Memory Leaks | Coursera

Memory Leaks

Memory Leaks

When you lose all references to a block of memory (that is, no pointers point at it), and the memory is still allocated, you have *leaked memory* (or you might say your program has a *memory leak*). For small programs that you write in this book, a memory leak may not seem like a big deal—all of the memory allocated to your program will be released by the operating system when your program exits, so who cares if you have a few hundred bytes lying around?

In small programs that run for at most a few seconds, a memory leak may not have much impact. However, in real programs, memory leaks present significant performance issues. Do you ever find that certain programs get slower and slower as they are open longer? Maybe you have a program which, if open for a day or two, you have to restart because it is annoyingly sluggish. A good guess would be that the programmers who wrote it were sloppy, and it is leaking memory.

You, however, are not going to be a sloppy programmer. You are going to free all your memory. When you write a program, you should run it in *valgrind*, and be sure you get the following message at the end:

```
All heap blocks were freed -- no leaks are possible
```

The next video shows code that allocates memory, but does not free it—leaking the memory. Notice how, when **p** goes out of scope, there are no more references to that block of memory, but it remains allocated. The memory cannot be reused for future allocation requests (as it has not been freed), but is not needed by the program any longer (so it should have been freed). The video concludes by fixing the code by adding a call to free.

✓

Completed