Developer Tools

#WWDC17

What's New in LLVM

Session 411

Devin Coughlin, Program Analysis Engineer
Duncan Exon Smith, Clang Frontend Manager

Agenda

API Availability Checking for Objective-C

Static Analyzer Checks

New Warnings

C++ Refactoring

Features from C++17

Link-Time Optimization

API Availability Checking

Important to Adopt New APIs

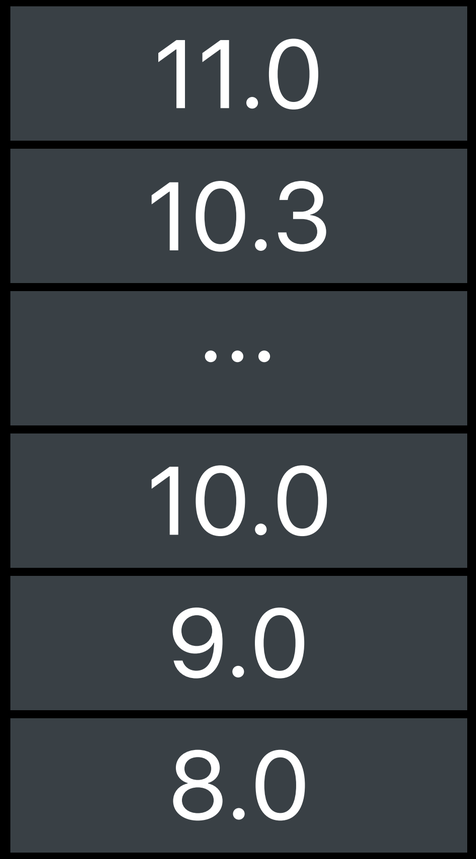
Every OS release comes with great new APIs

Customers expect you to adopt

Still have to support users on older OSes

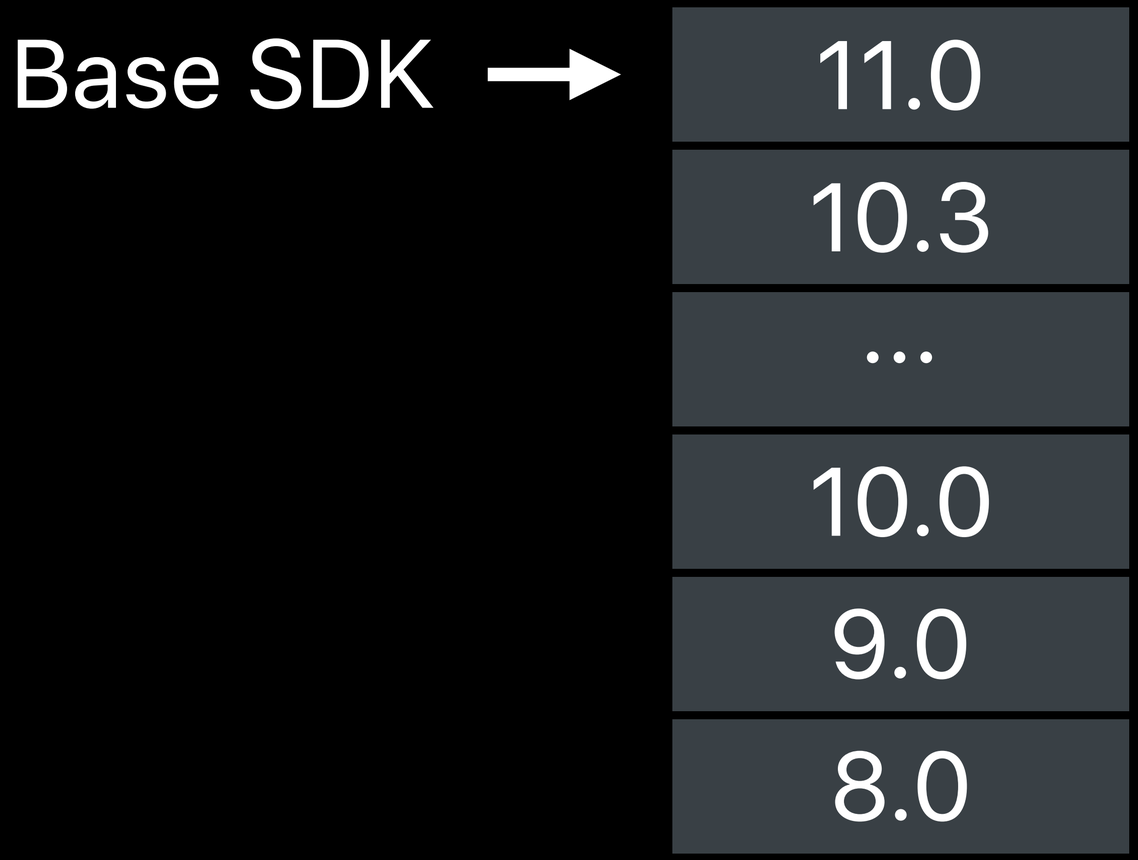
Base SDK and Deployment Target

Example: Supporting Multiple iOS Releases



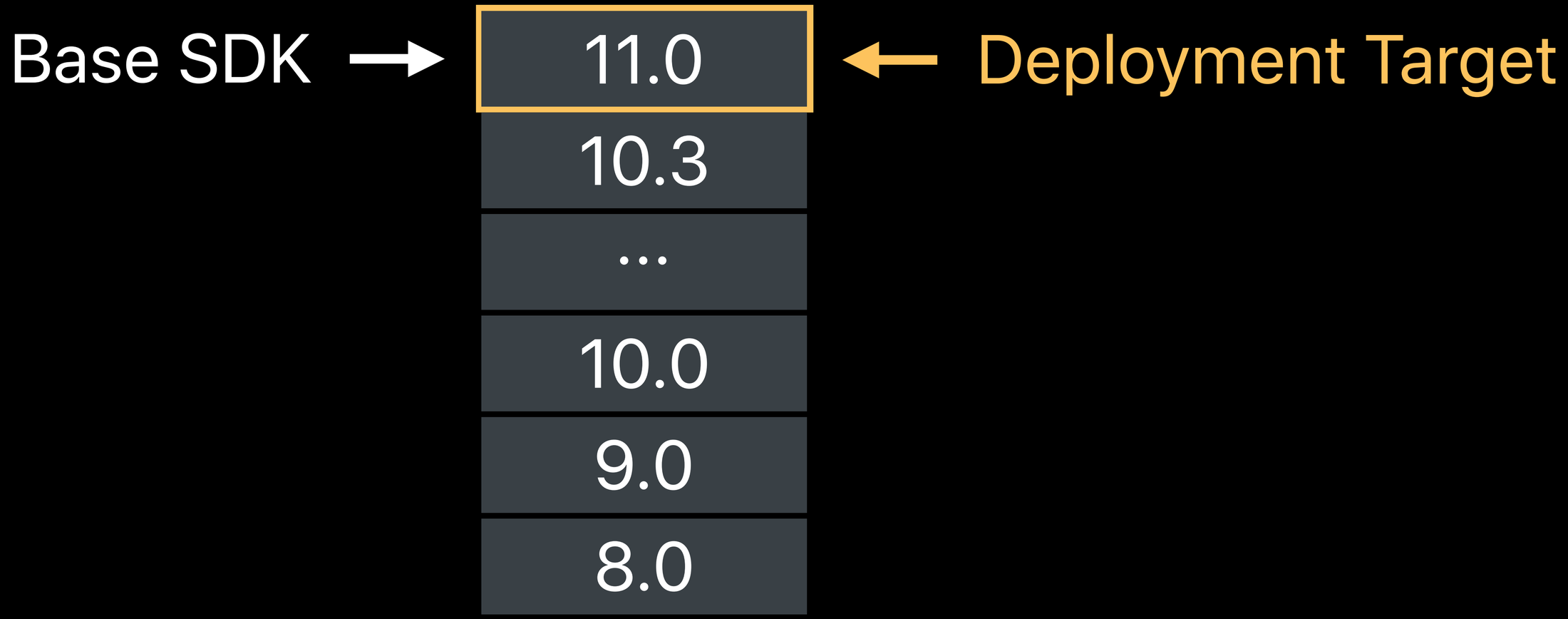
Base SDK and Deployment Target

Example: Supporting Multiple iOS Releases



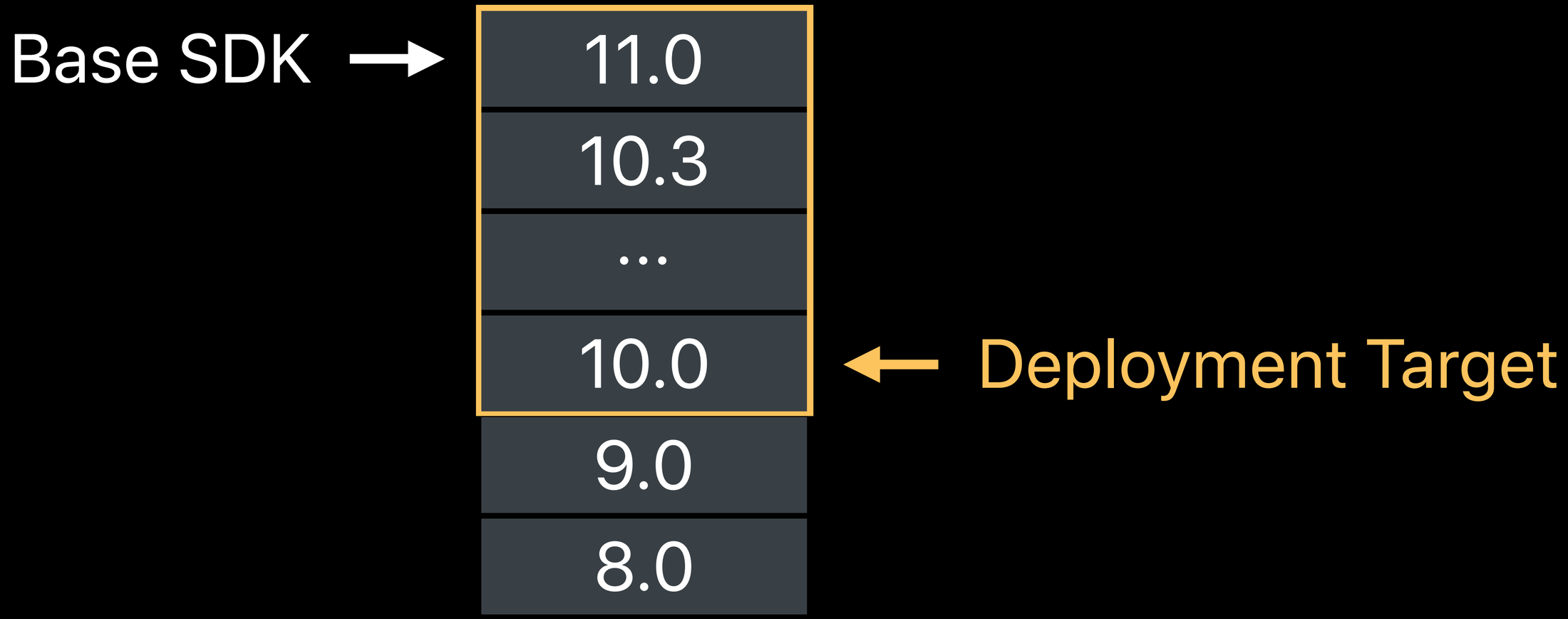
Base SDK and Deployment Target

Example: Supporting Multiple iOS Releases



Base SDK and Deployment Target

Example: Supporting Multiple iOS Releases



Only Call APIs When Available at Run Time

Crash when new API called on older OS

Query runtime for availability?

- Easy to get wrong
- Hard to test
- Different syntax for different kinds of APIs

```
&ARErrorDomain != NULL
```

```
&ARErrorDomain != NULL
```

```
futimens != NULL
```

```
[UIDragInteraction class]
```

```
[UIView instancesRespondToSelector:@selector(addInteraction:)]
```

```
[NSOrthography respondsToSelector:@selector(defaultOrthographyForLanguage:)]
```

Availability Checking in Swift

Unified query syntax `#available`

Availability Checking in Swift

Unified query syntax `#available`

Compiler catches missing availability checks

Availability Checking in Swift

Unified query syntax `#available`

Compiler catches missing availability checks

Availability Checking for Objective-C!

API Availability Checking in Objective-C

```
r = [VNDetectFaceRectanglesRequest new];
if ([handler performRequests:@[r] error:&error]) {
  // Draw rectangles
}
```

API Availability Checking in Objective-C

Compiler warns about unguarded uses of new API

```
r = [VNDetectFaceRectanglesRequest new];
if ([handler performRequests:@[r] error:&error]) {
  // Draw rectangles
}
```

⚠️ ‘VNDetectFaceRectangleRequest is only available on iOS 11.0 or newer

API Availability Checking in Objective-C

Compiler warns about unguarded uses of new API

Use `@available` to query API availability at run time

```
r = [VNDetectFaceRectanglesRequest new];  ⚠ 'VNDetectFaceRectangleRequest is only available on iOS 11.0 or newer
if ([handler performRequests:@[r] error:&error]) {
  // Draw rectangles
}
```

API Availability Checking in Objective-C

Compiler warns about unguarded uses of new API

Use `@available` to query API availability at run time

```
r = [VNDetectFaceRectanglesRequest new];
if ([handler performRequests:@[r] error:&error]) {
  // Draw rectangles
}
```

API Availability Checking in Objective-C

Compiler warns about unguarded uses of new API

Use `@available` to query API availability at run time

```
if (@available(iOS 11, *)) {
  r = [VNDetectFaceRectanglesRequest new];
  if ([handler performRequests:@[r] error:&error]) {
    // Draw rectangles
  }
} else {
  // Fall back when API not available
}
```

API Availability Checking in Objective-C

Compiler warns about unguarded uses of new API

Use `@available` to query API availability at run time

```
if (@available(iOS 11, *)) {
  r = [VNDetectFaceRectanglesRequest new];
  if ([handler performRequests:@[r] error:&error]) {
    // Draw rectangles
  }
} else {
  // Fall back when API not available
}
```

API Availability Checking in Objective-C

```
if (@available(iOS 11, *))
```

API Availability Checking in Objective-C

```
if (@available(iOS 11, *))
```

On iOS returns true when iOS 11 APIs are available

API Availability Checking in Objective-C

```
if (@available(iOS 11, *))
```

On iOS returns true when iOS 11 APIs are available

On all other platforms always returns true

Factor out Code with API_AVAILABLE()

Convenient to write entire methods with limited availability

```
@interface MyAlbumController : UIViewController

- (void)showFaces API_AVAILABLE(ios(11.0));

@end
```

Factor out Code with API_AVAILABLE()

Convenient to write entire methods with limited availability

Can apply to entire classes

```
API_AVAILABLE(ios(11.0))
@interface MyAlbumController : UIViewController


- (void)showFaces;

@end
```

API Availability Checking in C/C++

Use `__builtin_available` to check availability at runtime

```
if (__builtin_available(iOS 11, macOS 10.13, *)) {
    CFNewAPIOniOS11();
}
```

API Availability Checking in C/C++

Use `__builtin_available` to check availability at runtime

Include `<os/availability.h>` for the `API_AVAILABLE` macro

```
#include <os/availability.h>

void myFunctionForiOS11OrNewer(int i) API_AVAILABLE(ios(11.0), macos(10.13));
```

API Availability Checking in C/C++

Use `__builtin_available` to check availability at runtime

Include `<os/availability.h>` for the `API_AVAILABLE` macro

```
#include <os/availability.h>

class API_AVAILABLE(ios(11.0), macos(10.13)) MyClassForiOS11OrNewer;
```

Existing Projects: Warn for New API Only

Warn starting with iOS 11, tvOS 11, macOS 10.13, and watchOS 4

APIs introduced in older SDKs not checked at compile time

Existing code does not need to be rewritten

Use `@available` and `API_AVAILABLE` when adopting new APIs

New Projects: Warn for All Deployment Targets

All APIs are checked at compile time

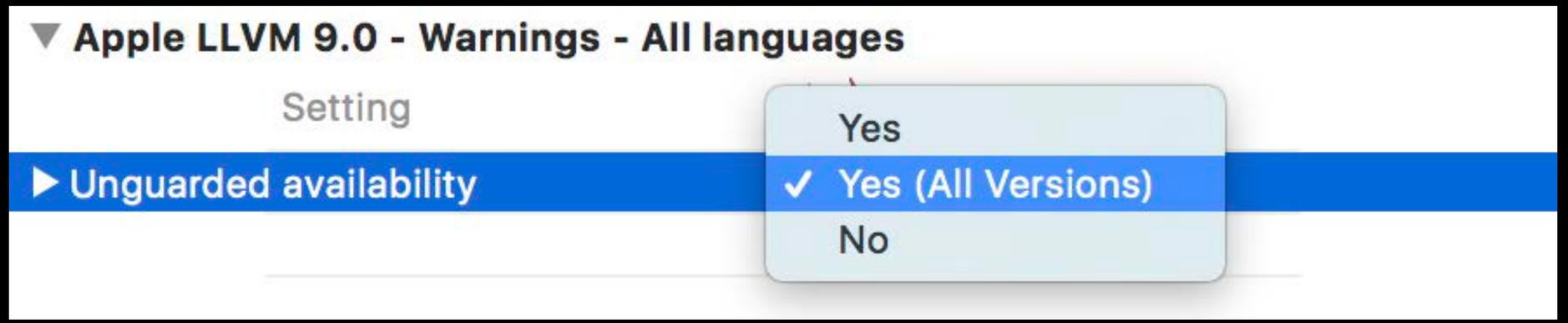
Use `@available` and `API_AVAILABLE` for APIs introduced after deployment target

