# How to define a template class in a .h file and implement it in a .cpp file

Febil Chacko Thanikal



22 Dec 2009 CPOL

This article describbes how to define a template class in a .h file and do its implementation in a .cpp file.

## Introduction

This article suggests three methods to implement template classes in a *.cpp* file.

## Background

The common procedure in C++ is to put the class definition in a C++ header file and the implementation in a C++ source file. Then, the source file is made part of the project, meaning it is compiled separately. But when we implement this procedure for template classes, some compilation and linking problems will arise.

### Compilation Issue

Here is some sample code:



```
// TestTemp.h
#ifndef _TESTTEMP_H_
#define _TESTTEMP_H_

template<class T>
class TestTemp
{
public:
    TestTemp();
    void SetValue( T obj_i );
    T Getalue();

private:

    T m_Obj;
};
#endif

// TestTemp.cpp
#include "TestTemp.h"

TestTemp::TestTemp()
{
}
void TestTemp::SetValue( T obj_i )
{

}
T TestTemp::Getalue()
{
    return m_Obj;
}
```

If you try to implement the template class like a normal class implementation as shown above, it will generate a set of compilation errors such as:

Hide Shrink ▲ Copy Code

```
: error C2955: 'TestTemp' : use of  class template requires template argument list
: error C2065: 'T' : undeclared identifier
```

### Reason

In this case, the compiler doesn't know about the object type. So it will not compile.

### Solution

To compile this class without any errors, you need to put the template specific declaration in a *.cpp* file, as shown below:

***Template Class Header File***

Hide Shrink ▲ Copy Code

```
// TestTemp.h
#ifndef _TESTTEMP_H_
#define _TESTTEMP_H_

template<class T>
class TestTemp
{

public:

    TestTemp();
    void SetValue( T obj_i );
    T Getalue();

private:

    T m_Obj;
};
#endif
```

***Template Class Source File***

Hide Shrink ▲ Copy Code

```
// TestTemp.cpp
#include "TestTemp.h"

template <class T>
TestTemp<T>::TestTemp()
{
}
template <class T>
void TestTemp<T>::SetValue( T obj_i )
{

}
template <class T>
T TestTemp<T>::Getalue()
{
    return m_Obj;
}
```

## Linking Issue

With the above code, after resolving all the compilation errors, you may get some link errors while you create an object of this class in any file other than *TestTemp.cpp*. Here is some sample code:

***Client Source File***

Hide Shrink ▲ Copy Code

```
// Client.cpp
#include "TestTemp.h"

    :
    TestTemp<int> TempObj;
    :
```

***Link Error***

Hide Shrink ▲ Copy Code

```
: error LNK2001: unresolved external symbol "public: __thiscall
TestTemp<int>::TestTemp<int>(void)"
(??0?$TestTemp@H@@QAE@XZ)
```

### Reason

When the compiler encounters a declaration of a `TestTemp` object of some specific type, e.g., `int`, it must have access to the template

implementation source. Otherwise, it will have no idea how to construct the `TestTemp` member functions. And, if you have put the implementation in a source (*TestTemp.cpp*) file and made it a separate part of the project, the compiler will not be able to find it when it is trying to compile the client source file. And, `#include`ing the header file (*TestTemp.h*) will not be sufficient at that time. That only tells the compiler how to allocate for the object data and how to build the calls to the member functions, not how to build the member functions. And again, the compiler won't complain. It will assume that these functions are provided elsewhere, and leave it to the linker to find them. So, when it's time to link, you will get "unresolved references" to any of the class member functions that are not defined "inline" in the class definition.

### Solution

There are different methods to solve this problem. You can select from any of the methods below depending on which is suitable for your application design.

## Mehtod 1

You can create an object of a template class in the same source file where it is implemented ( *TestTemp.cpp*). So, there is no need to link the object creation code with its actual implementation in some other file. This will cause the compiler to compile these particular types so the associated class member functions will be available at link time. Here is the sample code:

***Template Class Header File***

Hide Shrink ▲ Copy Code

```
// TestTemp.h
#ifndef _TESTTEMP_H_
#define _TESTTEMP_H_
template<class T>
class TestTemp
{
public:
    TestTemp();
    void SetValue( T obj_i );
    T Getalue();

private:
    T m_Obj;
};
#endif
```

***Template Class Source File***

Hide Shrink ▲ Copy Code

```
// TestTemp.cpp
#include "TestTemp.h"

template <class T>
TestTemp<T>::TestTemp()
{
}

template <class T>
void TestTemp<T>::SetValue( T obj_i )
{
}

template <class T>
T TestTemp<T>::Getalue()
{
    return m_Obj;
}

// No need to call this TemporaryFunction() function,
// it's just to avoid link error.
void TemporaryFunction ()
{
    TestTemp<int> TempObj;
}
```

***Client Source File***

Hide Shrink ▲ Copy Code

```
// Client.cpp
#include "TestTemp.h"

    :
        TestTemp<int> TempObj;
        TempObj.SetValue( 2 );
        int nValue = TempObj.Getalue();
    :
```

The temporary function in "*TestTemp.cpp*" will solve the link error. No need to call this function because it's global.

