```
Sticky Bits – Powered by Feabhas
                                                                                                  A blog looking at developing software
                                                                                                  for real-time and embedded systems
FEABHAS
 EMBEDDING SOFTWARE COMPETENCE
Home About this blog
← Templates and polymorphism
                                                                          Variadic templates →
                                                                                                                      Search
Template member functions
                                                                                                   Follow Us
Posted on July 24, 2014 by Glennan Carnie
                                     =
 Contents
                                                                                                  Categories
 1. Introduction
                                                                                                  Agile
 2. Template member functions
                                                                                                   ARM
                                                                                                  Build-systems
 3. Template functions on template classes
                                                                                                  C/C++ Programming
 4. Template constructors
                                                                                                   CMSIS
                                                                                                  Cortex
 5. Summary
                                                                                                   Design Issues
                                                                                                  General
Introduction
                                                                                                  Industry Analysis
                                                                                                   Linux
                                                                                                   Python
Previously we've looked at <u>template functions</u> and we've looked at <u>template classes</u>. This
                                                                                                   Python3
time, let's look at what happens when you combine them.
                                                                                                   RTOS
                                                                                                  Testing
                                                                                                   Toolchain
Template member functions
                                                                                                   training
                                                                                                  UML
                                                                                                   Uncategorized
A non-template class can have template member functions, if required.
                                                                                                   webinar
                                                                                                   Archives
 #include <iostream>
                                                                                                  February 2022
 using namespace std;

 October 2021

                                                                                                  September 2021
 class ADT
                                                                                                  August 2021
                                                                                                  July 2021
 public:
                                                                                                  June 2021
   template <typename T>
                                                                                                  May 2021
   void func(T val);
                                                                                                   January 2021
                                                                                                  November 2020
 template <typename T>

 October 2020

 void ADT::func(T val)
                                                                                                  August 2020
                                                                                                   April 2020
   cout << "Value is " << val << endl; // T must have overload for operator<<

 February 2020

                                                                                                   January 2020

 October 2019

  int main()
                                                                                                   September 2019
                                                                                                   July 2019
   ADT a1;
                                                                                                  May 2019
                                                                                                  April 2019
                      // Generates void ADT::func<int>(int val);
   a1.func(100);
                                                                                                  March 2019
                     // Generates void ADT::func<double>(double val);
   a1.func(17.6);
                                                                                                  ■ February 2019
                                                                                                   January 2019
                                                                                                   December 2018

 October 2018

                                                                                                   September 2018
Notice the syntax. Unlike a member function for a template class, a template member
                                                                                                   August 2018
function is just like a free template function but scoped to its containing class. In this case
                                                                                                   July 2018
                                                                                                   April 2018
the compiler generates a member function for each (unique) template type. As before, the
                                                                                                   February 2018
compiler will favour a non-template overload of a member function.
                                                                                                   January 2018
                                                                                                   December 2017
                                                                                                   November 2017
Template functions on template classes

 October 2017

                                                                                                   September 2017
What about if our class is itself a template? A template class will typically have member
                                                                                                   August 2017
functions defined in terms of its own template parameter, but may equally have member
                                                                                                   July 2017
                                                                                                   June 2017
functions that are themselves template functions. In this case the member function's