New Projects: Warn for All Deployment Targets

All APIs are checked at compile time

Use `@available` and `API_AVAILABLE` for APIs introduced after deployment target

Existing projects can opt-in

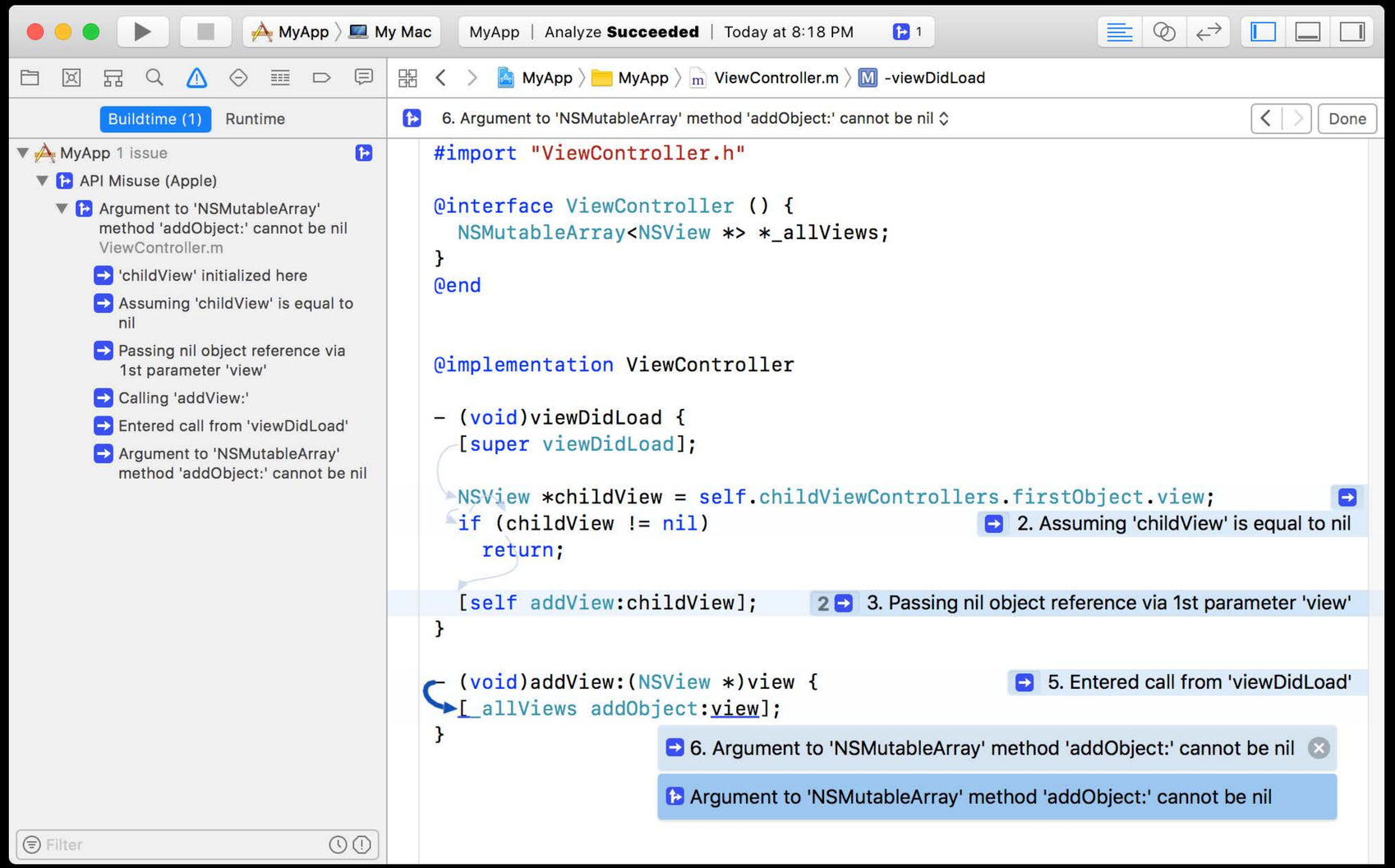



AVAILABLE NOW

Static Analyzer Checks

Finds Deep Bugs

Great at catching hard-to-reproduce, edge-case bugs



Three New Checks

NEW

Suspicious comparisons of `NSNumber` and `CFNumberRef`

Use of `dispatch_once()` on instance variables

Auto-synthesized `copy` properties of `NSMutable` types

Do Not Compare Number Objects to Scalars

Comparing `NSNumber` pointer value to `0` checks for `nil` – not zero number

```
@property NSNumber *photoCount;

- (BOOL)hasPhotos {
   return self.photoCount > 0;
}
```

Do Not Compare Number Objects to Scalars

Comparing `NSNumber` pointer value to `0` checks for `nil` – not zero number

```
@property NSNumber *photoCount;

- (BOOL)hasPhotos {
    return self.photoCount > 0;
}
```

Comparing pointer value to a scalar integer value

Do Not Compare Number Objects to Scalars

Comparing `NSNumber` pointer value to `0` checks for `nil` – not zero number

```
@property NSNumber *photoCount;

- (BOOL)hasPhotos {
    return self.photoCount.integerValue > 0;
}
```

Instead, compare integer value to integer value

Do Not Implicitly Convert Number Objects to Booleans

Implicit conversion to Boolean value checks for `nil` – not zero number

```
@property NSNumber *faceCount;

- (void)identifyFaces {
  if (self.faceCount)
    return;
  // Expensive Processing
}
```

Do Not Implicitly Convert Number Objects to Booleans

Implicit conversion to Boolean value checks for `nil` – not zero number

```
@property NSNumber *faceCount;

- (void)identifyFaces {
  if (self.faceCount)                    Converting pointer value to a primitive boolean value
    return;
  // Expensive Processing
}
```

Do Not Implicitly Convert Number Objects to Booleans

Implicit conversion to Boolean value checks for `nil` – not zero number

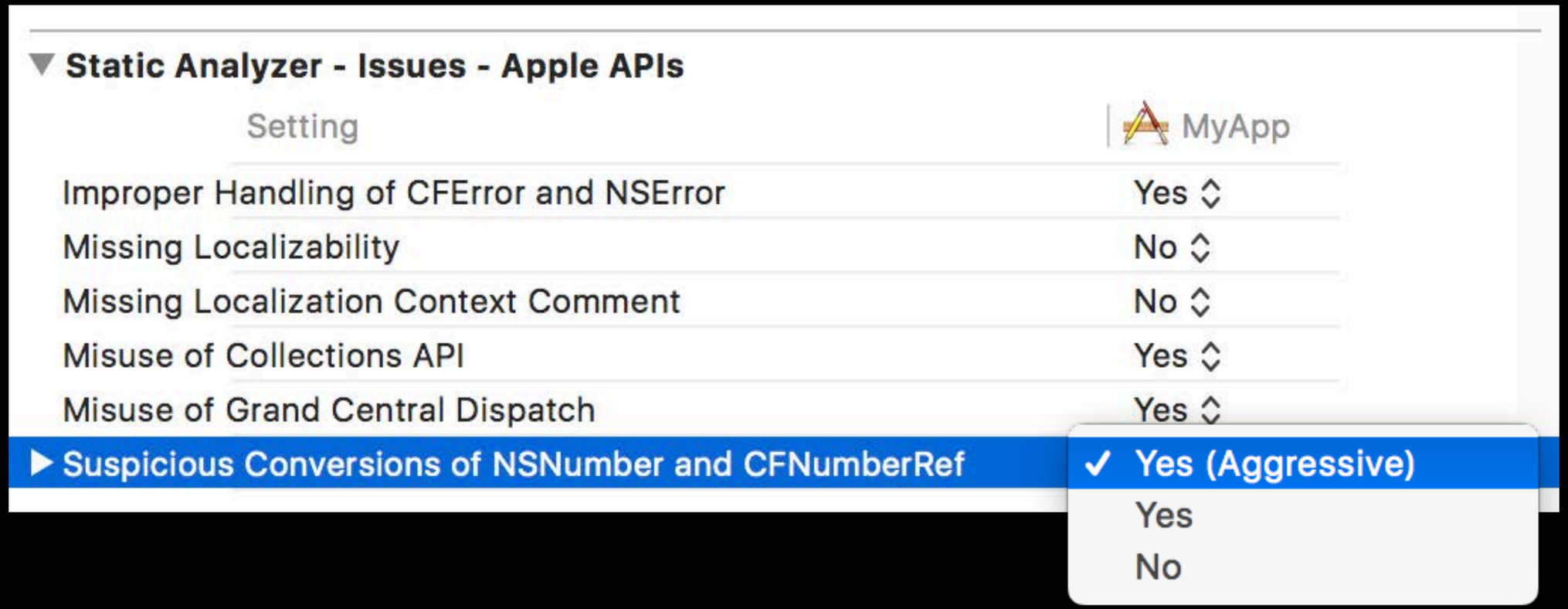
```
@property NSNumber *faceCount;

- (void)identifyFaces {
  if (self.faceCount != nil)
    return;
  // Expensive Processing
}
```

Instead, compare to nil explicitly

Control Check in Build Settings

Check for ambiguity by selecting ‘Yes (Aggressive)’:



Do Use `dispatch_once()` to Initialize Global State

```
+ (NSArray<NSImage *>)sharedPhotos {
  static NSArray<NSImage *> *sharedPhotos;
  static dispatch_once_t oncePredicate;
  dispatch_once(&oncePredicate, ^{
    sharedPhotos = [self loadPhotos];
  });
  return sharedPhotos;
}
```

Do Use `dispatch_once()` to Initialize Global State

```
+ (NSArray<NSImage *>)sharedPhotos {
  static NSArray<NSImage *> *sharedPhotos;
  static dispatch_once_t oncePredicate;
  dispatch_once(&oncePredicate, ^{
    sharedPhotos = [self loadPhotos];
  });
  return sharedPhotos;
}
```

Guarantees block is called exactly once

Do Use `dispatch_once()` to Initialize Global State

```
+ (NSArray<NSImage *>)sharedPhotos {
  static NSArray<NSImage *> *sharedPhotos;
  static dispatch_once_t oncePredicate;
  dispatch_once(&oncePredicate, ^{
    sharedPhotos = [self loadPhotos];
  });
  return sharedPhotos;
}
```

Guarantees block is called exactly once

Predicate must be global or static variable

Do Not Store `dispatch_once_t` in Instance Variables

```
@implementation Album {
  dispatch_once_t oncePredicate;
}


dispatch_once(&oncePredicate, ^{
  self.photos = [self loadPhotos];
});
```

Do Not Store `dispatch_once_t` in Instance Variables

```
@implementation Album {
  dispatch_once_t oncePredicate;
}


dispatch_once(&oncePredicate, ^{
  self.photos = [self loadPhotos];
});
```

Call to ‘dispatch_once’ uses instance variable for predicate

Do Not Store `dispatch_once_t` in Instance Variables

```
@implementation Album {
  NSLock *photosLock;
}



  self.photos = [self loadPhotos];
```

Instead, use a lock to guarantee initialization performed once

Do Not Store `dispatch_once_t` in Instance Variables

```
@implementation Album {
  NSLock *photosLock;
}

[photosLock lock];


  self.photos = [self loadPhotos];
```

Instead, use a lock to guarantee initialization performed once

Do Not Store `dispatch_once_t` in Instance Variables

```
@implementation Album {
  NSLock *photosLock;
}

[photosLock lock];
if (self.photos == nil) {
  self.photos = [self loadPhotos];
}
```

Instead, use a lock to guarantee initialization performed once

Do Not Store `dispatch_once_t` in Instance Variables

```
@implementation Album {
  NSLock *photosLock;
}

[photosLock lock];
if (self.photos == nil) {
  self.photos = [self loadPhotos];
}
[photosLock unlock];
```

Instead, use a lock to guarantee initialization performed once

Do Not Auto-Synthesize NSMutable copy Properties

Setter calls `-copy`, which yields an immutable copy

```
@property (copy) NSMutableArray<NSImage *> *photos;

- (void)replaceWithStockPhoto:(NSImage *)stockPhoto {
  self.photos = [NSMutableArray<NSImage *> new];
  [self.photos addObject:stockPhoto];
}
```

Do Not Auto-Synthesize NSMutable copy Properties

Setter calls `-copy`, which yields an immutable copy

```
@property (copy) NSMutableArray<NSImage *> *photos;

- (void)replaceWithStockPhoto:(NSImage *)stockPhoto {
  self.photos = [NSMutableArray<NSImage *> new];
  [self.photos addObject:stockPhoto];
}
```

```
-[__NSArray0 addObject:]: unrecognized selector sent to instance
```

Do Not Auto-Synthesize NSMutable copy Properties

Setter calls `-copy`, which yields an immutable copy

```
@property (copy) NSMutableArray<NSImage *> *photos;
```

Property of mutable type has ‘copy’ attribute

```
- (void)replaceWithStockPhoto:(NSImage *)stockPhoto {
  self.photos = [NSMutableArray<NSImage *> new];
  [self.photos addObject:stockPhoto];
}
```

Write NSMutable copy Accessors Explicitly

Instead, write explicit setter that calls `-mutableCopy`

```
@property (copy) NSMutableArray<NSImage *> *photos;
```

Property of mutable type has ‘copy’ attribute

Write NSMutable copy Accessors Explicitly

Instead, write explicit setter that calls `-mutableCopy`

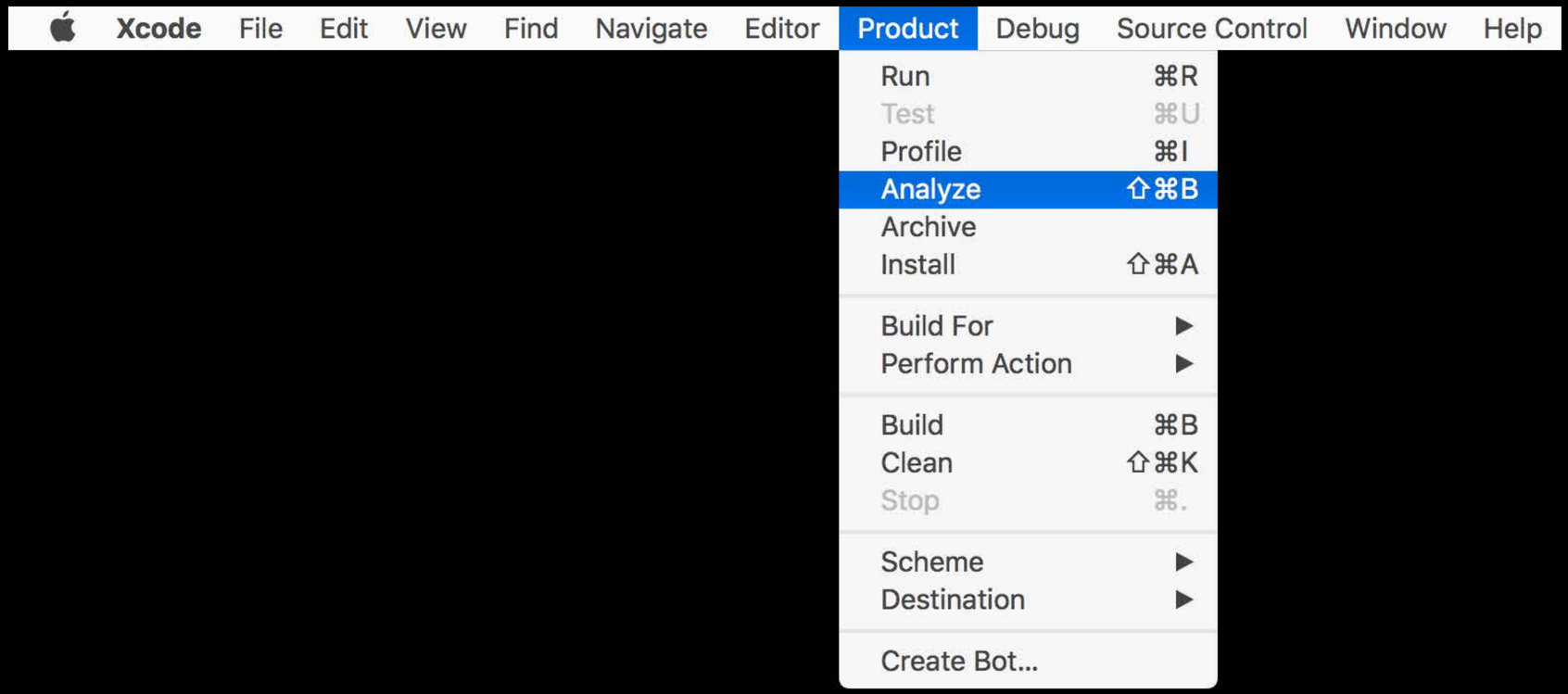
```
@property (copy) NSMutableArray<NSImage *> *photos;

@synthesize photos = _photos;

- (void)setPhotos:(NSMutableArray<NSImage *> *)photos {
  _photos = [photos mutableCopy];
}
```

Run Analyzer on Your Code!

