```
==58838== Memcheck, a memory error detector
==58838== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==58838== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==58838== Command: ./valgrind_test
==58838==
==58838== Conditional jump or move depends on uninitialised value(s)
==58838==    at 0x109181: main (in /home/kam/Documents/school/eecs_678/eecs678-debuggers-lab/debuggers/valgrind_test)
==58838==
==58838==
==58838== HEAP SUMMARY:
==58838==     in use at exit: 1,087 bytes in 140 blocks
==58838==   total heap usage: 140 allocs, 0 frees, 1,087 bytes allocated
==58838==
==58838== 10 bytes in 1 blocks are possibly lost in loss record 1 of 4
==58838==    at 0x483B7F3: malloc (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==58838==    by 0x10919D: main (in /home/kam/Documents/school/eecs_678/eecs678-debuggers-lab/debuggers/valgrind_test)
==58838==
==58838== 1,035 (552 direct, 483 indirect) bytes in 69 blocks are definitely lost in loss record 4 of 4
==58838==    at 0x483B7F3: malloc (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==58838==    by 0x109160: main (in /home/kam/Documents/school/eecs_678/eecs678-debuggers-lab/debuggers/valgrind_test)
==58838==
==58838== LEAK SUMMARY:
==58838==    definitely lost: 552 bytes in 69 blocks
==58838==    indirectly lost: 483 bytes in 69 blocks
==58838==      possibly lost: 10 bytes in 1 blocks
==58838==    still reachable: 42 bytes in 1 blocks
==58838==         suppressed: 0 bytes in 0 blocks
==58838== Reachable blocks (those to which a pointer was found) are not shown.
==58838== To see them, rerun with: --leak-check=full --show-leak-kinds=all
==58838==
==58838== Use --track-origins=yes to see where uninitialised values come from
==58838== For lists of detected and suppressed errors, rerun with: -s
==58838== ERROR SUMMARY: 72 errors from 3 contexts (suppressed: 0 from 0)
```

Error 1: Conditional jump depends on uninitialized value(s)

```
==58838== Conditional jump or move depends on uninitialised value(s)
==58838==    at 0x109181: main (in /home/kam/Documents/school/eecs_678/eecs678-debuggers-lab/debuggers/valgrind_test)
```

This is clearly because the inconspicuously named variable ‘uninitialized_variable’ is indeed uninitialized when it is first used.

```c
int main() {
  int uninitialized_variable; // This variable is never given a value.

  for (; uninitialized_variable < 100; uninitialized_variable++) {
    void** definitely_lost = (void**) malloc(sizeof(void*)); // allocate a
                                                             // pointer on the
```

Easy fix: we initialize it in the for loop.

```c
  int uninitialized_variable; // This variable is never given a value.

  for (uninitialized_variable = 0; uninitialized_variable < 100; uninitialized_variable++) {
    void** definitely_lost = (void**) malloc(sizeof(void*)); // allocate a
                                                             // pointer on the
```

Error 2: Definitely lost memory

```c
  for (uninitialized_variable = 0; uninitialized_variable < 100; uninitialized_variable++) {
    void** definitely_lost = (void**) malloc(sizeof(void*)); // allocate a
                                                             // pointer on the
                                                             // heap.

    *definitely_lost = (void*) malloc(7); // Give the pointer something else to
                                          // point to on the heap. This will be
                                          // indirectly lost.
```

We’re malloc’ing a double pointer and then malloc’ing 7 bytes to the pointer definitely_lost is pointing to.

Easy fix, after using those pointers for whatever reason, free them.

```
for (uninitialized_variable = 0; uninitialized_variable < 100; uninitialized_variable++) {
  void** definitely_lost = (void**) malloc(sizeof(void*)); // allocate a
                                                           // pointer on the
                                                           // heap.

  *definitely_lost = (void*) malloc(7); // Give the pointer something else to
                                        // point to on the heap. This will be
                                        // indirectly lost.

  free(*definitely_lost);
  free(definitely_lost);
}
```

Error 3/4: still reachable and possibly lost memory

```
still_reachable = malloc(42); // This value is never freed but is pointed to
                              // in the global scope at program completion.

free(still_reachable);

possibly_lost = malloc(10);
possibly_lost += 4; // This is similar to still reachable except there is a
                    // pointer pointing to the middle of the allocated block
                    // but nothing points to the front of the block. This is
                    // very odd behavior and usually is a memory leak (but not
                    // always).

possibly_lost -= 4; // or somehow reset possibly_lost to the beginning of the memory block
free(possibly_lost);
```

Same thing, just free those mallocs when you're done using them. With possibly_lost, you need to reset it to the beginning of the memory block somehow so you can free the whole thing.

Shazam

```
==60055== Memcheck, a memory error detector
==60055== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==60055== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==60055== Command: ./valgrind_test
==60055==
==60055==
==60055== HEAP SUMMARY:
==60055==     in use at exit: 0 bytes in 0 blocks
==60055==   total heap usage: 202 allocs, 202 frees, 1,552 bytes allocated
==60055==
==60055== All heap blocks were freed -- no leaks are possible
==60055==
==60055== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```