                                                                                                   May 2017
template parameter is different to the owning class'.
                                                                                                  April 2017
                                                                                                  March 2017
                                                                                                   February 2017
The syntax for specifying the template member function is rather obtuse (to say the least):
                                                                                                   January 2017
                                                                                                   December 2016
                                                                                                   November 2016
                                          Template member
 template<typename T>

 October 2016

 class Utility
                                                                                                   September 2016
                                           function
                                                                                                   August 2016
 public:
                                                                                                  July 2016
   void op(T elem);
                                                                                                   April 2016
   template<typename U> ←

 January 2016

   void func(U elem);
                                                                                                   December 2015
                                                                                                   November 2015

 October 2015

 template<typename T>
 void Utility<T>::op(T elem)
                                                                                                   September 2015
                                                                                                   August 2015
   // op()'s behaviour...
                                                                                                  July 2015
                                                                                                  June 2015
 template<typename T>
                                                                                                  May 2015
 template<typename U>

 January 2015

 void Utility<T>::func(U elem)
                                                                                                  December 2014
                                                                                                  November 2014
   // func()'s behaviour...

 October 2014

                                                                                                  September 2014
                     int main()
                                                                                                   August 2014
                      Utility<int> utility;
                                                                                                  July 2014
                       utility.func(100);
                                                                                                  June 2014
                       utility.func("Hello World");
                                                                                                  May 2014
                                                                                                  April 2014
                                                                                                  March 2014
                                                                                                   February 2014
The containing class' template declaration must be 'outer' one; func is a template function

 January 2014

                                                                                                   November 2013
(typename U) that belongs to the class Utility<T> (typename T)
                                                                                                  September 2013
                                                                                                   August 2013
                                                                                                   July 2013
Notice that the template function must have a different template parameter identifier to its
                                                                                                   June 2013
containing class (even if the same type is being used to instantiate both the class and the
                                                                                                   May 2013
function).
                                                                                                  April 2013
                                                                                                  • February 2013
                                                                                                   January 2013
Template constructors
                                                                                                   November 2012

 October 2012

                                                                                                   August 2012
A common application of template member functions on template classes is constructors;
                                                                                                   July 2012
particularly if inheritance is involved. (Have a look <u>here</u> for more on template inheritance)
                                                                                                   June 2012
                                                                                                  May 2012
                                                                                                  April 2012
Let's revisit a previous example.
                                                                                                   March 2012
                                                                                                  December 2011
                                                                                                   November 2011
 class Waypoint
                                                                                                   June 2011
                                                                                                  May 2011
 public:
   Waypoint(float lat = 0.0, float lon = 0.0)
                                                                                                   April 2011
                                                  Template parameter is the base class.
   void display();
                                                                                                  March 2011
                                                                                                  February 2011
  private:
                                                                                                  January 2011
   float longitude;
                                                                                                   December 2010
   float latitude;
                     template<typename T>
                                                                                                   November 2010
                     class Named : public T ←

 October 2010

                     public:
                                                                                                   September 2010
                       Named(const char* str) : name(str) {}
                                                                                                   August 2010
                       void display();