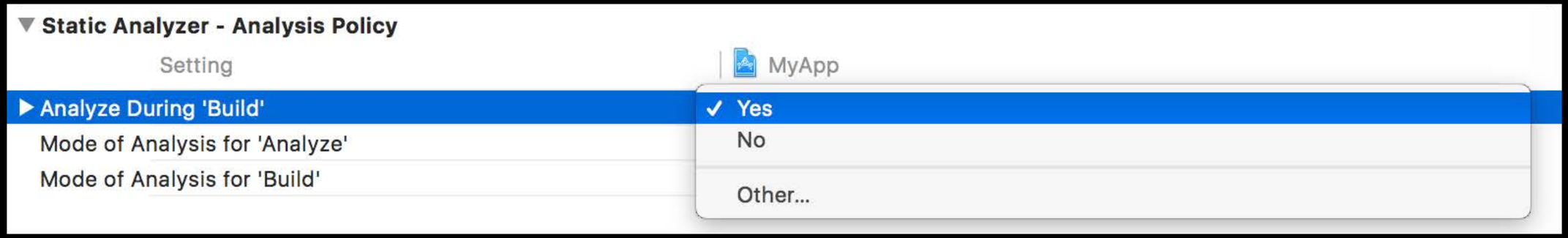
Supports Objective-C, C, C++



Run Analyzer on Your Code!

Supports Objective-C, C, C++

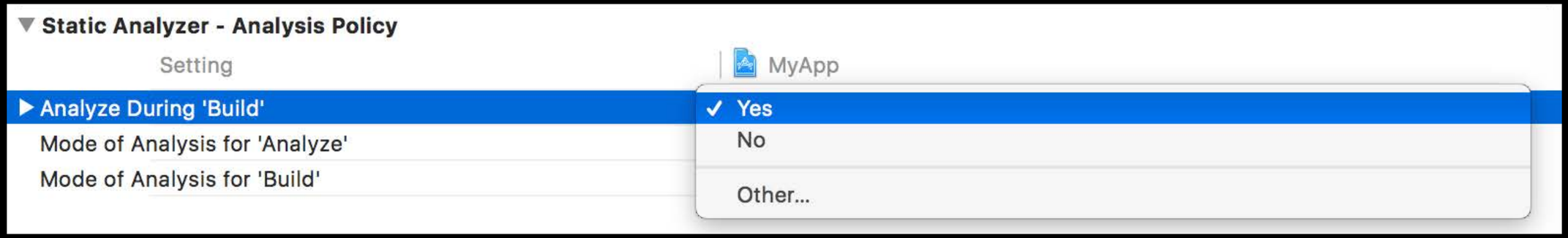
Analyze during build



Run Analyzer on Your Code!

Supports Objective-C, C, C++

Analyze during build



New Warnings

Duncan Exon Smith, Clang Frontend Manager

Capturing Parameters in Blocks with ARC

Most parameters are safe to capture in blocks

Capturing Parameters in Blocks with ARC

Most parameters are safe to capture in blocks

```
- (void)validateDictionary:(NSDictionary *)dict usingChecker:(Checker *)checker {
  [dict enumerateKeysAndObjectsUsingBlock:^(id key, id obj, BOOL *stop) {
    if ([checker checkObject:obj forKey:key]) return;
    *stop = YES;
  }];
}
```

Capturing Parameters in Blocks with ARC

Most parameters are safe to capture in blocks

```
- (void)validateDictionary:(NSDictionary *)dict usingChecker:(Checker *)checker {
  [dict enumerateKeysAndObjectsUsingBlock:^(id key, id obj, BOOL *stop) {
    if ([checker checkObject:obj forKey:key]) return;
    *stop = YES;
  }];
}
```

Capturing Parameters in Blocks with ARC

Most parameters are safe to capture in blocks

```
- (void)validateDictionary:(NSDictionary *)dict usingChecker:(Checker *)checker {
  [dict enumerateKeysAndObjectsUsingBlock:^(id key, id obj, BOOL *stop) {
    if ([checker checkObject:obj forKey:key]) return;
    *stop = YES;
  }];
}
```

Capturing Parameters in Blocks with ARC

Most parameters are safe to capture in blocks

```
- (    )validateDictionary:(NSDictionary *)dict usingChecker:(Checker *)checker
                                                    error:(NSError **)error {
  __block BOOL isValid = YES;
  [dict enumerateKeysAndObjectsUsingBlock:^(id key, id obj, BOOL *stop) {
    if ([checker checkObject:obj forKey:key]) return;
    *stop = YES; isValid = NO;
    if (error) *error = [NSError errorWithDomain:...];
  }];
  return isValid;
}
```

Capturing Parameters in Blocks with ARC

Most parameters are safe to capture in blocks

```
- (BOOL)validateDictionary:(NSDictionary *)dict usingChecker:(Checker *)checker
                                                      error:(NSError **)error {
  __block BOOL isValid = YES;
  [dict enumerateKeysAndObjectsUsingBlock:^(id key, id obj, BOOL *stop) {
    if ([checker checkObject:obj forKey:key]) return;
    *stop = YES; isValid = NO;
    if (error) *error = [NSError errorWithDomain:...];
  }];
  return isValid;
}
```

Capturing Parameters in Blocks with ARC

Most parameters are safe to capture in blocks

```
- (BOOL)validateDictionary:(NSDictionary *)dict usingChecker:(Checker *)checker
                                                      error:(NSError **)error {
  __block BOOL isValid = YES;
  [dict enumerateKeysAndObjectsUsingBlock:^(id key, id obj, BOOL *stop) {
    if ([checker checkObject:obj forKey:key]) return;
    *stop = YES; isValid = NO;
    if (error) *error = [NSError errorWithDomain:...];
  }];
  return isValid;
}
```

Capturing Parameters in Blocks with ARC

Most parameters are safe to capture in blocks

```
- (BOOL)validateDictionary:(NSDictionary *)dict usingChecker:(Checker *)checker
                                                      error:(NSError **)error {
  __block BOOL isValid = YES;
  [dict enumerateKeysAndObjectsUsingBlock:^(id key, id obj, BOOL *stop) {
    if ([checker checkObject:obj forKey:key]) return;
    *stop = YES; isValid = NO;
    if (error) *error = [NSError errorWithDomain:...];
  }];
  return isValid;
}
```

Capturing Parameters in Blocks with ARC

Most parameters are safe to capture in blocks

```
- (BOOL)validateDictionary:(NSDictionary *)dict usingChecker:(Checker *)checker
                                                      error:(NSError **)error {
  __block BOOL isValid = YES;
  [dict enumerateKeysAndObjectsUsingBlock:^(id key, id obj, BOOL *stop) {
    if ([checker checkObject:obj forKey:key]) return;
    *stop = YES; isValid = NO;
    if (error) *error = [NSError errorWithDomain:...];
  }];
  return isValid;
}
```

Capturing Parameters in Blocks with ARC

Most parameters are safe to capture in blocks

```
- (BOOL)validateDictionary:(NSDictionary *)dict usingChecker:(Checker *)checker
                                                      error:(NSError **)error {
  __block BOOL isValid = YES;
  [dict enumerateKeysAndObjectsUsingBlock:^(id key, id obj, BOOL *stop) {
    if ([checker checkObject:obj forKey:key]) return;
    *stop = YES; isValid = NO;
    if (error) *error = [NSError errorWithDomain:...];
  }];
  return isValid;
}
```

Assigning to Out Parameters in Blocks Is Unsafe

Out parameters are implicitly `__autoreleasing` in ARC

```
- (BOOL)validateDictionary:(NSDictionary *)dict usingChecker:(Checker *)checker
                                                      error:(NSError **)error {
  __block BOOL isValid = YES;
  [dict enumerateKeysAndObjectsUsingBlock:^(id key, id obj, BOOL *stop) {
    if ([checker checkObject:obj forKey:key]) return;
    *stop = YES; isValid = NO;
    if (error) *error = [NSError errorWithDomain:...];
  }];
  return isValid;
}
```

Assigning to Out Parameters in Blocks Is Unsafe

Out parameters are implicitly `__autoreleasing` in ARC

```
- (BOOL)validateDictionary:(NSDictionary *)dict usingChecker:(Checker *)checker
                                                      error:(NSError **)error {
  __block BOOL isValid = YES;
  [dict enumerateKeysAndObjectsUsingBlock:^(id key, id obj, BOOL *stop) {
    if ([checker checkObject:obj forKey:key]) return;
    *stop = YES; isValid = NO;
    if (error) *error = [[[NSError errorWithDomain:...] retain] autorelease];
  }];
  return isValid;
}
```

```
//Assigning to Out Parameters in Blocks Is Unsafe
//Out parameters are implicitly __autoreleasing in ARC

- (BOOL)validateDictionary:(NSDictionary *)dict usingChecker:(Checker *)checker
                                                    error:(NSError **)error {
  __block BOOL isValid = YES;
  [dict enumerateKeysAndObjectsUsingBlock:^(id key, id obj, BOOL *stop) {
    if ([checker checkObject:obj forKey:key]) return;
    *stop = YES; isValid = NO;
    if (error) *error = [[[NSError errorWithDomain:...] retain] autorelease];
  }];
  return isValid;
}

- (void)enumerateKeysAndObjectsUsingBlock:(void (^)(KeyT key, ObjectT obj, BOOL *stop)block {
  @autoreleasepool {
    ...
  }
}
```

```
//Assigning to Out Parameters in Blocks Is Unsafe
//Out parameters are implicitly __autoreleasing in ARC

- (BOOL)validateDictionary:(NSDictionary *)dict usingChecker:(Checker *)checker
                                                    error:(NSError **)error {
  __block BOOL isValid = YES;
  [dict enumerateKeysAndObjectsUsingBlock:^(id key, id obj, BOOL *stop) {
    if ([checker checkObject:obj forKey:key]) return;
    *stop = YES; isValid = NO;
    if (error) *error = [[[NSError errorWithDomain:...] retain] autorelease];
  }];
  return isValid;
}

- (void)enumerateKeysAndObjectsUsingBlock:(void (^)(KeyT key, ObjectT obj, BOOL *stop)block {
  @autoreleasepool {
    ...
  }
}
```

```
//Assigning to Out Parameters in Blocks Is Unsafe
//Out parameters are implicitly __autoreleasing in ARC

- (BOOL)validateDictionary:(NSDictionary *)dict usingChecker:(Checker *)checker
                                                    error:(NSError **)error {
  __block BOOL isValid = YES;
  [dict enumerateKeysAndObjectsUsingBlock:^(id key, id obj, BOOL *stop) {
    if ([checker checkObject:obj forKey:key]) return;
    *stop = YES; isValid = NO;
    if (error) *error = [[[NSError errorWithDomain:...] retain] autorelease];
  }];
  return isValid;
}

- (void)enumerateKeysAndObjectsUsingBlock:(void (^)(KeyT key, ObjectT obj, BOOL *stop)block {
  @autoreleasepool {
    ...
  }
}
```

Assigning to Out Parameters in Blocks Is Unsafe

Out parameters are implicitly `__autoreleasing` in ARC

```
- (BOOL)validateDictionary:(NSDictionary *)dict usingChecker:(Checker *)checker
                                                      error:(NSError **)error {
  __block BOOL isValid = YES;
  [dict enumerateKeysAndObjectsUsingBlock:^(id key, id obj, BOOL *stop) {
    if ([checker checkObject:obj forKey:key]) return;
    *stop = YES; isValid = NO;
    if (error) *error = [NSError errorWithDomain:...];
  }];
  return isValid;
}
```

⚠️ Block captures an autoreleasing out-parameter

Assigning to Out Parameters in Blocks Is Unsafe

Out parameters are implicitly `__autoreleasing` in ARC

```
- (BOOL)validateDictionary:(NSDictionary *)dict usingChecker:(Checker *)checker
                                                      error:(NSError *__strong *)error {
  __block BOOL isValid = YES;
  [dict enumerateKeysAndObjectsUsingBlock:^(id key, id obj, BOOL *stop) {
    if ([checker checkObject:obj forKey:key]) return;
    *stop = YES; isValid = NO;
    if (error) *error = [NSError errorWithDomain:...];
  }];
  return isValid;
}
```

⚠️ Block captures an autoreleasing out-parameter

Capture a Strong Out Parameter

If all callers use ARC, mark out parameter as `__strong` to keep value alive

```
- (BOOL)validateDictionary:(NSDictionary *)dict usingChecker:(Checker *)checker
                                                      error:(NSError *__strong *)error {
  __block BOOL isValid = YES;
  [dict enumerateKeysAndObjectsUsingBlock:^(id key, id obj, BOOL *stop) {
    if ([checker checkObject:obj forKey:key]) return;
    *stop = YES; isValid = NO;
    if (error) *error = [NSError errorWithDomain:...];
  }];
  return isValid;
}
```

⚠️ Block captures an autoreleasing out-parameter

Capture a Strong Out Parameter

If all callers use ARC, mark out parameter as `__strong` to keep value alive

```
- (BOOL)validateDictionary:(NSDictionary *)dict usingChecker:(Checker *)checker
                                                      error:(NSError *__strong *)error {
  __block BOOL isValid = YES;
  [dict enumerateKeysAndObjectsUsingBlock:^(id key, id obj, BOOL *stop) {
    if ([checker checkObject:obj forKey:key]) return;
    *stop = YES; isValid = NO;
    if (error) *error = [NSError errorWithDomain:...];
  }];
  return isValid;
}
```

Capture a Strong Out Parameter

If all callers use ARC, mark out parameter as `__strong` to keep value alive

```
- (BOOL)validateDictionary:(NSDictionary *)dict usingChecker:(Checker *)checker
                                                      error:(NSError *         *)error {
  __block BOOL isValid = YES;
  [dict enumerateKeysAndObjectsUsingBlock:^(id key, id obj, BOOL *stop) {
    if ([checker checkObject:obj forKey:key]) return;
    *stop = YES; isValid = NO;
    if (error) *error = [NSError errorWithDomain:...];
  }];
  return isValid;
}
```

Capture a Strong Out Parameter

If all callers use ARC, mark out parameter as `__strong` to keep value alive

```
- (BOOL)validateDictionary:(NSDictionary *)dict usingChecker:(Checker *)checker
                                                      error:(NSError **)error {
  __block BOOL isValid = YES;
  __block NSError *strongError = nil;
  [dict enumerateKeysAndObjectsUsingBlock:^(id key, id obj, BOOL *stop) {
    if ([checker checkObject:obj forKey:key]) return;
    *stop = YES; isValid = NO;
    strongError = [NSError errorWithDomain:...];
  }];
  if (error) *error = strongError;
  return isValid;
}
```

Capture a Strong Local Variable

Without changing the API, use a local `__block` variable to keep value alive

```
- (BOOL)validateDictionary:(NSDictionary *)dict usingChecker:(Checker *)checker
                                                      error:(NSError **)error {
  __block BOOL isValid = YES;
  __block NSError *strongError = nil;
  [dict enumerateKeysAndObjectsUsingBlock:^(id key, id obj, BOOL *stop) {
    if ([checker checkObject:obj forKey:key]) return;
    *stop = YES; isValid = NO;
    strongError = [NSError errorWithDomain:...];
  }];
  if (error) *error = strongError;
  return isValid;
}
```

Capture a Strong Local Variable

Without changing the API, use a local `__block` variable to keep value alive

```
- (BOOL)validateDictionary:(NSDictionary *)dict usingChecker:(Checker *)checker
                                                      error:(NSError **)error {
  __block BOOL isValid = YES;
  __block NSError *strongError = nil;
  [dict enumerateKeysAndObjectsUsingBlock:^(id key, id obj, BOOL *stop) {
    if ([checker checkObject:obj forKey:key]) return;
    *stop = YES; isValid = NO;
    strongError = [NSError errorWithDomain:...];
  }];
  if (error) *error = strongError;
  return isValid;
}
```

