This reading discusses some details of pointers and safe memory access. We hope to help you avoid common crashes and bugs in your code.

You should refer back to the lecture on heap memory, which gave an example of an uninitialized variable producing an unpredictable result. Let's go over a few definitions and concepts first.

Segmentation fault (Segfault)

Various kinds of programming bugs related to pointers and memory can cause your program to crash. On Linux, if you dereference an address that you shouldn't, this is often called "segmentation fault," or "segfault." For example, if you dereference a pointer that is set to nullptr, it will almost certainly cause a segfault and crash immediately. This code will segfault:

```
// This code compiles successfully, runs, and CRASHES with a segfault!
int* n = nullptr;
std::cout << *n << std::endl;
```

On other operating systems, the error message may say something different. *Some* C++ compilers may respond to this in other unpredictable ways, but this is one type of error that has a fairly reliable symptom. It is actually an example of what we talk about in the next paragraph.

Undefined behavior

Sometimes it's possible to write a program that compiles, but that is not really a safe or valid program. There are some improper ways to write C++ that are not strictly forbidden by the C++ standard, but the C++ standard doesn't define how compilers are supposed to handle those situations, so your compiler may generate a program that works, or not! This is called "**undefined behavior**." Many beginning programmers make the mistake of thinking that just because their program compiles and runs for them on their system, that it must be valid and safe code. **This isn't true.** "It works for me" isn't an excuse, so be sure to proofread your code, use safe practices, and avoid relying on undefined behavior.

Many times, undefined behavior is caused by the careless use of uninitialized variables. Let's talk about initialization.

Initialization