                                                                                                   July 2010
                                             Template
                                                                                                   June 2010
                     private:
                                                                                                  May 2010
                                             type must
                       string name;
                                                                                                  April 2010
                                             support
                                                                                                  March 2010
                                             display()
                     template<typename T>
                                                                                                  ■ February 2010
                     void Named<T>::display()
                                                                                                   January 2010
                       cout << name << " ";
                                                                                                   December 2009
                      T::display(); 	
                                                                                                   November 2009

 October 2009

                                                                                                   September 2009
   int main()
     Waypoint wp1;
     Named<Waypoint> wp2("Home");
     wp1.display();
     wp2.display();
Above is an example of parameterised inheritance. In the earlier article I (deliberately)
ignored the construction of the base class objects. The problem we face is that we cannot
determine the structure of the base class constructor – each base class constructor will
have its own number of parameters, with different types – so how can we write a Named
constructor that satisfies any potential base class?
Let's split this problem into two: parameters of different types; and different numbers of
parameters
We can deal with different parameter types by making the Named class constructor a
template member function.
 template<typename T>
 class Named : public T
  public:
   template<typename Arg1, typename Arg2>
   Named(const char* str, Arg1&& arg1, Arg2&& arg2);
   void display();
  private:
   string name;
 template<typename T>
 template<typename Arg1, typename Arg2>
  Named<T>::Named(const char* str, Arg1&& arg1, Arg2&& arg2) :
   T(std::forward<Arg1>(arg1), std::forward<Arg2>(arg2)),
   name(str)
                     Pass on parameter
                     unchanged
OK... deep breath... let's wade through this mire of syntax and work out what's happening.
The constructor for Named has three parameters: a string literal (const char*) and two
template parameter arguments. Notice these parameters are passed as <u>r-value references</u>
(Arg&&). In this situation Scott Meyers refers to these as Universal References – meaning
"a reference that can bind to anything".
In the body of the constructor, notice the use of the template function std::forward. This
function ensures that the types of the parameters are forwarded on without changing their
types – that is, l-values remain l-values, r-values remain r-values.
(I'm deliberately glossing over the details of these mechanisms here as they're not the
focus of this article. For a detailed description I highly recommend reading Meyers'
excellent article "Universal References in C++11"
Now when we construct a Named object, its constructor (function) can deduce the types of
the supplied arguments and pass them on to the (template-parameter) base class.
  class Waypoint
  public:
   Waypoint(float lat = 0.0, float lon = 0.0)
   void display();
  private:
   float longitude;
   float latitude;
  int main()
   Waypoint wp1;
   Named<Waypoint> wp2("Home", 1.30, 53.775);
   // Instantiates a constructor of the form:
   // Named<Waypoint>::Named(const char*, double&&, double&&)
   wp1.display();
   wp2.display();
So far, so good, but our Named constructor is still limited: It will only work for base classes
that have exactly two parameters. If we try and use a class with three parameters we get a
problem:
 class Lamp
 public:
   Lamp(char rm, int dev, bool onState);
   void on();
   void off();
   void display();
  private:
   char room;
   int device;
   bool isOn;
            int main()
              Lamp study;
             Named<Lamp> desk("Desk lamp", 'A', 1, true); // ERROR!
              study.display();
              desk.display();
                                            ERROR C2661: 'Lamp::Lamp' :
                                                           no overloaded function
                                                           takes 2 arguments
Our Lamp class is a candidate for being the Named base class, since it supports the display()
method. However, its constructor requires three parameters – and our template
constructor can only supply two.
The short-term fix is to overload the Named constructor to take three parameters:
 template<typename T>
 class Named : public T
  public:
   template<typename Arg1, typename Arg2>
   Named(const char* str, Arg1&& arg1, Arg2&& arg2);
   template<typename Arg1, typename Arg2, typename Arg3>
   Named(const char* str, Arg1&& arg1, Arg2&& arg2, Arg3&& arg3);
   void display();
  private:
   string name;
  int main()
   Lamp study;
   Named<Lamp> desk("Desk lamp", 'A', 1, true);
   // Instantiates a constructor of the form:
   // Named<Lamp>::Named(const char*, char&&, int&&, bool&&)
   study.display();
   desk.display();
This works; but it means we'll have to overload the constructor for zero parameters, one
parameter, two, three, four, etc. This can quickly become onerous, so in the next article
we'll have a look at a new mechanism in C++11 designed to make this more flexible –
Variadic Templates.
Summary
Template member functions allow us to parameterise functions independently of the class
they belong to. They can be added to both template and non-template classes.
Template member functions follow all the usual rules of template functions – they can be
overloaded (both by template versions and non-template versions) and they may be
overridden by derived classes.
If you'd like to know more about templates, and C++ programming – particularly for
embedded and real-time applications – visit the Feabhas website. You may find the
following of interest:
<u>C++-501 – C++ for Embedded Developers</u>
<u>C++-502 – C++ for Real-Time Developers</u>
AC++-501 – Advanced C++
AC++-401 - Advanced C++ for C++11
                 Glennan Carnie
                 Technical Consultant at Feabhas Ltd
                 Glennan is an embedded systems and software engineer with over 20 years experience,
                 mostly in high-integrity systems for the defence and aerospace industry.
                 He specialises in C++, UML, software modelling, Systems Engineering and process
                 development.
 Like (9) Dislike (1)
                           Glennan Carnie
                           Website | + posts
                          Glennan is an embedded systems and software engineer with over 20 years experience, mostly in high-integrity systems for the defence and aerospace industry.
                          He specialises in C++, UML, software modelling, Systems Engineering and process development.
```

Variadic templates →

This entry was posted in <u>C/C++ Programming</u> and tagged <u>C++</u>, <u>C++11</u>, <u>constructor</u>, <u>forwarding</u>, <u>functions</u>, <u>inheritance</u>,

member functions, Overloading, r-value, r-value references, Templates. Bookmark the permalink.

1 Response to Template member functions

Kranti Madineni says:

I have become a fan of yours.

Like (2) Dislike (0)

April 21, 2018 at 12:00 pm

f 💆 in 🔤

← Templates and polymorphism

Leave a Reply

Enter your comment here...