Capture a Strong Local Variable

Without changing the API, use a local `__block` variable to keep value alive

```
- (BOOL)validateDictionary:(NSDictionary *)dict usingChecker:(Checker *)checker
                                                      error:(NSError **)error {
  __block BOOL isValid = YES;
  __block NSError *strongError = nil;
  [dict enumerateKeysAndObjectsUsingBlock:^(id key, id obj, BOOL *stop) {
    if ([checker checkObject:obj forKey:key]) return;
    *stop = YES; isValid = NO;
    strongError = [NSError errorWithDomain:...];
  }];
  if (error) *error = strongError;
  return isValid;
}
```

Capture a Strong Local Variable

Without changing the API, use a local `__block` variable to keep value alive

```
- (BOOL)validateDictionary:(NSDictionary *)dict usingChecker:(Checker *)checker
                                                      error:(NSError **)error {
  __block BOOL isValid = YES;
  __block NSError *strongError = nil;
  [dict enumerateKeysAndObjectsUsingBlock:^(id key, id obj, BOOL *stop) {
    if ([checker checkObject:obj forKey:key]) return;
    *stop = YES; isValid = NO;
    strongError = [NSError errorWithDomain:...];
  }];
  if (error) *error = strongError;
  return isValid;
}
```

Declaring Functions without Parameters

Non-prototype declarations in C and Objective-C

Declaring Functions without Parameters

Non-prototype declarations in C and Objective-C

```
int foo();
```

Declaring Functions without Parameters

Non-prototype declarations in C and Objective-C

```
int foo();
...
foo();
foo(1, 2, 3);
foo(&x, y);
```

Declaring Functions without Parameters

Non-prototype declarations in C and Objective-C

```
int foo();
...
foo();
foo(1, 2, 3);
foo(&x, y);
```

An empty parameter list does not declare a prototype

Declaring Functions without Parameters

Non-prototype declarations in C and Objective-C

```
int foo();
...
foo();
foo(1, 2, 3);
foo(&x, y);
```

An empty parameter list does not declare a prototype

C language decision from 1977 that we are stuck with

Declaring Functions without Parameters

Non-prototype declarations in C and Objective-C

```
int foo();
...
foo();
foo(1, 2, 3);
foo(&x, y);
```

An empty parameter list does not declare a prototype

C language decision from 1977 that we are stuck with

Error-prone and ill-advised

Strict Prototypes

Warning for non-prototype declarations in C and Objective-C

```
int foo();                                  ⚠ This function declaration is not a prototype
...
foo();
foo(1, 2, 3);
foo(&x, y);
```

An empty parameter list does not declare a prototype

C language decision from 1977 that we are stuck with

Error-prone and ill-advised

Strict Prototypes

Warning for non-prototype declarations in C and Objective-C

```
int foo();
...
foo();
foo(1, 2, 3);
foo(&x, y);
```

⚠️ This function declaration is not a prototype

Strict Prototypes

Warning for non-prototype declarations in C and Objective-C

```
int foo(void);
...
foo();
foo(1, 2, 3);
foo(&x, y);
```

⚠️ This function declaration is not a prototype

Strict Prototypes

Warning for non-prototype declarations in C and Objective-C

```
int foo(void);
...
foo();
foo(1, 2, 3);
foo(&x, y);
```

Too many arguments to function call; expected 0

Too many arguments to function call; expected 0

Strict Prototypes

Warning for non-prototype declarations in C and Objective-C

```
int takesBlock(int (^block)());

- (void)takesBlock:(int (^)())block;
```

Strict Prototypes

Warning for non-prototype declarations in C and Objective-C

```
int takesBlock(int (^block)());                  ⚠ This block declaration is not a prototype

- (void)takesBlock:(int (^)())block;             ⚠ This block declaration is not a prototype
```

Strict Prototypes

Warning for non-prototype declarations in C and Objective-C

```
int takesBlock(int (^block)(void));
```

⚠️ This block declaration is not a prototype

```
- (void)takesBlock:(int (^)(void))block;
```

⚠️ This block declaration is not a prototype

Strict Prototypes

Warning for non-prototype declarations in C and Objective-C

```
int takesBlock(int (^block)(void));

- (void)takesBlock:(int (^)(void))block;
```

Strict Prototypes

Warning for non-prototype declarations in C and Objective-C

```
int takesBlock(int (^block)(void));

- (void)takesBlock:(int (^)(void))block;

[self takesBlock:^(int a, int b, int c) {
  return a + b + c;
}];
```

Strict Prototypes

Warning for non-prototype declarations in C and Objective-C

```
int takesBlock(int (^block)(void));

- (void)takesBlock:(int (^)(void))block;

[self takesBlock:^(int a, int b, int c) {
  return a + b + c;
}];
```

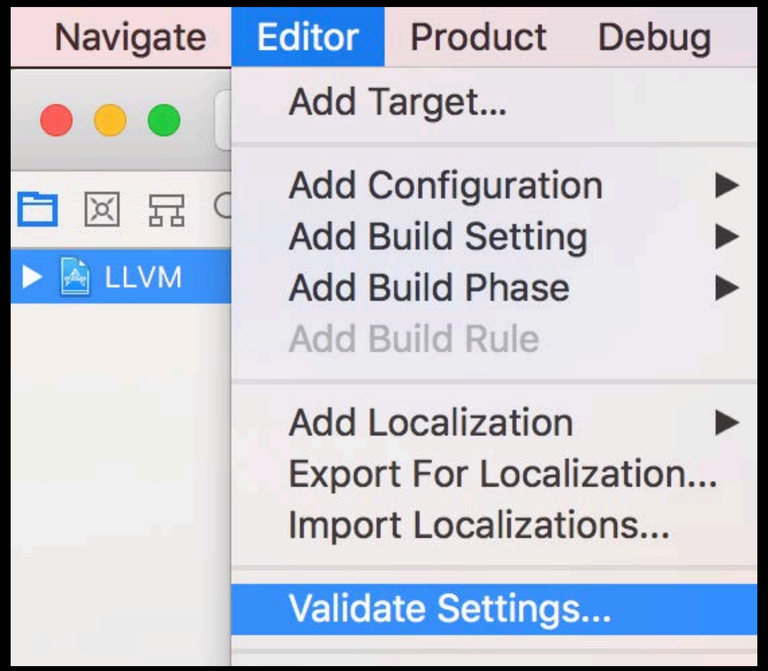
Incompatible block pointer types

Enable Warnings in Build Settings

Validate settings and upgrade warnings to errors

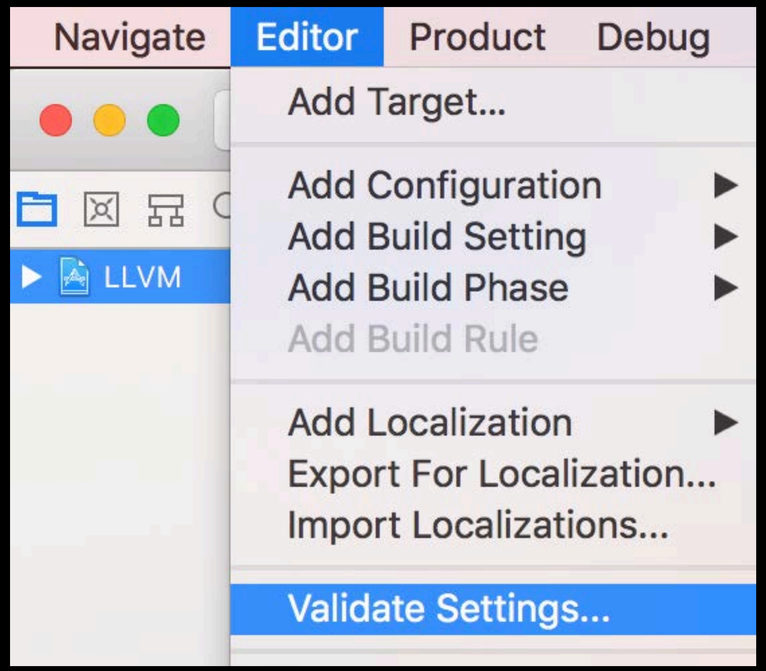
Enable Warnings in Build Settings

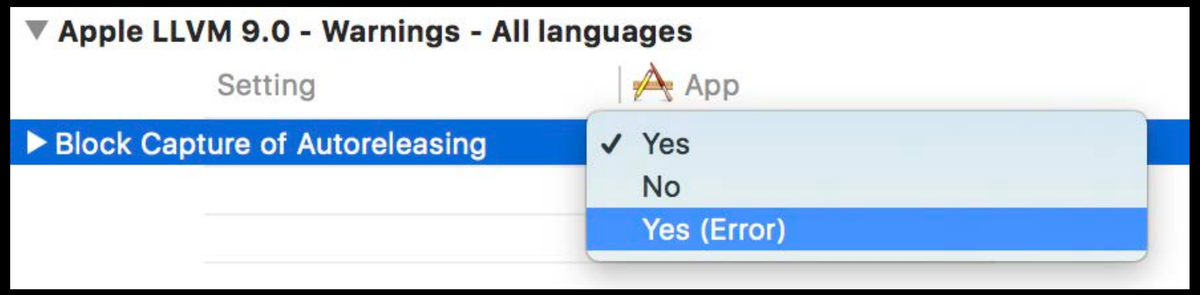
Validate settings and upgrade warnings to errors

