of the run, Valgrind does “Garbage Collection”
- Unreferenced memory in C/C++ ⇒ memory leak

Example

```plaintext
==8990== 100 bytes in 1 blocks are definitely lost in loss record 1 of 1
==8990== at 0x4024C1C: malloc (vg_replace_malloc.c:195)
==8990== by 0x8048430: main (sample.c:6)
```

- Valgrind provides stack-trace for the allocation point
- 3 kinds:
  - Definitely lost (no pointers to allocation)
  - Probably lost (pointers only to the middle of the allocation)
  - Still reachable (block hasn’t been free’d before exit, but pointers to it still exists)
Suppression Files

- Valgrind tends to be very noisy
- Most of the times it is indicating bugs that should be fixed
  - But not always the one we want to fix right now
- Sometimes it is correct code, which Valgrind failed to understand
  - Mostly in sophisticated/extremely optimized library code
  - Also possible when having unusual interactions with the kernel
- Valgrind includes a mechanism to silent a specific error
  - Works with all tools that report errors
  - Simple file format, see documentation for details
  - Valgrind includes suppression for many common libs
References

- http://haifux.org/lectures/239/
- http://valgrind.org