# CS 246 Fall 2015 - Tutorial 6

October 23, 2015

## 1 Summary

- Valgrind

## 2 Memory Errors and Valgrind

Note that all code used in this tutorial relies on undefined behaviour and may or may not work the same way on your machine.

### 2.1 Valgrind Basics

- Memory errors (e.g. memory leaks, uninitialized values) are a common problem in C/C++.
- Although your operating system will free any memory leaks when the program ends, many programs run for a long time (eg. browsers, operating systems, video games, word processors) and may slow down or crash due to leaks and other errors.
- It can be really hard to tell where and why these errors are actually happening.
- valgrind is a program designed to help us with this task.
- To run valgrind, we simply need to put valgrind before our normal bash command:

  ```
  valgrind ./exec
  ```

- Your program will run much slower, and extra information printed on stderr will tell you about potential problems.
- If you compile with the -g option, you'll get line number information for debugging purposes.

### 2.2 Memory Leaks

- You'll notice that some of valgrind's output talks about heap usage. This lets you know if your program is leaking any memory.
- Don't worry if it says that there are 72,206 bytes of "still reachable" memory: this is normal and is the fault of the C++ standard library.
- Memory leaks are broken down into four categories:
  - "Definitely lost" blocks are blocks that you forgot to delete and you no longer can. For example, if you forget to delete some heap memory which is only pointed to by a local variable in a function, then the memory is definitely lost once the function ends.
  - "Indirectly lost" blocks are blocks that weren't deleted because you didn't delete a "definitely lost" block. For example, if you don't delete a linked list, the first node is definitely lost and the rest are indirectly lost. These usually go away when you fix any definite leaks.
  - "Possibly lost" blocks are usually blocks that you forgot to delete and are definitely lost. Occasionally, they might actually be blocks which are still reachable. You will usually not encounter these unless you've made pointer arithmetic errors in unusual places.

  - "Still reachable" blocks are blocks that you forgot to delete but, up until the end of the program, still could've deleted them if you remembered. For example, if a global variable contains a pointer to some heap memory, that memory is still reachable.

- If you pass the --leak-check=full option, more detailed information on memory leaks will be printed.
- Note that other arguments need to be passed to valgrind *b*efore the program is!
- Note: `new` int[0] returns allocated memory and will cause a memory leak if not deleted! Furthermore, dereferencing this pointer results in undefined behaviour.

## 2.3 Segmentation Faults

- Most memory errors which aren't memory leaks end up resulting in a segmentation fault.
- A segmentation fault is raised when the operating system realizes that your program is trying to access memory that it shouldn't have access to.
- This is very coarse, so unfortunately if you are accessing memory that you do have access to–but that you don't want to access–a segmentation fault might not be raised.
- When a segmentation fault is raised, valgrind will print a *s*tack trace: a listing of all the functions that were called up until the fault, and which lines they were on (if you passed the -g option when compiling).
- The most familiar example of a segfault happens when you attempt to dereference a null pointer:

```
int *i = NULL;
*i = 3;
```

## 2.4 Silent Memory Errors

- Unfortunately, some memory errors can happen without any effect on the observable outcome of our program.
- This doesn't mean they aren't important. Why?
  - They might sometimes, but not always, alter observable behaviour.
  - After the program is changed they may begin to alter its behaviour.
  - They may result in code whose behaviour is different when recompiled, especially on a different system or compiler.
- Valgrind catches these errors for us, and may sometimes end our program even when it would normally keep running.
- Types of common memory errors which usually won't be reported:
  - Double free
  - Uninitialized values
  - Off-by-one pointer errors
  - Dangling pointers