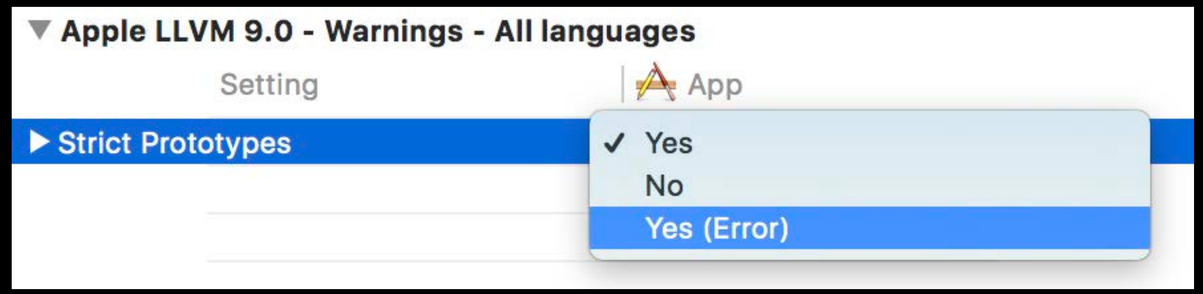


Enable Warnings in Build Settings

Validate settings and upgrade warnings to errors







C++ Refactoring

Extract function

If to switch

Rename global variable

Rename template specializations

Rename field

Missing abstract function override

Rename member function

Missing switch cases

Rename class

Repeated reference expression

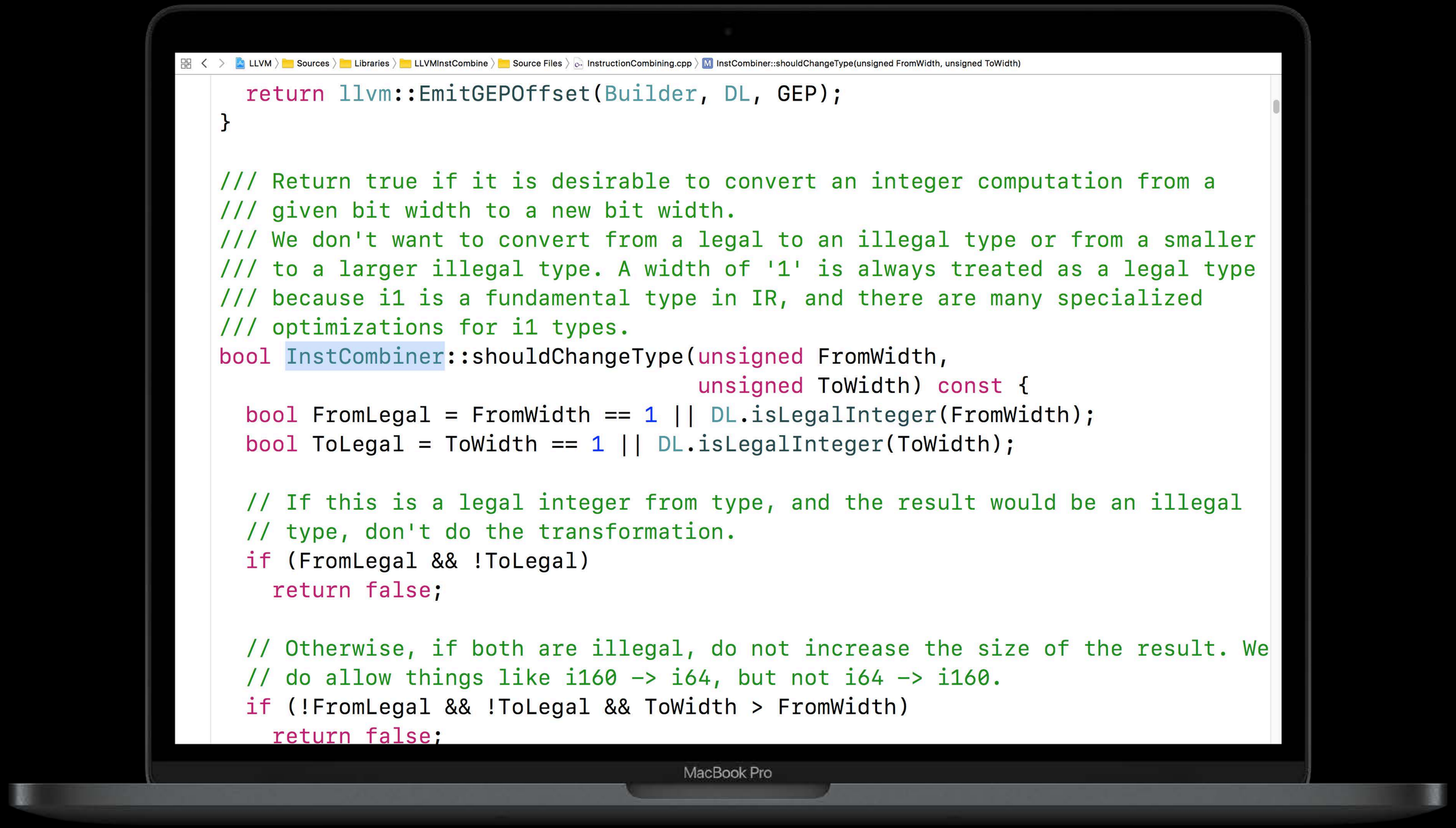
Switch to if

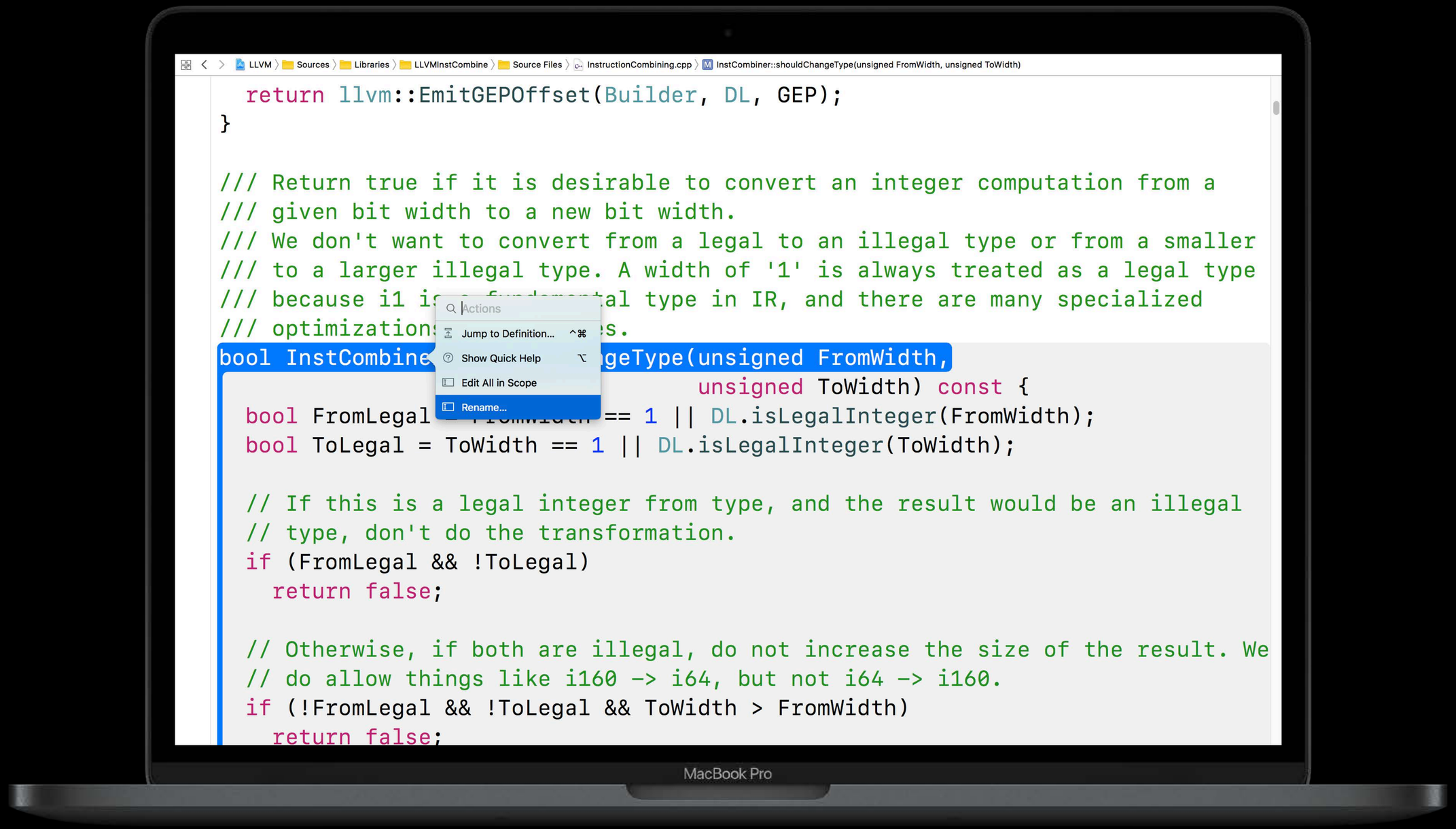
Extract templated function

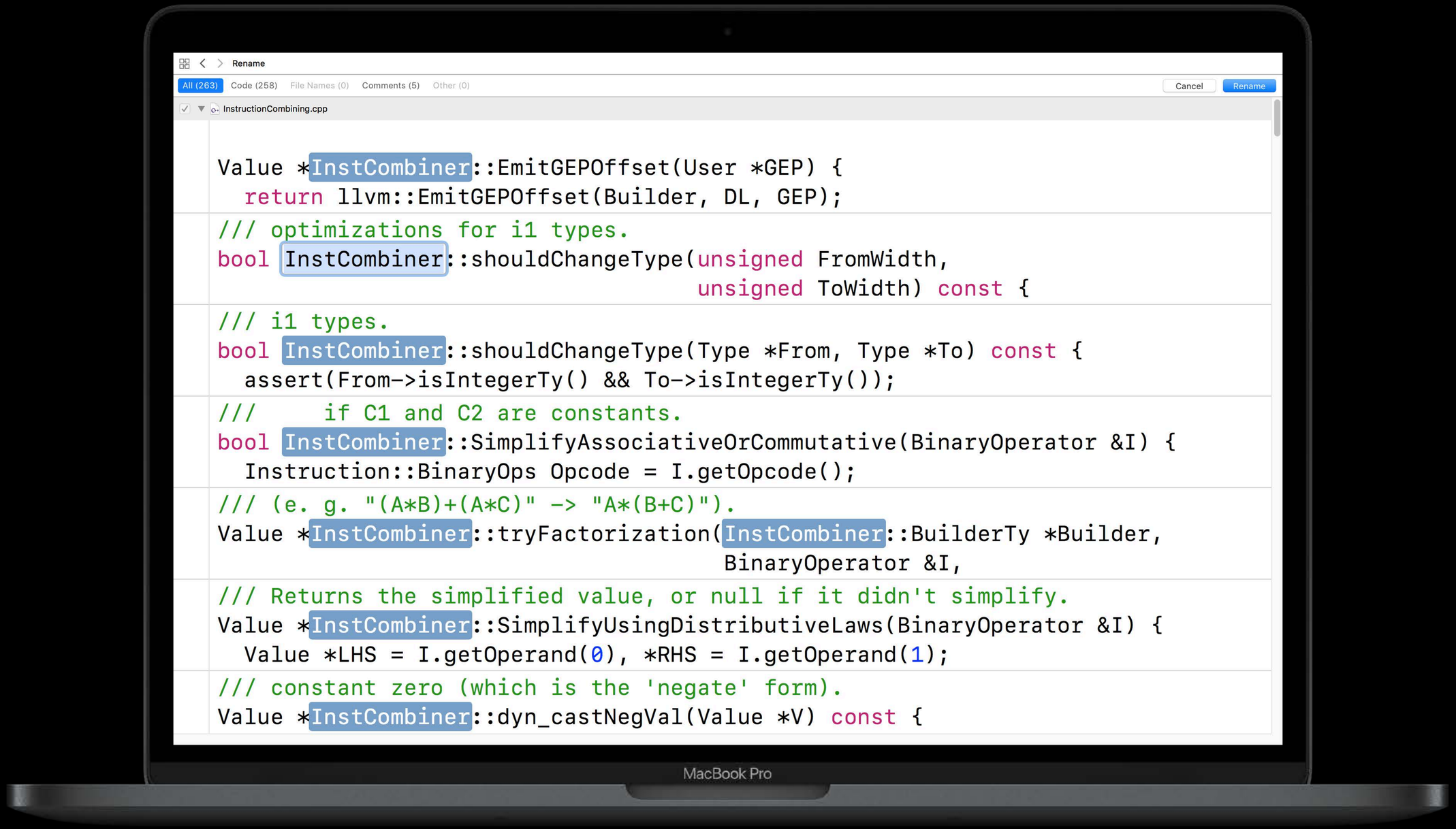
Rename function

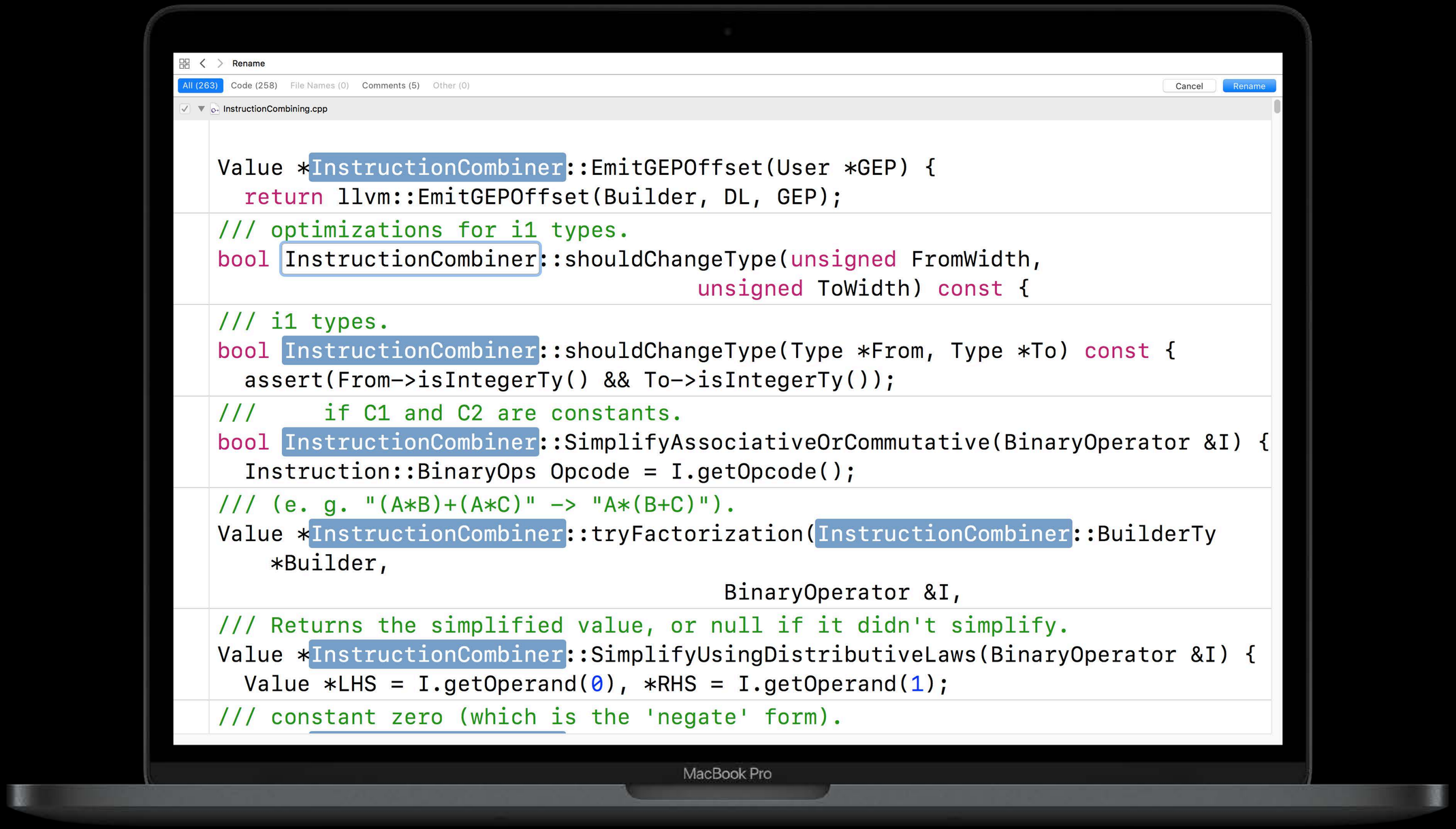
Missing function definition

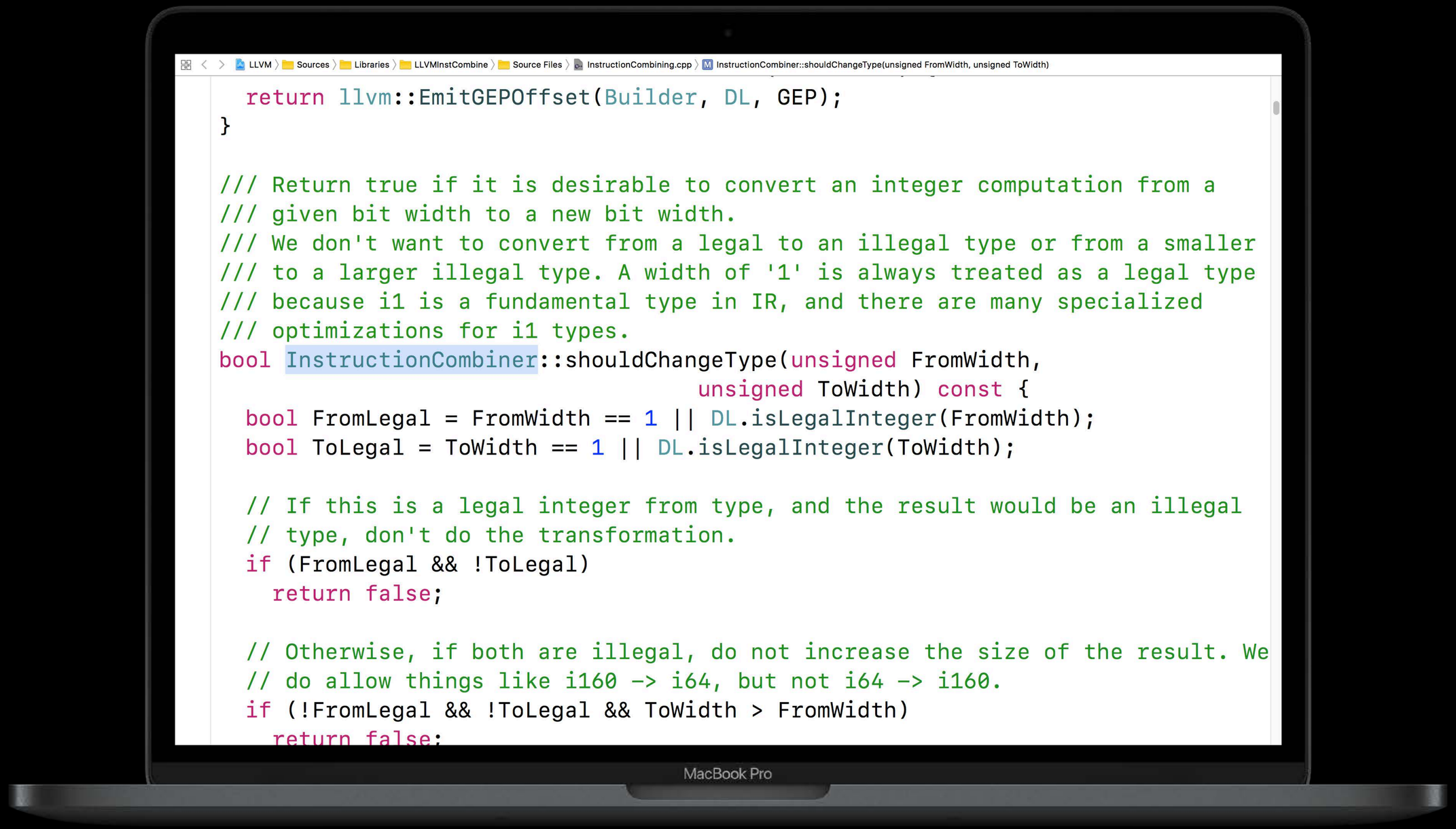
Rename local variable

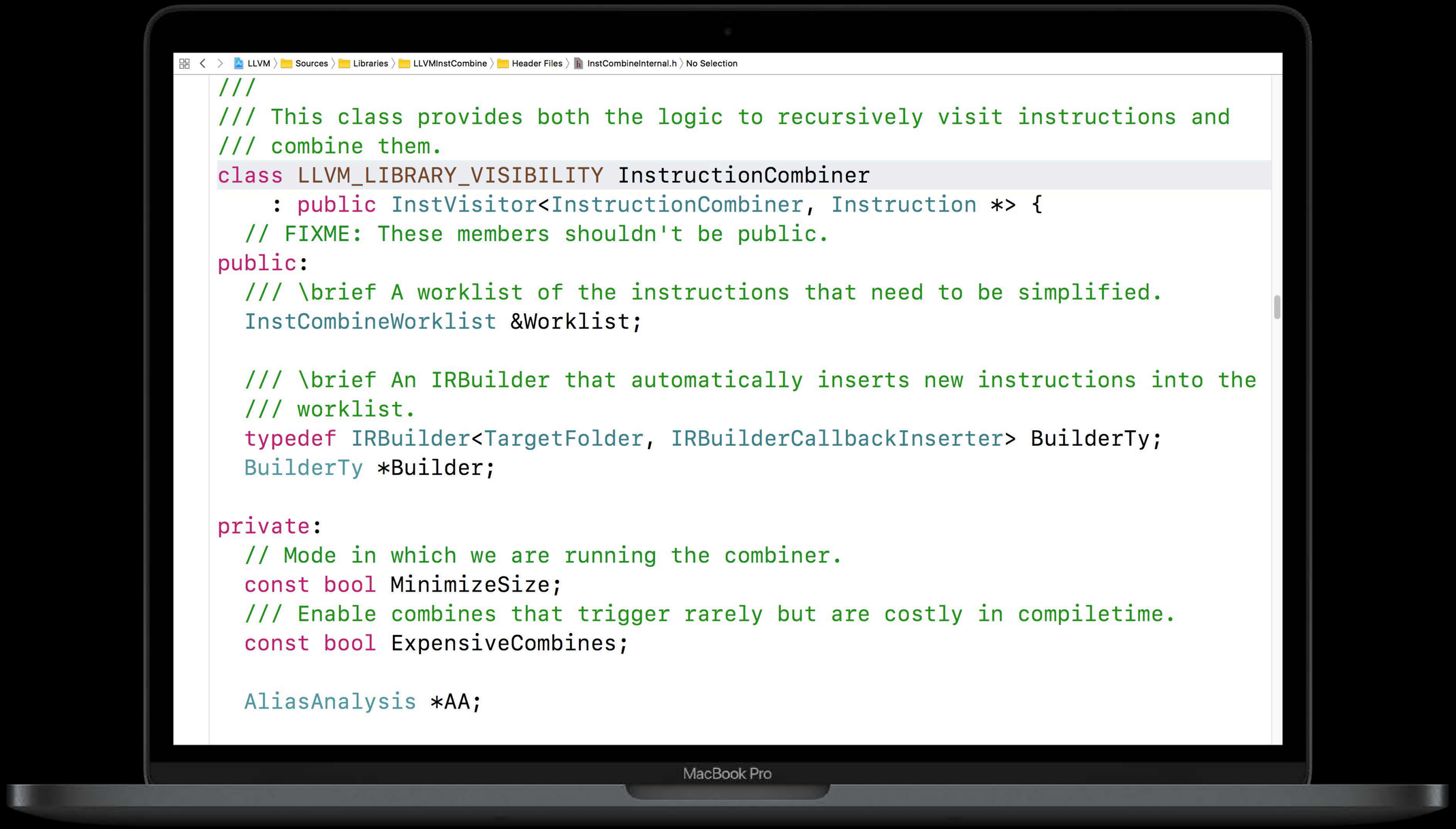

LLVM 〉 Sources 〉 Libraries 〉 LLVMInstCombine 〉 Source Files 〉 InstructionCombining.cpp 〉 InstCombiner::shouldChangeType(unsigned FromWidth, unsigned ToWidth)
return llvm::EmitGEPOffset(Builder, DL, GEP);
}
/// Return true if it is desirable to convert an integer computation from a
/// given bit width to a new bit width.
/// We don't want to convert from a legal to an illegal type or from a smaller
/// to a larger illegal type. A width of '1' is always treated as a legal type
/// because i1 is a fundamental type in IR, and there are many specialized
/// optimizations for i1 types.
bool InstCombiner::shouldChangeType(unsigned FromWidth,
unsigned ToWidth) const {
bool FromLegal = FromWidth == 1 || DL.isLegalInteger(FromWidth);
bool ToLegal = ToWidth == 1 || DL.isLegalInteger(ToWidth);
// If this is a legal integer from type, and the result would be an illegal
// type, don't do the transformation.
if (FromLegal && !ToLegal)
return false;
// Otherwise, if both are illegal, do not increase the size of the result. We
// do allow things like i160 -> i64, but not i64 -> i160.
if (!FromLegal && !ToLegal && ToWidth > FromWidth)
return false;
MacBook Pro

