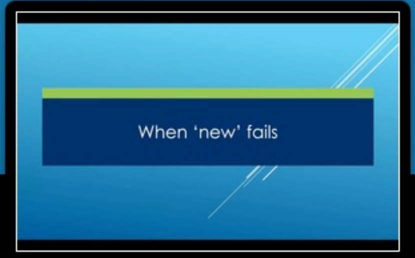
When ‘new’ fails

In some rare cases, the ‘new’ operator will fail to allocate dynamic memory from the heap. When that happens, and you have no mechanism in place to handle that failure, an exception will be thrown and your program will crash. BAD!


When ‘new’ fails

Simulating memory allocation failure

```cpp
//Try to allocate a insanely huge array in one go
//Unhandled new failure : Crash
int* lots_of_ints1 { new int[1000000000000000000] }; // May give an error about
                                                    // exceeding array size.


//Use a huge loop to try and exhaust the memory capabilities
//of your system. When new fails, your program is going to
//forcefuly terminate.

for (size_t i{} ; i < 100000000000 ; ++i){
     int* lots_of_ints2 { new int[10000000] };
}
```

Checking for failed memory allocations

- Through the exception mechanism
- The std::nothrow setting

The exception mechanism

```
//Handling the exception

//Handle the problem in a way that makes sense for your application.
//For example if you were trying to allocate memory to store color information
//for your application, and allocation fails, you could opt for showing some
//feedback message to the user, and rendering your app in black/white.
for (size_t i{} ; i < 100000000000 ; ++i){
    try{
        int* lots_of_ints3 { new int[10000000]};
    }catch(std::exception& ex){
        std::cout << "Cought exception ourselves : " << ex.what() << std::endl;
    }
}
```

std::nothrow

```
//std::nothrow : ideal if you don't want exceptions thrown when
//new fails

for (size_t i{} ; i < 1000000000000 ; ++i){
    int* lots_of_ints4 { new(std::nothrow) int[10000000] };

    if(lots_of_ints4 == nullptr){
        //Don't try to dereference and use lots_of_ints4 in here.
        //you'll get UB. No memory has really been allocated here
        //It failed and returned nullptr because of the std::nothrow setting
        std::cout << "Memory allocation failed" << std::endl;
    }else{
        std::cout << "Memory allocation succeeded" << std::endl;
    }

}
```