

## Forum:[Hack Forums](#)

Thread: [Useful trick or very sinful? Restrict template types by using forward declaration?](#)

**Goodies** - September 2, 2016, 3:24 pm

Obviously, you cannot create a definition for a template class/function in a header file and put the implementation in its own .cpp file as-is. There are ways around that, like creating an \_imp file and including that in the header file itself so the implementation is actually there.

But I found another (clever) way and I'd never seen it before.

Say I had a simple wrapper for random bytes. This is basically what I'm working with.

a.hpp file:

PHP Code

```
<?php
```

```
#pragma once // i know...
```

```
template<class gen_t>
```

```
class Random{
```

```
protected:
```

```
    std::random_device m_rdev;
```

```
    // random device by instance
```

```
    std::shared_ptr<gen_t> m_prng;
```

```
    // shared_ptr so other people can retrieve it.
```

```
public:
```

```
    Random();
```

```
    // initialize the prng with a seed
```

```
    void seed();
```

```
// when called, generates a random seed for the MT PRNG using std::random_device and seeds m_
prng with it.
```

```
    std::shared_ptr<gen_t> getprng(void) const;
```

```
    // returns exactly what m_prng.
```

```
};
```

PHP Code

```
<?php
```

```
#include "a.hpp"
```

```
template<class gen_t>
```

```
Random<gen_t>::Random() : m_prng(new gen_t){
```

```
    this->seed();
```

```
}
```

```
template<class gen_t>
```

```
void Random<gen_t>::seed() {
```

```
    std::array<uint32_t, gen_t::state_size> seed;
```

```
    std::generate(std::begin(seed), std::end(seed), std::ref(m_rdev));
```

```
    std::seed_seq seq(std::cbegin(seed), std::cend(seed));
```

```
    m_prng->seed(seq); // seed the instance of the MT RNG.
```

```
}
```

```
template<class gen_t>
std::shared_ptr<gen_t> Vaniche::Random<gen_t>::getprng() const {
    return m_prng;
}
```

Two things are true here: Anyone can provide any type in the template...

But also it will not compile....

So, say I want to only support the two standard mersenne twister types.

```
std::mt19937
std::mt19937_64
```

I can add this to the bottom of my .cpp implementation file:

PHP Code

```
<?php
```

```
template class Random<std::mt19937>;
template class Random<std::mt19937_64>;
```

And now I can instantiate one of those two in another cpp file.

Main.cpp

PHP Code

```
<?php
```

```
#include "a.hpp"
#include "customEngine.hpp" // provides SomeCustomEngine
```

```
int main(int argc, char **argv){
    Random<std::mt19937> r32; // sure
    Random<std::mt19937_64> r64; // fine
    Random<SomeCustomEngine> rCustom; // compiler error... has not been forwardly declared.
}
```

This takes care of the two problems:

- ~ I can now legally put a template header in one file and its definition in another
- ~ I can restrict the types so I don't need to worry about strange behavior when using other types.

Is this good?

Is this common?

Is this completely unnecessary?

Is this a bad practice?

**Derpy Hooves ?** - September 2, 2016, 8:17 pm

There's three common methods I see used when creating templated classes.

1. put it all in the header.
2. separate into two files and include the implementation at the bottom of the header.
3. separate into two files and explicitly instantiate templates in implementation file(what you did).

All three are legitimate supported options.

I usually just go with the first one since it's a lot less of a headache later on down the line.

The second option works the same as the first, but is there if you feel like you really need separate files.

The third option could be annoying later on; If you are working on a larger project and you need to use a templated class with a new type, it's easier if you don't have to hunt down all of the affected classes and add explicit instantiations.

I would only go with the third option if I was absolutely sure that would only use the class with specific types (like in your case).

tl;dr what you did is fine.

**Goodies** - September 2, 2016, 8:36 pm

(Sep 2, 2016 08:17 PM)Derpy Hooves ? Wrote: [\[CITE\]](#) There's three common methods I see used when creating templated classes.

1. put it all in the header.
2. separate into two files and include the implementation at the bottom of the header.
3. separate into two files and explicitly instantiate templates in implementation file(what you did).

All three are legitimate supported options.

I usually just go with the first one since it's a lot less of a headache later on down the line.

The second option works the same as the first, but is there if you feel like you really need separate files.

The third option could be annoying later on; If you are working on a larger project and you need to use a templated class with a new type, it's easier if you don't have to hunt down all of the affected classes and add explicit instantiations.

I would only go with the third option if I was absolutely sure that would only use the class with specific types (like in your case).

tl;dr what you did is fine.

I ended up going with option two, but it's still useful to know.

Thanks.