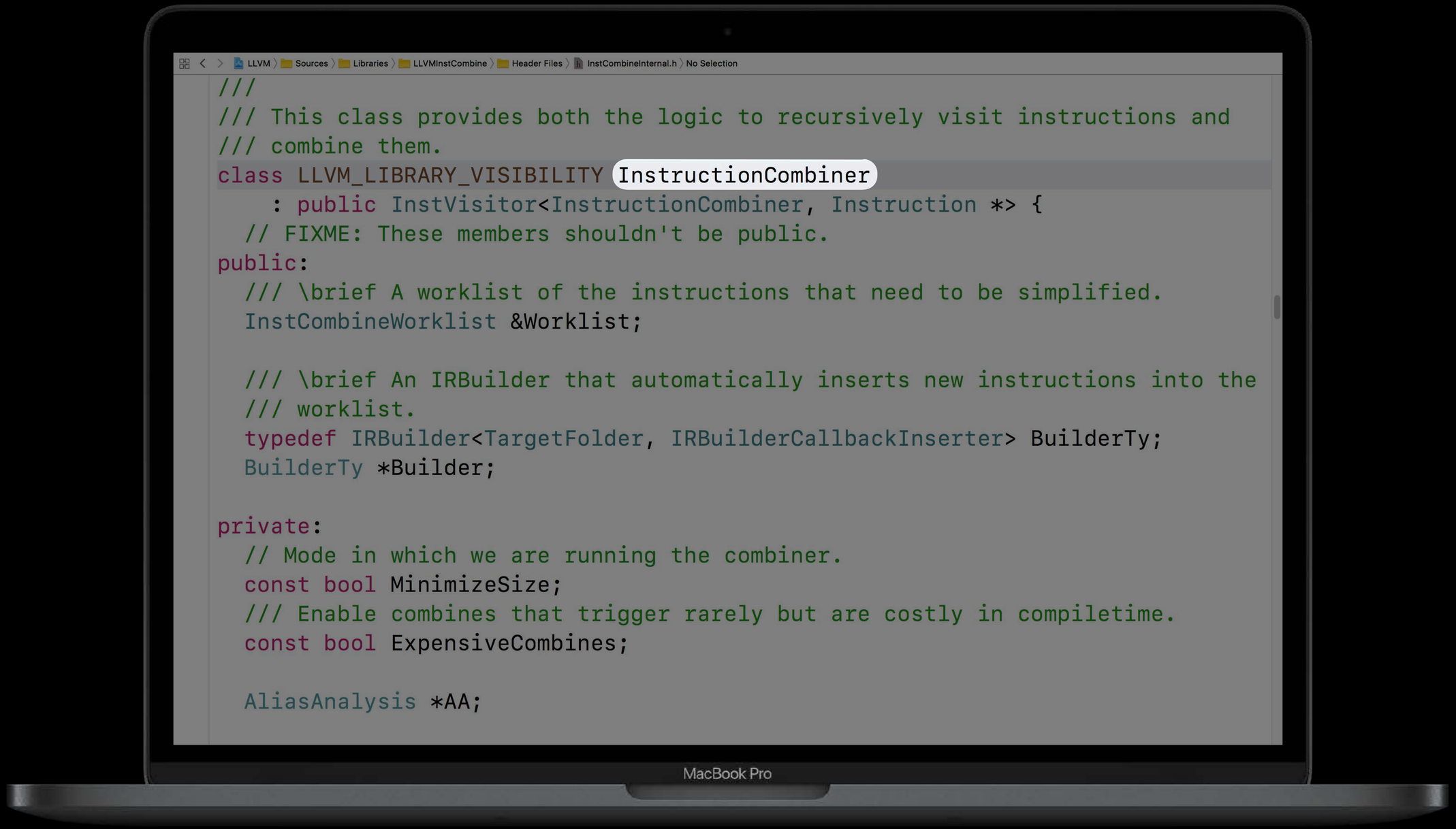

LLVM
Sources
Libraries
LLVMInstCombine
Source Files
InstructionCombining.cpp
InstCombiner::shouldChangeType(unsigned FromWidth, unsigned ToWidth)
return llvm::EmitGEPOffset(Builder, DL, GEP);
}
/// Return true if it is desirable to convert an integer computation from a
/// given bit width to a new bit width.
/// We don't want to convert from a legal to an illegal type or from a smaller
/// to a larger illegal type. A width of '1' is always treated as a legal type
/// because i1 is a fundamental type in IR, and there are many specialized
/// optimizations.
bool InstCombine
geType(unsigned FromWidth,
unsigned ToWidth) const {
bool FromLegal
== 1 || DL.isLegalInteger(FromWidth);
bool ToLegal = ToWidth == 1 || DL.isLegalInteger(ToWidth);
// If this is a legal integer from type, and the result would be an illegal
// type, don't do the transformation.
if (FromLegal && !ToLegal)
return false;
// Otherwise, if both are illegal, do not increase the size of the result. We
// do allow things like i160 -> i64, but not i64 -> i160.
if (!FromLegal && !ToLegal && ToWidth > FromWidth)
return false;
Actions
Jump to Definition... ^⌘
Show Quick Help ⌥
Edit All in Scope
Rename...
MacBook Pro

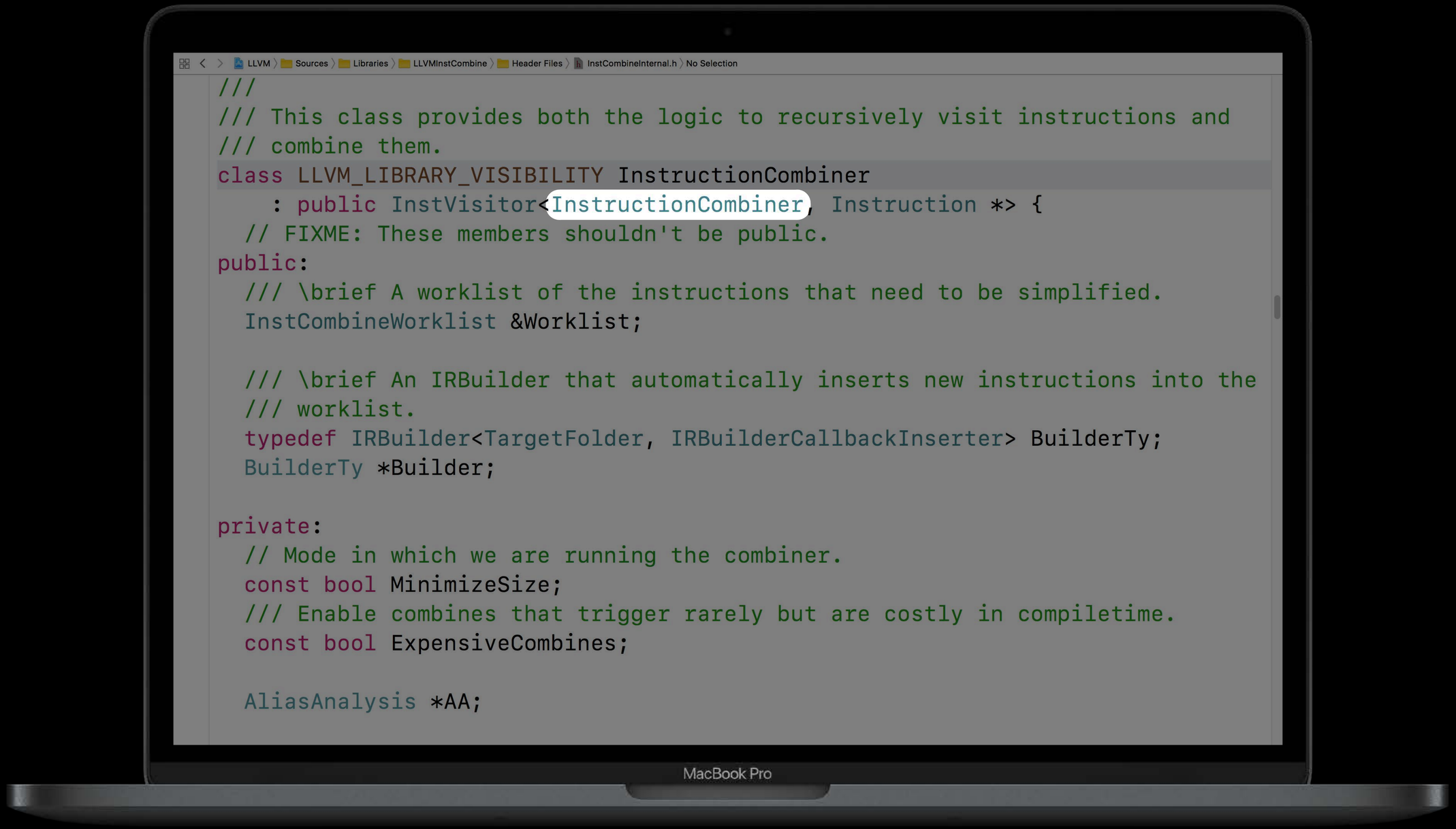

Rename
All (263)
Code (258)
File Names (0)
Comments (5)
Other (0)
Cancel
Rename
InstructionCombining.cpp
Value *InstCombiner::EmitGEPOffset(User *GEP) {
return llvm::EmitGEPOffset(Builder, DL, GEP);
/// optimizations for i1 types.
bool InstCombiner::shouldChangeType(unsigned FromWidth,
unsigned ToWidth) const {
/// i1 types.
bool InstCombiner::shouldChangeType(Type *From, Type *To) const {
assert(From->isIntegerTy() && To->isIntegerTy());
/// if C1 and C2 are constants.
bool InstCombiner::SimplifyAssociativeOrCommutative(BinaryOperator &I) {
Instruction::BinaryOps Opcode = I.getOpcode();
/// (e. g. "(A*B)+(A*C)" -> "A*(B+C)").
Value *InstCombiner::tryFactorization(InstCombiner::BuilderTy *Builder,
BinaryOperator &I,
/// Returns the simplified value, or null if it didn't simplify.
Value *InstCombiner::SimplifyUsingDistributiveLaws(BinaryOperator &I) {
Value *LHS = I.getOperand(0), *RHS = I.getOperand(1);
/// constant zero (which is the 'negate' form).
Value *InstCombiner::dyn_castNegVal(Value *V) const {
MacBook Pro

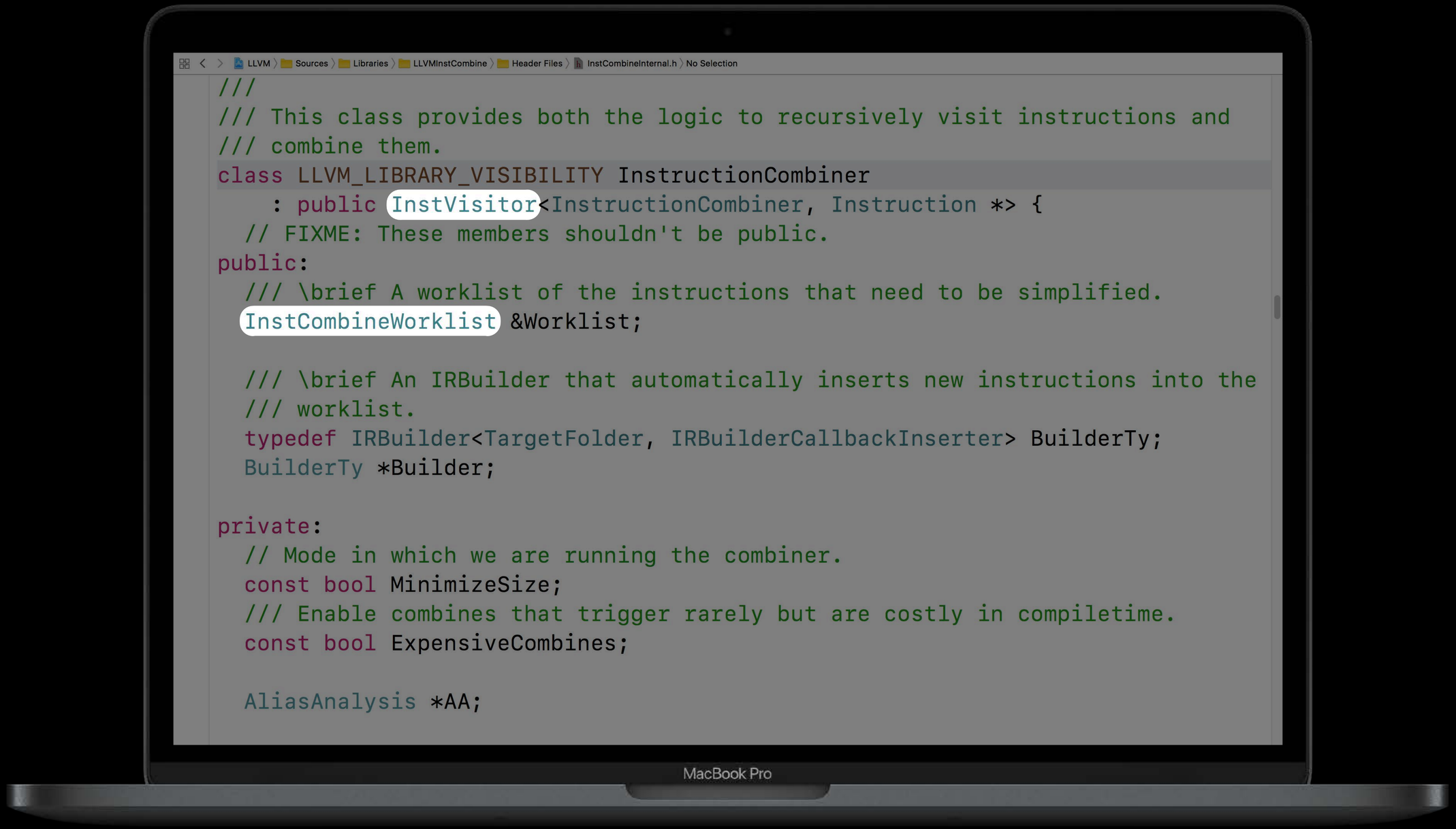

Rename
All (263)
Code (258)
File Names (0)
Comments (5)
Other (0)
Cancel
Rename
InstructionCombining.cpp
Value *InstructionCombiner::EmitGEPOffset(User *GEP) {
return llvm::EmitGEPOffset(Builder, DL, GEP);
/// optimizations for i1 types.
bool InstructionCombiner::shouldChangeType(unsigned FromWidth,
unsigned ToWidth) const {
/// i1 types.
bool InstructionCombiner::shouldChangeType(Type *From, Type *To) const {
assert(From->isIntegerTy() && To->isIntegerTy());
/// if C1 and C2 are constants.
bool InstructionCombiner::SimplifyAssociativeOrCommutative(BinaryOperator &I) {
Instruction::BinaryOps Opcode = I.getOpcode();
/// (e. g. "(A*B)+(A*C)" -> "A*(B+C)").
Value *InstructionCombiner::tryFactorization(InstructionCombiner::BuilderTy
*Builder,
BinaryOperator &I,
/// Returns the simplified value, or null if it didn't simplify.
Value *InstructionCombiner::SimplifyUsingDistributiveLaws(BinaryOperator &I) {
Value *LHS = I.getOperand(0), *RHS = I.getOperand(1);
/// constant zero (which is the 'negate' form).
MacBook Pro

