

## Template Headers / Source

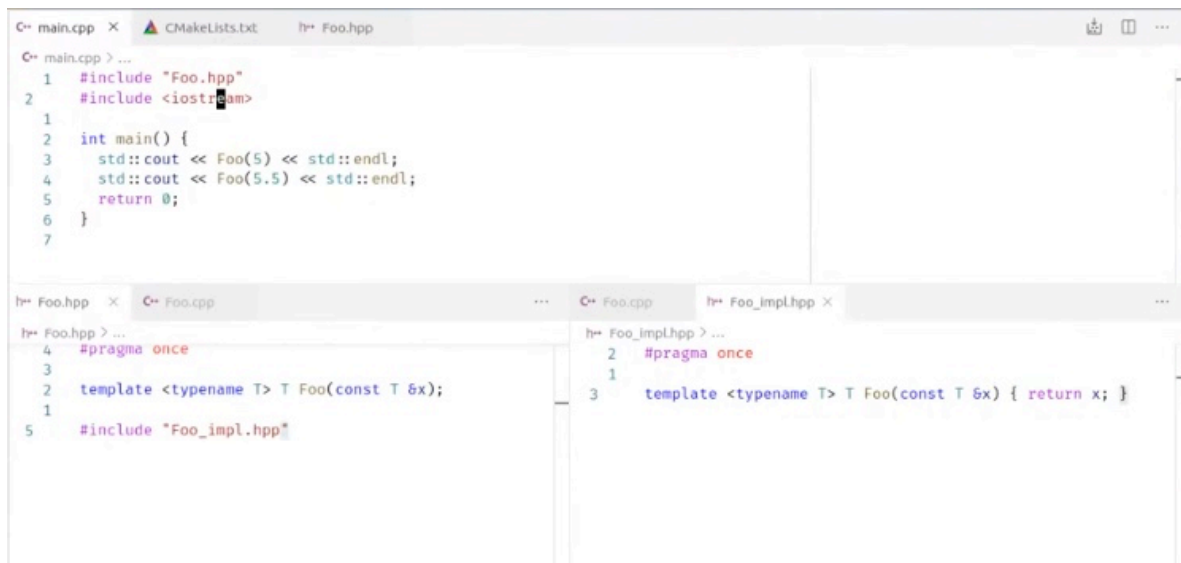
- Concrete templates are instantiated at compile time.
- Linker does not know about implementation.
- There are three options for template classes:
  - Declare and define in the header files



```
C++ main.cpp
9 // @file      main.cpp
8 // @author    Ignacio Vizzo    [ivizzo@uni-bonn.de]
7 //
6 // Copyright (c) 2019 Ignacio Vizzo, all rights reserved
5 #include <iostream>
4
3 #include "Foo.hpp"
2
1 int main() {
10     std::cout << Foo(5) << std::endl;
1     std::cout << Foo(5.5) << std::endl;
2     return 0;
3 }
4

Foo.hpp Foo.hpp/📄 Foo
2 #pragma once
1
3 template <typename T> T Foo(const T &x) { return x; }
```

- Declare in NAME.hpp file, implement in NAME\_impl.hpp file, add **#include** <NAME\_impl.hpp> in end of NAME.hpp



```
C++ main.cpp x CMakeLists.txt Foo.hpp
C++ main.cpp > ...
1 #include "Foo.hpp"
2 #include <iostream>
1
2 int main() {
3     std::cout << Foo(5) << std::endl;
4     std::cout << Foo(5.5) << std::endl;
5     return 0;
6 }
7

Foo.hpp x C++ Foo.cpp Foo_impl.hpp x
Foo.hpp > ...
4 #pragma once
3
2 template <typename T> T Foo(const T &x);
1
5 #include "Foo_impl.hpp"

C++ Foo.cpp
2 #pragma once
1
3 template <typename T> T Foo(const T &x) { return x; }
```

- Declare in \*.hpp file, implement in \*.cpp file, in the end of the \*.cpp add explicit instantiation for types you expect to use.

```
C++ main.cpp
1  #include "Foo.hpp"
2  #include <iostream>
1
2  int main() {
3      std::cout << Foo(5) << std::endl;
4      std::cout << Foo(5.5) << std::endl;
5      return 0;
}

Foo.hpp
3  #pragma once
2
1  template <typename T> T Foo
   (const T &x);
4

Foo.cpp Foo.cpp/...
2  #include "Foo.hpp"
1
3  template <typename T> T Foo(const T &x) { return x; }
1
2  template int Foo<int>(const int &x);
3  template double Foo<double>(const double &x);
```