Asked 3 years, 8 months ago Modified 3 years, 8 months ago Viewed 2k times

I have a class template in myclass.hpp:



```
template<class T, class P>
```



```
class myclass
};
```

In my main.cc I create an object of the class:

```
myclass<int, double> mc;
otherfunc<myclass>(mc);
```

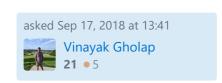
In some other header file header1.hpp:

```
template<class MyClass>
void otherfunc(MyClass const &mc)
/* Access through 'mc' the underlying template parameters T and P*/
```

How can I access the template parameter T and P in header1.hpp?

```
c++ templates class-template
```

Share Edit Follow Flag



Highest score (default)

\$

Sorted by:

otherfunc<myclass>(mc) is invalid with given otherfunc declaration. you might use otherfunc(mc) (let deduction occurs) or otherfunc<myclass<int, double>> (mc) . – Jarod42 Sep 17, 2018 at 13:46 🖍

3 Answers



How can I access the template parameter T and P in header1.hpp?



Provide public type definitions in your class myclass:



```
template<class T, class P>
class myclass
public:
    typedef T T_type;
    typedef P P_type;
};
```

Thus you can access those types as

```
typename myclass::T_Type x;
typename myclass::P_Type y;
```

elsewhere.

Share Edit Follow Flag



Example:



template<class T, class P> void otherfunc(myclass<T, P> const &mc) {}

using ParamP = typename MyClass::ParamP;

Alternatively:

template<class T, class P> class myclass using ParamT = T; using ParamP = P; **}**; template<class MyClass> void otherfunc(MyClass const &mc) using ParamT = typename MyClass::ParamT;

Share Edit Follow Flag



#1

}



One way is to typedef within myclass.



```
template<class T, class P>
class myclass
public:
    typedef T typeT;
    typedef P typeP;
};
```

And refer to them like

```
template<class MyClass>
void otherfunc(MyClass const &mc)
{
    typename MyClass::typeT myMember;
}
```

Another way is to use decltype. You likely don't literally need to use the template parameters, but intend to use the same type as a member or return value of a myclass member. Thus, something like this:

```
template<class T, class P>
struct myclass
 T memberT;
 P memberP;
};
template<class MyClass>
void otherfunc(MyClass const &mc)
 using T = decltype(MyClass::memberT);
 using P = decltype(MyClass::memberP);
 T var1;
 P var2;
}
```

Share Edit Follow Flag



35.1k • 13 • 77 • 137