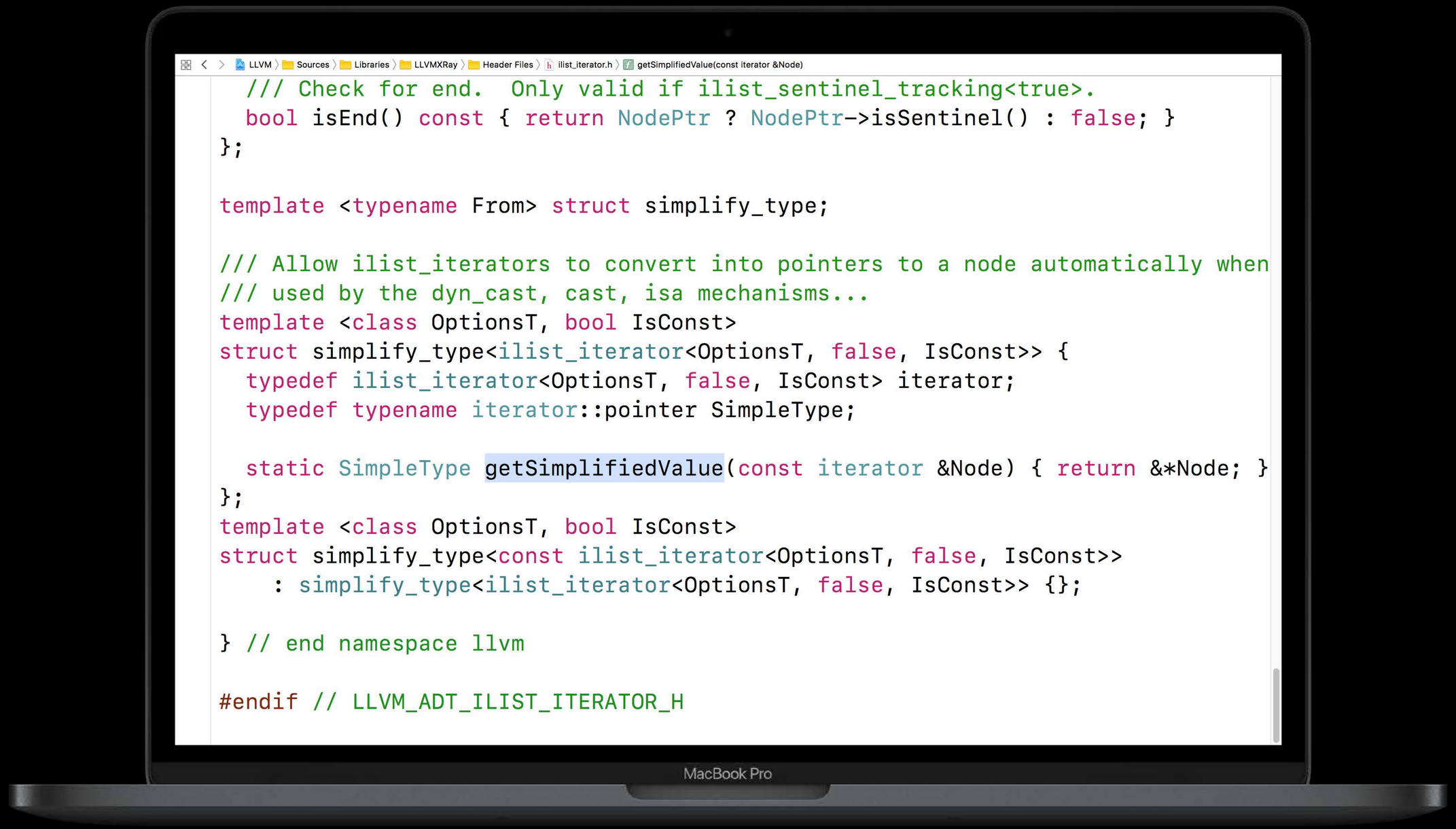

LLVM 〉 Sources 〉 Libraries 〉 LLVMInstCombine 〉 Source Files 〉 InstructionCombining.cpp 〉 InstructionCombiner::shouldChangeType(unsigned FromWidth, unsigned ToWidth)
return llvm::EmitGEPOffset(Builder, DL, GEP);
}
/// Return true if it is desirable to convert an integer computation from a
/// given bit width to a new bit width.
/// We don't want to convert from a legal to an illegal type or from a smaller
/// to a larger illegal type. A width of '1' is always treated as a legal type
/// because i1 is a fundamental type in IR, and there are many specialized
/// optimizations for i1 types.
bool InstructionCombiner::shouldChangeType(unsigned FromWidth,
unsigned ToWidth) const {
bool FromLegal = FromWidth == 1 || DL.isLegalInteger(FromWidth);
bool ToLegal = ToWidth == 1 || DL.isLegalInteger(ToWidth);
// If this is a legal integer from type, and the result would be an illegal
// type, don't do the transformation.
if (FromLegal && !ToLegal)
return false;
// Otherwise, if both are illegal, do not increase the size of the result. We
// do allow things like i160 -> i64, but not i64 -> i160.
if (!FromLegal && !ToLegal && ToWidth > FromWidth)
return false;
MacBook Pro

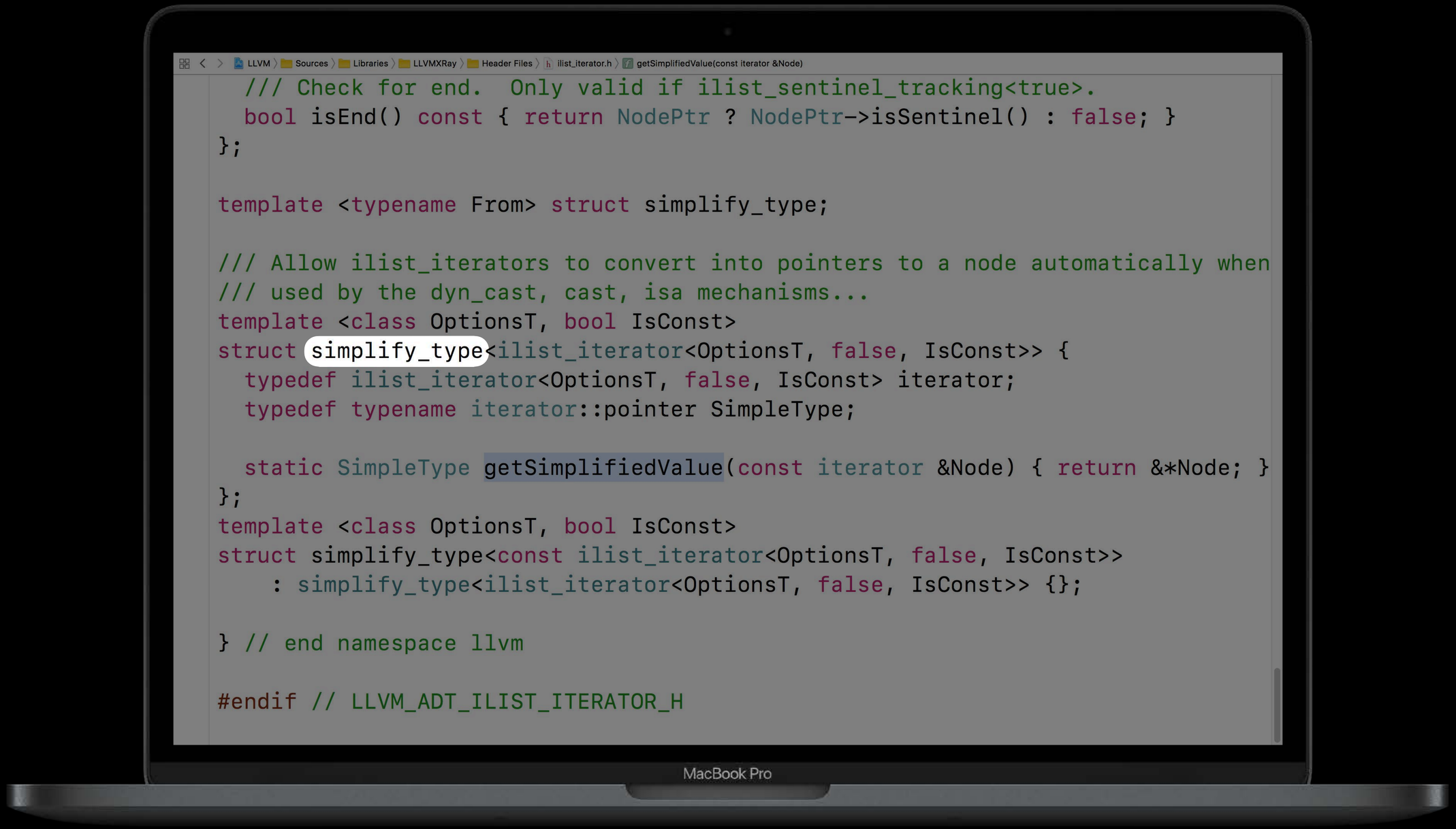

LLVM 〉 Sources 〉 Libraries 〉 LLVMInstCombine 〉 Header Files 〉 InstCombineInternal.h 〉 No Selection
///
/// This class provides both the logic to recursively visit instructions and
/// combine them.
class LLVM_LIBRARY_VISIBILITY InstructionCombiner
: public InstVisitor<InstructionCombiner, Instruction *> {
// FIXME: These members shouldn't be public.
public:
/// \brief A worklist of the instructions that need to be simplified.
InstCombineWorklist &Worklist;
/// \brief An IRBuilder that automatically inserts new instructions into the
/// worklist.
typedef IRBuilder<TargetFolder, IRBuilderCallbackInserter> BuilderTy;
BuilderTy *Builder;
private:
// Mode in which we are running the combiner.
const bool MinimizeSize;
/// Enable combines that trigger rarely but are costly in compiletime.
const bool ExpensiveCombines;
AliasAnalysis *AA;
MacBook Pro

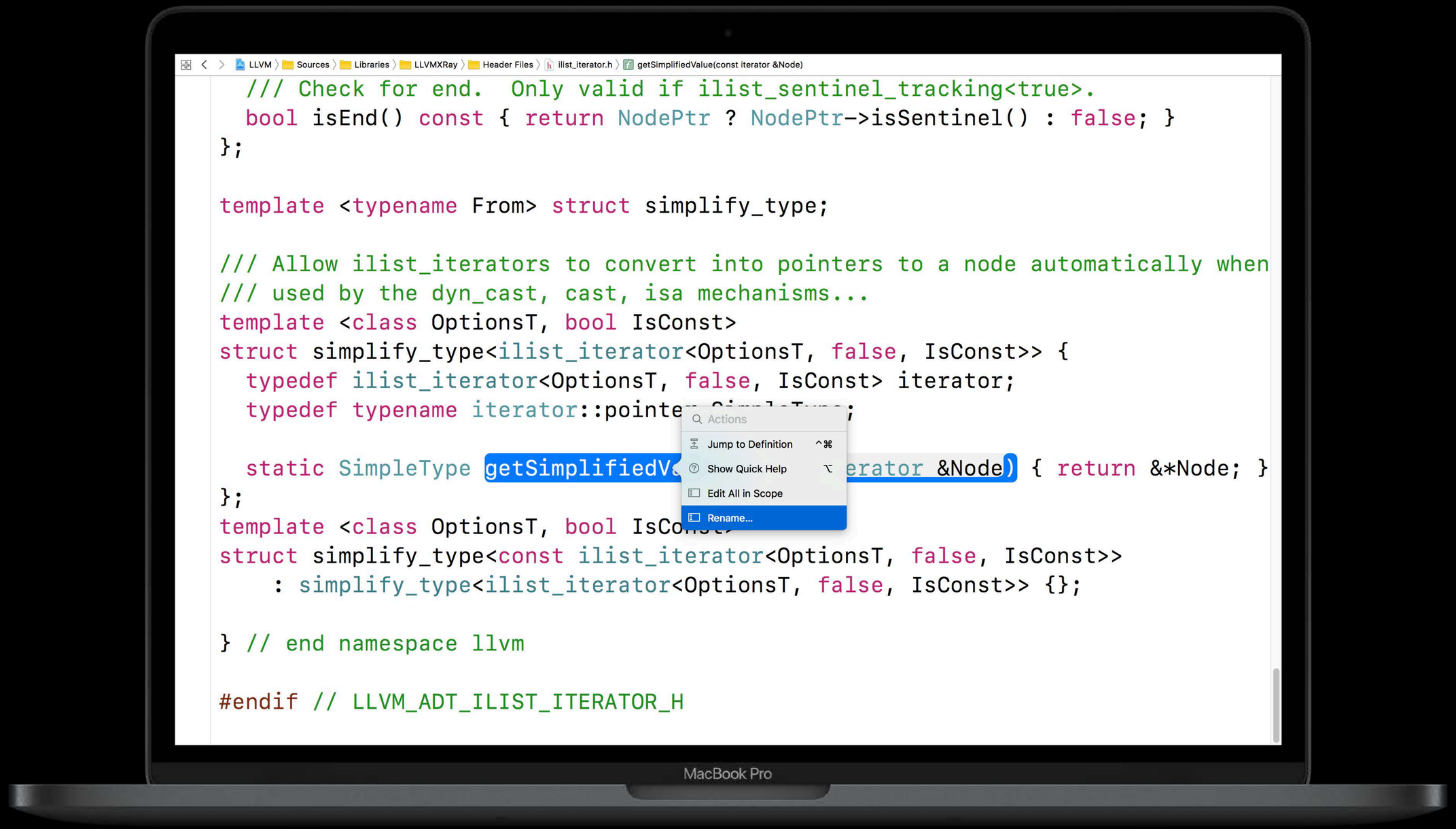

LLVM 〉 Sources 〉 Libraries 〉 LLVMInstCombine 〉 Header Files 〉 InstCombineInternal.h 〉 No Selection
///
/// This class provides both the logic to recursively visit instructions and
/// combine them.
class LLVM_LIBRARY_VISIBILITY InstructionCombiner
: public InstVisitor<InstructionCombiner, Instruction *> {
// FIXME: These members shouldn't be public.
public:
/// \brief A worklist of the instructions that need to be simplified.
InstCombineWorklist &Worklist;
/// \brief An IRBuilder that automatically inserts new instructions into the
/// worklist.
typedef IRBuilder<TargetFolder, IRBuilderCallbackInserter> BuilderTy;
BuilderTy *Builder;
private:
// Mode in which we are running the combiner.
const bool MinimizeSize;
/// Enable combines that trigger rarely but are costly in compiletime.
const bool ExpensiveCombines;
AliasAnalysis *AA;
MacBook Pro

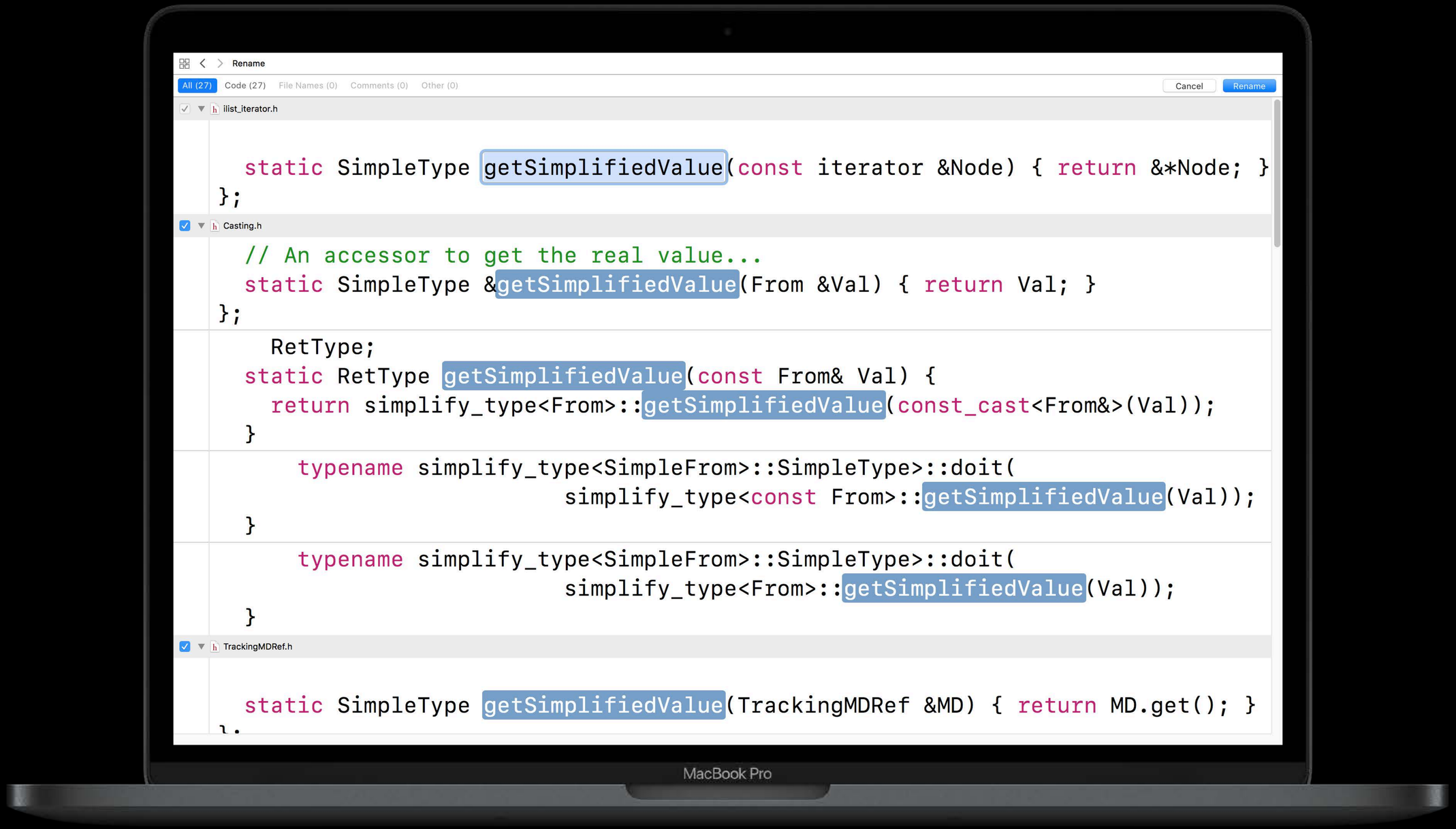

LLVM 〉 Sources 〉 Libraries 〉 LLVMInstCombine 〉 Header Files 〉 InstCombineInternal.h 〉 No Selection
///
/// This class provides both the logic to recursively visit instructions and
/// combine them.
class LLVM_LIBRARY_VISIBILITY InstructionCombiner
: public InstVisitor<InstructionCombiner, Instruction *> {
// FIXME: These members shouldn't be public.
public:
/// \brief A worklist of the instructions that need to be simplified.
InstCombineWorklist &Worklist;
/// \brief An IRBuilder that automatically inserts new instructions into the
/// worklist.
typedef IRBuilder<TargetFolder, IRBuilderCallbackInserter> BuilderTy;
BuilderTy *Builder;
private:
// Mode in which we are running the combiner.
const bool MinimizeSize;
/// Enable combines that trigger rarely but are costly in compiletime.
const bool ExpensiveCombines;
AliasAnalysis *AA;
MacBook Pro

