HOW TO (IN)FIX YOUR CODE

Pascal Bormann

cenit

DRIVEN BY YOUR VISION.

ONCE UPON A TIME…

```
void Raycast(const Ray& ray,
             std::vector<RaycastHit>& oHits,
             bool propagateToChildren,
             bool findFirstHit,
             bool calculateHitData) const;
```

```
Raycast( ray, hits, true, true, false );
```

THE NIGHT IS DARK AND FULL OF BOOLEANS

```
enum class Flags {
    None,
    Propagate          = 0x01,
    FindFirstHit       = 0x02,
    CalculateHitData = 0x04
};


void Raycast(const Ray& ray,
             std::vector<RaycastHit>& oHits,
             Flags flags = Flags::None) const;
```

```
Raycast( ray, hits,
 Flags::Propagate | Flags::FindFirstHit );
```

```
if(flags == Flags::FindFirstHit){
    //...
}
```

```
if( (static_cast<int>(flags) &
     static_cast<int>(Flags::FindFirstHit))
    != 0 ){
   //...
}
```

LET’S WRITE SOME FUNCTIONS

```
template<typename Enum>
Enum Or( Enum l, Enum r ) {
   using T = typename std::underlying_type<Enum>::type;
   return static_cast<Enum>(
      static_cast<T>(l) | static_cast<T>(r) );
}
```

```
template<typename Enum>
bool Contains( Enum src, Enum dst) {
   using T = typename std::underlying_type<Enum>::type;
   return (static_cast<T>(src) & static_cast<T>(dst)) != 0;
}
```

```
Raycast(ray, hits,
  Or(Flags::Propagate, Flags::FindFirstHit));
```

```
if( Contains(flags, Flags::FindFirstHit) ){
   //...
}
```

VERY READABLE, THIS IS!

https://www.flickr.com/photos/isherwoodchris/6955395580

C++ PREFIX NOTATION

```
Contains(flags, Flags::FindFirstHit)
```

C++ PREFIX NOTATION

```
Contains(flags, Flags::FindFirstHit)
```

HASKELL PREFIX NOTATION

```
Contains flags FindFirstHit
```

C++ PREFIX NOTATION

```
Contains(flags, Flags::FindFirstHit)
```

HASKELL PREFIX NOTATION

```
Contains flags FindFirstHit
```

HASKELL INFIX NOTATION

```
flags `Contains` FindFirstHit
```

INFIX YOUR FUNCTIONS IN 2 EASY STEPS!

```
flags | Contains | Flags::FindFirstHit
```

INFIX YOUR FUNCTIONS IN 2 EASY STEPS!

BIND LEFT ARGUMENT AND FUNCTION
INTO A WRAPPER OBJECT

```
flags | Contains | Flags::FindFirstHit
```

INFIX YOUR FUNCTIONS IN 2 EASY STEPS!

BIND LEFT ARGUMENT AND FUNCTION
INTO A WRAPPER OBJECT

```
flags | Contains | Flags::FindFirstHit
```

INVOKE BY CALLING **operator|**
WITH RIGHT ARGUMENT

```
template<typename T, typename BinaryFunc>
struct InfixHelper
{
   InfixHelper( T&& left, BinaryFunc func ) :
      _left( std::forward<T>(left) ),
      _func( func ) {}

   template<typename U>
   inline decltype(auto) operator|( U&& other ) {
      return _func( _left, std::forward<U>(other) );
   }

   T _left;
   BinaryFunc _func;
};
```

OVERLOAD GLOBAL operator|

```
template<typename T, typename Func>
InfixHelper<T, Func>
   operator|( T&& left, Func func )
{
   return InfixHelper<T, Func>(std::forward<T>(left), func);
}
```

OVERLOAD GLOBAL `operator|`

THIS MIGHT CAUSE PROBLEMS
WITH OVERLOAD RESOLUTION...

```
template<typename T, typename Func>
InfixHelper<T, Func>
   operator|( T&& left, Func func )
{
   return InfixHelper<T, Func>(std::forward<T>(left), func);
}
```

OVERLOAD GLOBAL **operator|**

A TYPE WRAPPER FIXES THIS!

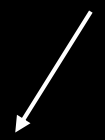
```
template<typename T, typename Func>
InfixHelper<T, Func>
   operator|( T&& left, Infix_t<Func> infix )
{
   return InfixHelper<T, Func>(
      std::forward<T>(left), infix._func );
}
```

```
template<typename BinaryFunc>
struct Infix_t
{
   explicit Infix_t( BinaryFunc func ) :
      _func( func ) {}

   BinaryFunc _func;
};


template<typename BinaryFunc>
Infix_t<BinaryFunc> Infix( BinaryFunc func )
{
   return Infix_t<BinaryFunc>( func );
}
```

```
auto flags = Flags::Propagate |
             Infix(Or<Flags>) |
             Flags::FindFirstHit;

if( flags | Infix(Contains<Flags>) |
    Flags::FindFirstHit) {
   //...
}
```

```
auto or = Infix(Or<Flags>);
auto contains = Infix(Contains<Flags>);
```

DO ONCE, STORE SOMEWHERE



```
auto or = Infix(Or<Flags>);
auto contains = Infix(Contains<Flags>);
```

DO ONCE, STORE SOMEWHERE

```
auto or = Infix(Or<Flags>);
auto contains = Infix(Contains<Flags>);

auto flags = Flags::Propagate | or |
             Flags::FindFirstHit;

if( flags | contains | Flags::FindFirstHit) {
   //...
}
```

GENERIC LAMBDAS = AWESOME

```
auto find = Infix( [](auto& c, auto&& val )
{
   return std::find(std::begin(c), std::end(c), val);
} );

std::vector<int> numbers = { 1, 2, 3, 4, 5, 6 };

auto four = numbers | find | 4;

std::cout << *four << std::endl;
```

THANKS FOR LISTENING!

p.bormann@cenit.de