typeless functor

Bronek Kozicki
brok@spamcop.net
http://b.kozicki.pl/functor

The goal

- Encapsulate the function call without leaking type information (*parameters etc.*) and
- make it type-safe
  and
- make it very cheap to create, copy and execute

Design choices

- No heap allocation (cheap to create), but
- PODs only
  and
- size is an issue

How are we going to do it?

- Take the function/functor address (or the functor object itself, if it's a POD)
- take its arguments (if there are any)
- stuff it all into `char[X]` and make it a member of `function` class
- Encapsulate function call so that type information is recovered and control is passed to function/functor

But how?

1. Take *typed* data and operation you want to perform upon it
2. Produce two pointers from it:
   - Pointer to generic function that knows the type, casts `char*` back to actual type and performs the operation that we want
   - Pointer `char*` that contains the data
3. Pass then around, and call the function when you need it

Again, how?

We basically split *typed* data into two pieces of information:

- Pointer to function that *knows* the type, but does not know the actual data
- Pointer to data stripped from type information

This is what `pack` does; the function that *knows the type* is called `exec`

In other words, we take the piece of stack (*data stripped from type information*), carry it around *(copy the data as a bunch of bytes)* and *execute it* whenever we want (*function that knows the type puts it "back in" and does what we originally intended*). I will call it *deferring* type information, as type is *deferred* from compile time to runtime.

The term deferring *as used here has nothing to do with deferred function call.*

But is it legal? *Yes.*

And portable? *Absolutely, but size matters.*

And there is a limit – you can only do this with Plain Old Data.

Is there a workaround*? Use heap allocation instead of copying the object as a bunch of bytes*

How serious limitation is it? *All pointers are PODs anyway, and simple functors (your own wrappers) can also be PODs. They just* may not *own resources*

Demo

Basic idea: T0.cpp

… and little encapsulation: T1.cpp

… and arguments: T2.cpp

… handle those that take arguments and those that do not: T3.cpp

Scope guard

Goal: encapsulate function call and execute it at the end of the scope

Demo

Simple scope guard: T4.cpp

… exercise move semantics: T5.cpp

… an attempt to chain scope guards: T6.cpp

What if we actually keep type?

- No problem, we can keep *some* type information and *defer* the one we do not want. Example: T7.cpp
- We can also mix them both, i.e. generate typeless functor from typed one: T8.cpp, and vice-versa (exercise to the reader, see hints at the next slide)

Can we *access* the data…

… after we *deferred* its type information?

Yes. We just need one extra function pointer. Example: T9.cpp

Encapsulation is your friend: we replaced casts with union. Refactoring leads to T10.cpp, then T11.cpp

Can we handle *pointer-to-member-function* ?

Absolutely! We just make it an argument, so that:

```
f(a);
```

… is replaced with:

```
(f.*a)();
```

Example: T12.cpp and T13.cpp.

Caution, we have some metaprogramming here

Can we have it all?

Err … sure, look at T14.cpp . But it is not *really* everything.

Potential uses

- First of all – try to remember how *deferring type information* works, but use it scarcely. The library facility is not that important, as there might be more uses than I imagined;
- Try to use it where Boost.Any or Boost.Function (or tr1::function) are *not quite* right for your scenario

But what about the library?

- Improved Boost.Any?
- *Cheap* deleter support for unique_ptr, tr1::shared_ptr etc.? (BTW, look at http://www.kangaroologic.com/move_ptr/libs/move_ptr/doc/dynamic_move_ptr.html)
- Typeless functor?
- Scope guard?

I need your ideas. The right library utility is a balance between usability and complexity/cost that is difficult to get right without real-life use scenarios.

Where is the code?

http://b.kozicki.pl/functor

and on ACCU website

http://accu.org/index.php/conferences/2006

Questions?