Compiler aided optimization of the pimpl-idiom

Alexander Richardson (alr48@cam.ac.uk)

University of Cambridge

Tuesday 14\textsuperscript{th} April, 2015
Pimpl-idiom

- Used to keep binary compatibility in C++ libraries
- Heavily used by e.g. Qt and KDE
- **Problem**: requires extra memory allocations
Example

```
//foo.h
class Foo {
 public:
   Foo(const char* s);
   // ...
 private:
   FooPrivate* d;
};

// foo.cpp
class FooPrivate {
   // data members
}
Foo::Foo(const char* s) : d(new FooPrivate(s)) {}
```
Compiler aided optimization of the pimpl-idiom

Tuesday 14th April, 2015
Even more overhead with inheritance

Derived* foo = new Derived()

malloc()

Base::d | Derived::d

malloc()

BasePrivate

DerivedPrivate

malloc()
Derived* foo = new Derived()

malloc()
calculate_offset()
calculate_offset()
Solution

- One large `malloc()` call and then use `placement new`
- **Must** retain binary compatibility
- Could be done at the library level
  - Error-prone and hard to debug
  - Requires changing every `new` expression!
- **Better:** Let clang do the work for us
Solution

```cpp
//foo.h
class Foo {
  public:
    Foo(const char* s);
    // ...
  private:
    [[clang::pimpl]] FooPrivate* d; // only need to add one attribute
};

// foo.cpp
class FooPrivate {
  // data members
};
Foo::Foo(const char* s) : d(new FooPrivate(s)) {}
```
Solution

- Generate three static data members per class
  - `sizeof(private class)`
  - `alignof(private class)`
  - Total required allocation size (optimization)

- Generate extra constructor overloads
  - `Foo(int x)`
  - `Foo(int x, void* dpointer)`
  - If `dpointer` is non-null use placement new
    - Pass adjusted `dpointer` to base class constructor
  - Let original constructor delegate to new one and pass `nullptr` for the `dpointer` parameter

- Add custom `operator delete` to private class

- Replace every `new Foo(args)` expression by
  - `void ∗ buffer = ::operator new(Foo::totalSize);`
  - `Foo ∗ foo = new(buffer) Foo(args,buffer + sizeof(Foo) + align);`
Solution

- Generate three static data members per class
  - `sizeof(private class)`
  - `alignof(private class)`
  - Total required allocation size (optimization)
- Generate extra constructor overloads
  - `Foo(int x) → Foo(int x, void* dpointer)`
  - If `dpointer` is non-null use `placement new`
  - Pass adjusted `dpointer` to base class constructor
Solution

- Generate three static data members per class
  - `sizeof(private class)`
  - `alignof(private class)`
  - Total required allocation size (optimization)
- Generate extra constructor overloads
  - `Foo(int x) → Foo(int x, void* dpointer)`
  - If `dpointer` is non-null use `placement new`
  - Pass adjusted `dpointer` to base class constructor
- Let original constructor delegate to new one and pass `nullptr` for the `dpointer` parameter
Solution

- Generate three static data members per class
  - \texttt{sizeof(private class)}
  - \texttt{alignof(private class)}
  - Total required allocation size (optimization)
- Generate extra constructor overloads
  - \texttt{Foo(int x) \rightarrow Foo(int x, void* dpointer)}
  - If \texttt{dpointer} is non-null use \textit{placement new}
  - Pass adjusted \texttt{dpointer} to base class constructor
- Let original constructor delegate to new one and pass \texttt{nullptr} for the \texttt{dpointer} parameter
- Add custom operator \texttt{delete} to private class
Solution

- Generate three static data members per class
  - `sizeof(private class)`
  - `alignof(private class)`
  - Total required allocation size (optimization)
- Generate extra constructor overloads
  - `Foo(int x) → Foo(int x, void* dpointer)`
  - If `dpointer` is non-null use `placement new`
  - Pass adjusted `dpointer` to base class constructor
- Let original constructor delegate to new one and pass `nullptr` for the `dpointer` parameter
- Add custom operator delete to private class
- Replace every `new Foo(args)` expression by

```cpp
void* buffer = ::operator new(Foo::totalSize);
Foo* foo = new (buffer) Foo(args, buffer + sizeof(Foo) + align);
```
Conclusion

- Over 50% speedup in allocation-heavy benchmark
- Total memory usage reduced by about 3%
- Code at https://github.com/a-richardson/clang
- Questions → alr48@cam.ac.uk