## Method 2

You can `#include` the source file that implements your template class in your client source file. Here is the sample code:

***Template Class Header File***

Hide Shrink ▲ Copy Code

```
// TestTemp.h
#ifndef _TESTTEMP_H_
#define _TESTTEMP_H_

template<class T>
class TestTemp
{
public:
    TestTemp();
    void SetValue( T obj_i );
    T Getalue();
private:
    T m_Obj;
};
#endif
```

***Template Class Source File***

Hide Shrink ▲ Copy Code

```
// TestTemp.cpp
#include "TestTemp.h"

template <class T>
TestTemp<T>::TestTemp()
{
}

template <class T>
void TestTemp<T>::SetValue( T obj_i )
{
}

template <class T>
T TestTemp<T>::Getalue()
{
   return m_Obj;
}
```

***Client Source File***

Hide Shrink ▲ Copy Code

```
// Client.cpp
#include "TestTemp.h"
#include "TestTemp.cpp"
          :
        TestTemp<int> TempObj;
        TempObj.SetValue( 2 );
        int nValue = TempObj.Getalue();
              :
```

## Method 3

You can `#include` the source file that implements your template class (*TestTemp.cpp*) in your header file that defines the template class (*TestTemp.h*), and remove the source file from the project, not from the folder. Here is the sample code:

***Template Class Header File***

Hide Shrink ▲ Copy Code

```
// TestTemp.h
#ifndef _TESTTEMP_H_
#define _TESTTEMP_H_
template<class T>
class TestTemp
{
public:
    TestTemp();
    void SetValue( T obj_i );
    T Getalue();
private:
    T m_Obj;
};
#include "TestTemp.cpp"

#endif
```

***Template Class Source File***

Hide Shrink ▲ Copy Code

```cpp
// TestTemp.cpp
#include "TestTemp.h"

template <class T>
TestTemp<T>::TestTemp()
{
}
template <class T>
void TestTemp<T>::SetValue( T obj_i )
{
}

template <class T>
T TestTemp<T>::Getalue()
{
    return m_Obj;
}
```

***Client Source File***

Hide Shrink ▲ Copy Code

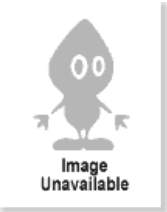
```cpp
// Client.cpp
#include "TestTemp.h"
                :
    TestTemp<int> TempObj;
    TempObj.SetValue( 2 );
    int nValue = TempObj.Getalue();
                :
```

# License



# Share

# About the Author




**Febil Chacko Thanikal**
Software Developer
India

No Biography provided

# Comments and Discussions

You must Sign In to use this message board.

Search Comments

First Prev Next

**Linking Error Solution is not working**
Ujjwal Gangwal 26-Nov-19 6:25

**Nicely written article.**
Benktesh Sharma 31-Jul-19 2:44

**Why remove from project in Method 3?**

Member 13272964 5-Oct-18 4:47

**This works in Visual C++**
mla154 22-May-18 5:09

**Method 1 is the best as it will optimize the build .**
patoria 6-Mar-18 23:13

**Google says don't separate into forward declarations.**
Chal McCollough 14-Jun-17 19:33

**Situation where Method 2 works but Method 3 fails on G++ 4.9.3 on cygwin**
Member 11872238 29-Jul-15 8:18

Re: Situation where Method 2 works but Method 3 fails on G++ 4.9.3 on cygwin
Member 11872238 29-Jul-15 8:40

**error**
Member 11642792 24-Jul-15 10:17

**Method 4 (similar to method 2)**
ddCubinator 8-Jul-15 3:42

**Beginner's question**
Member 11733572 1-Jun-15 8:42

**My vote of 5**
Member 11010766 13-Aug-14 5:49

**[My vote of 2] Please use the right vocabulary**
feraudy 3-May-14 2:10

**My vote of 5**
Member 9870583 11-Jun-13 12:30

**And an elegant architecture for organizing C++ header files of template classes**
Khaari 15-Apr-13 20:03

**Method 4 (Another method)**
Siddharth Hegde 19-Dec-12 2:09

Re: Method 4 (Another method)
Khaari 6-Apr-13 23:00

**Method 3 requires inline**
E.Hyde 5-Sep-12 0:59

**Nice article**
rajajay82 18-Jan-10 1:04

**I don't like those solutions**
Odys! 23-Dec-09 13:00

**Some comments to your ideas:**
dtr22 23-Dec-09 9:59

Re: Some comments to your ideas:
Febil Chacko Thanikal 23-Dec-09 16:45

**I don't like any of these solutions.**
millerize 23-Dec-09 7:04

Re: I don't like any of these solutions.
rxantos 6-Mar-16 23:02

**A simple doubt**
Adam Roderick J 22-Dec-09 22:34