How to make your code swifter with Objective-C++

Objective-C++

Type inference & constants

```
if  let document = self.document  {
    [self loadDocument:document]
}
```

Objective-C++

Type inference & constants

```
#define let const auto

if (let document = self.document) {
    [self loadDocument:document];
}
```

Objective-C++

Type inference & constants

```
#define let const auto

if (let document = self.document) {
    [self loadDocument:document];
}
```

Objective-C++

Type inference & constants

```
#define let const auto

if (let document = self.document) {
    [self loadDocument:document];
}
```

Objective-C++

Type inference

```
void(^handler)(NSData *,
               NSURLResponse *,
                     NSError *) = ^(
                        NSData *data,
                   NSURLResponse *response,
                         NSError *error) {
  // some handler
};
```

Objective-C++

Type inference

```
auto handler = ^(NSData *data,
                 NSURLResponse *response,
                       NSError *error) {
  // some handler
};
```

Objective-C++

Boxing

```
std::vector<CGFloat> ratios;
ratios.reserve(sampleSize);
for (int idx = 0; idx < max; idx += max/sample) {
  CGFloat r = <someCalculation>;
  ratios.push_back(r);
}
std::sort(ratios.begin(), ratios.end());
middleRatio = ratios[ratios.size() / 2];
```

Objective-C++

Boxing

```
std::vector<CGFloat> ratios;
ratios.reserve(sampleSize);
for (int idx = 0; idx < max; idx += max/sample) {
  CGFloat r = <someCalculation>;
  ratios.push_back(r);
}
std::sort(ratios.begin(), ratios.end());
middleRatio = ratios[ratios.size() / 2];
```

Objective-C++

Boxing

```
std::vector<CGFloat> ratios;
ratios.reserve(sampleSize);
for (int idx = 0; idx < max; idx += max/sample) {
  CGFloat r = <someCalculation>;
  ratios.push_back(r);
}
std::sort(ratios.begin(), ratios.end());
middleRatio = ratios[ratios.size() / 2];
```

Objective-C++

Boxing

```
std::vector<CGFloat> ratios;
ratios.reserve(sampleSize);
for (int idx = 0; idx < max; idx += max/sample) {
  CGFloat r = <someCalculation>;
  ratios.push_back(r);
}
std::sort(ratios.begin(), ratios.end());
middleRatio = ratios[ratios.size() / 2];
```

Objective-C++

Boxing

```
std::vector<CGFloat> ratios;
ratios.reserve(sampleSize);
for (int idx = 0; idx < max; idx += max/sample) {
  CGFloat r = <someCalculation>;
  ratios.push_back(r);
}
std::sort(ratios.begin(), ratios.end());
middleRatio = ratios[ratios.size() / 2];
```

Objective-C++

Boxing

```
std::vector<CGFloat> ratios;
ratios.reserve(sampleSize);
for (int idx = 0; idx < max; idx += max/sample) {
  CGFloat r = <someCalculation>;
  ratios.push_back(r);
}
std::sort(ratios.begin(), ratios.end());
middleRatio = ratios[ratios.size() / 2];
```

Objective-C++

Boxing

```
std::vector<CGFloat> ratios;
ratios.reserve(sampleSize);
for (int idx = 0; idx < max; idx += max/sample) {
  CGFloat r = <someCalculation>;
  ratios.push_back(r);
}
std::sort(ratios.begin(), ratios.end());
middleRatio = ratios[ratios.size() / 2];
```

Objective-C++

Boxing

```
std::vector<CGFloat> ratios;
ratios.reserve(sampleSize);
for (int idx = 0; idx < max; idx += max/sample) {
  CGFloat r = <someCalculation>;
  ratios.push_back(r);
}
std::sort(ratios.begin(), ratios.end());
middleRatio = ratios[ratios.size() / 2];
```

Objective-C++

Operators

```
CGSize operator/(const CGSize &lhs, CGFloat f) {
    return CGSize{ lhs.width / f, lhs.height / f };
}

CGSize zoomSize = self.bounds.size / zoomScale;
```

Objective-C++

Operators

```
CGSize operator/(const CGSize &lhs, CGFloat f) {
    return CGSize{ lhs.width / f, lhs.height / f };
}

CGSize zoomSize = self.bounds.size / zoomScale;
```

Objective-C++

Locks

```
@interface MyObject () {
  std::mutex _myLock;
}
@end

@implementation MyObject
- (void)doSomething {
  std::lock_guard<std::mutex> lock(_myLock);
  // do stuff
}
@end
```

Objective-C++

Locks

```
@interface MyObject () {
  std::mutex _myLock;
}
@end

@implementation MyObject
- (void)doSomething {
  std::lock_guard<std::mutex> lock(_myLock);
  // do stuff
}
@end
```

Objective-C++

Locks

```
@interface MyObject () {
  std::mutex _myLock;
}
@end

@implementation MyObject
- (void)doSomething {
  std::lock_guard<std::mutex> lock(_myLock);
  // do stuff
}
@end
```