

compile-Time Assertions

Template Metaprogramming I. Reference: Alexandrescu, MC++D Ch 2.

1. An implementation

```

template <bool b>
struct CompileTimeCheck
{
    CompileTimeCheck( ... ) ;
} ;

template<>
struct CompileTimeCheck<false> {} ;

#define CTASSERT( expr, msg )
{
    struct ERROR_##msg {} ;
    (void) sizeof( CompileTimeCheck<expr>( ERROR_##msg() ) ) ;
}
    
```

Usage:

```

template<typename T>
void babble( T t )
{
    CTASSERT( is_pointer<T>::value, Babble_requires_pointer ) ;
    // ...
    cout << *t << endl ;
}
    
```

So `babble("hey")` will compile fine, but `babble(3.14)` will fail, with a message such as:
 Error E2031 TraitsFun.cpp 107: Cannot cast from 'ERROR_Babble_requires_pointer' to 'CompileTimeCheck<0>' in function babble<double>(double)

2. How it works

When T is bound to `float`, using standard C preprocessor tricks, the line

```
CTASSERT( is_pointer<T>::value, Babble_requires_pointer ) ;
```

expands to

```

void babble( float t )
{
    {
        struct ERROR_Babble_requires_pointer {} ;
        (void) sizeof( CompileTimeCheck< false >( ERROR_Babble_requires_pointer() ) ) ;
    }
    // ...
    cout << *t << endl ;
}
    
```

This will not compile, because the class `CompileTimeCheck< false >` does not have a 1-argument (converting) constructor.

When T is bound to `char*`, the code *does* compile, because `CompileTimeCheck< true >` does have a 1-argument constructor. No constructor call needs to be made, however, since the `sizeof` expression is computed at compile-time. (So the linker does not need to look for the ctor body.)