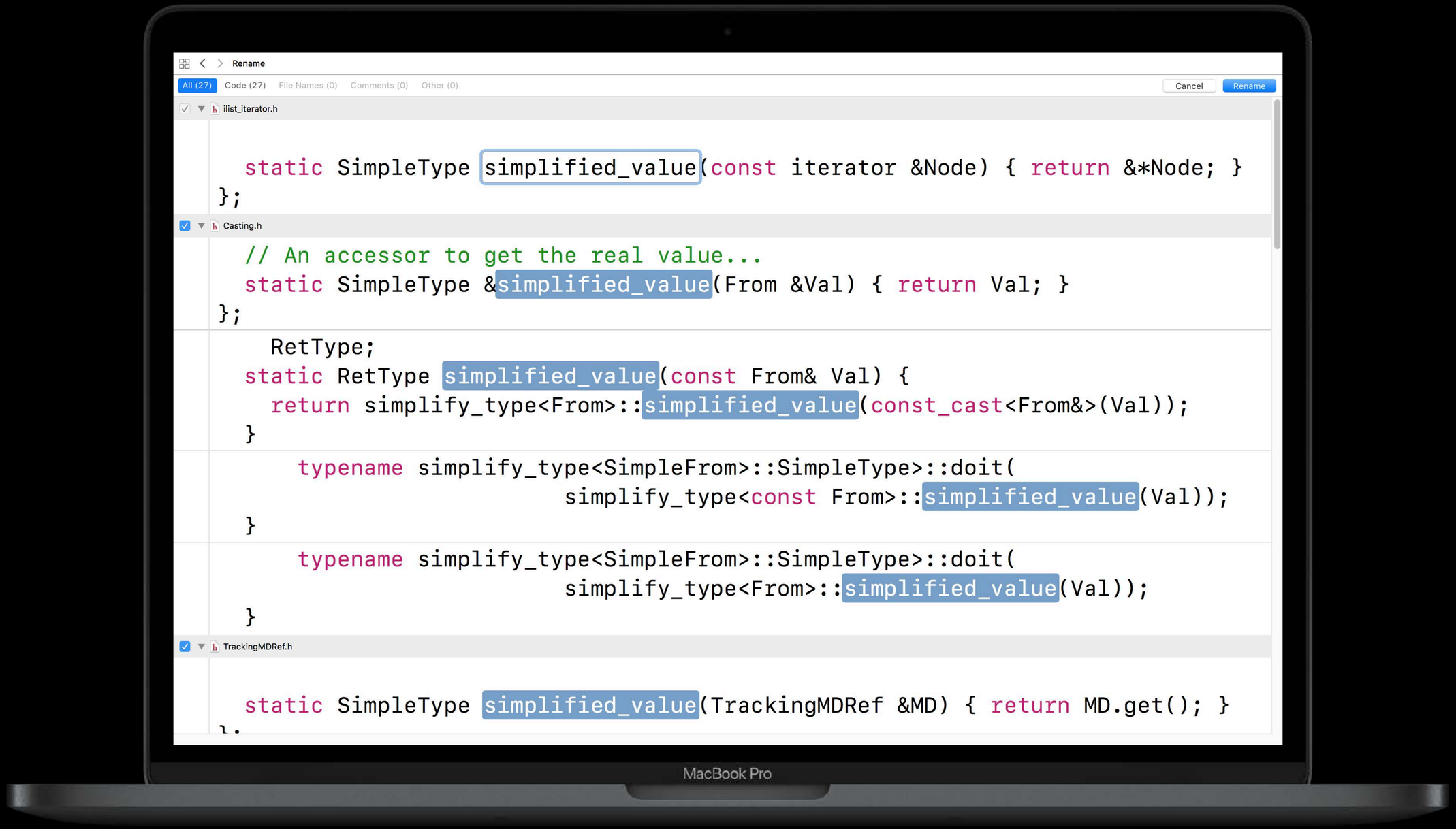

LLVM 〉 Sources 〉 Libraries 〉 LLVMInstCombine 〉 Header Files 〉 InstCombineInternal.h 〉 No Selection
///
/// This class provides both the logic to recursively visit instructions and
/// combine them.
class LLVM_LIBRARY_VISIBILITY InstructionCombiner
: public InstVisitor<InstructionCombiner, Instruction *> {
// FIXME: These members shouldn't be public.
public:
/// \brief A worklist of the instructions that need to be simplified.
InstCombineWorklist &Worklist;
/// \brief An IRBuilder that automatically inserts new instructions into the
/// worklist.
typedef IRBuilder<TargetFolder, IRBuilderCallbackInserter> BuilderTy;
BuilderTy *Builder;
private:
// Mode in which we are running the combiner.
const bool MinimizeSize;
/// Enable combines that trigger rarely but are costly in compiletime.
const bool ExpensiveCombines;
AliasAnalysis *AA;
MacBook Pro

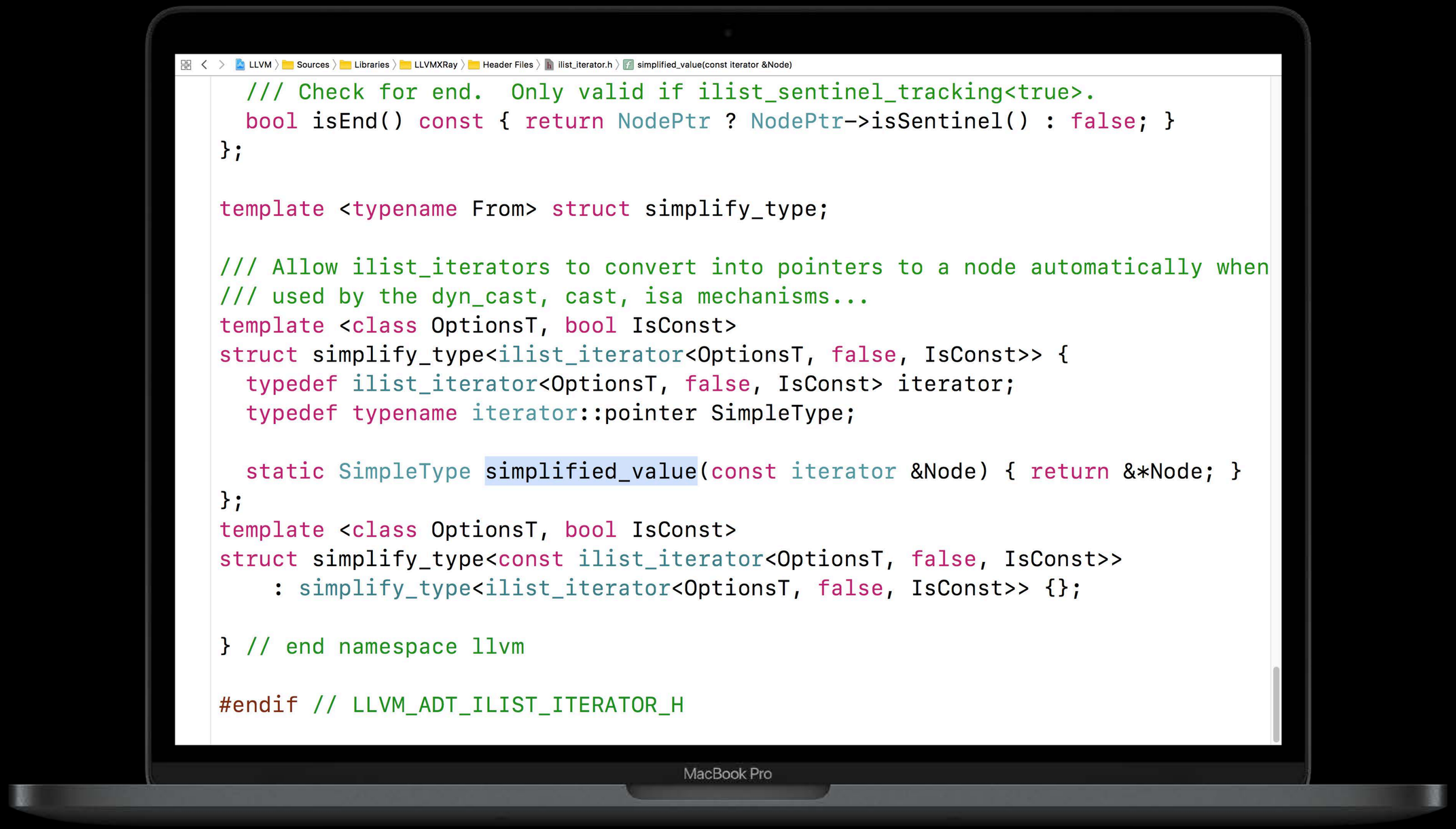

LLVM 〉 Sources 〉 Libraries 〉 LLVMXRay 〉 Header Files 〉 ilist_iterator.h 〉 getSimplifiedValue(const iterator &Node)
/// Check for end. Only valid if ilist_sentinel_tracking<true>.
bool isEnd() const { return NodePtr ? NodePtr->isSentinel() : false; }
};
template <typename From> struct simplify_type;
/// Allow ilist_iterators to convert into pointers to a node automatically when
/// used by the dyn_cast, cast, isa mechanisms...
template <class OptionsT, bool IsConst>
struct simplify_type<ilist_iterator<OptionsT, false, IsConst>> {
typedef ilist_iterator<OptionsT, false, IsConst> iterator;
typedef typename iterator::pointer SimpleType;
static SimpleType getSimplifiedValue(const iterator &Node) { return &*Node; }
};
template <class OptionsT, bool IsConst>
struct simplify_type<const ilist_iterator<OptionsT, false, IsConst>>
: simplify_type<ilist_iterator<OptionsT, false, IsConst>> {};
} // end namespace llvm
#endif // LLVM_ADT_ILIST_ITERATOR_H
MacBook Pro

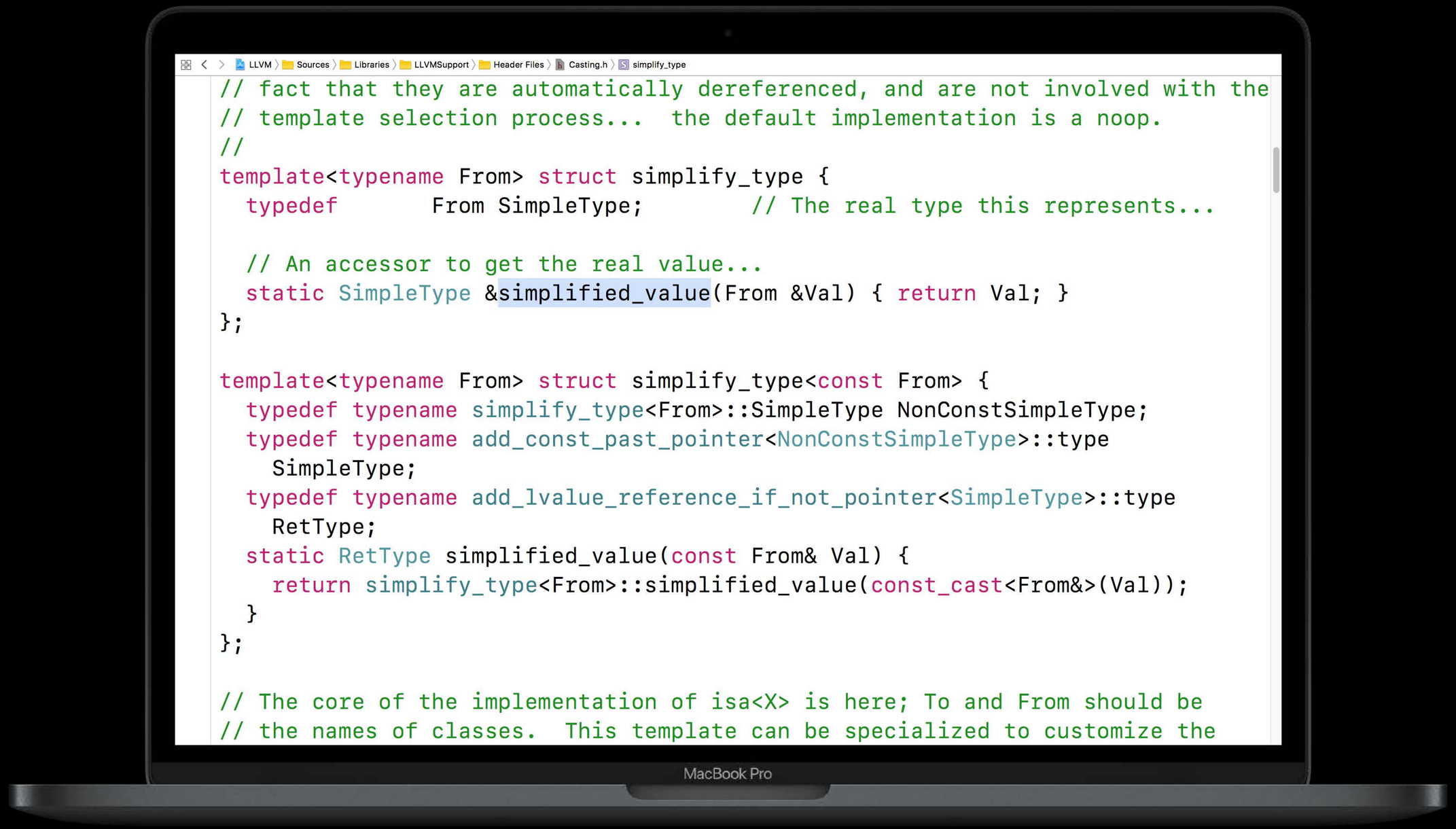

LLVM Sources Libraries LLVMXRay Header Files ilist_iterator.h getSimplifiedValue(const iterator &Node)
/// Check for end. Only valid if ilist_sentinel_tracking<true>.
bool isEnd() const { return NodePtr ? NodePtr->isSentinel() : false; }
};
template <typename From> struct simplify_type;
/// Allow ilist_iterators to convert into pointers to a node automatically when
/// used by the dyn_cast, cast, isa mechanisms...
template <class OptionsT, bool IsConst>
struct simplify_type<ilist_iterator<OptionsT, false, IsConst>> {
typedef ilist_iterator<OptionsT, false, IsConst> iterator;
typedef typename iterator::pointer SimpleType;
static SimpleType getSimplifiedValue(const iterator &Node) { return &*Node; }
};
template <class OptionsT, bool IsConst>
struct simplify_type<const ilist_iterator<OptionsT, false, IsConst>>
: simplify_type<ilist_iterator<OptionsT, false, IsConst>> {};
} // end namespace llvm
#endif // LLVM_ADT_ILIST_ITERATOR_H
MacBook Pro

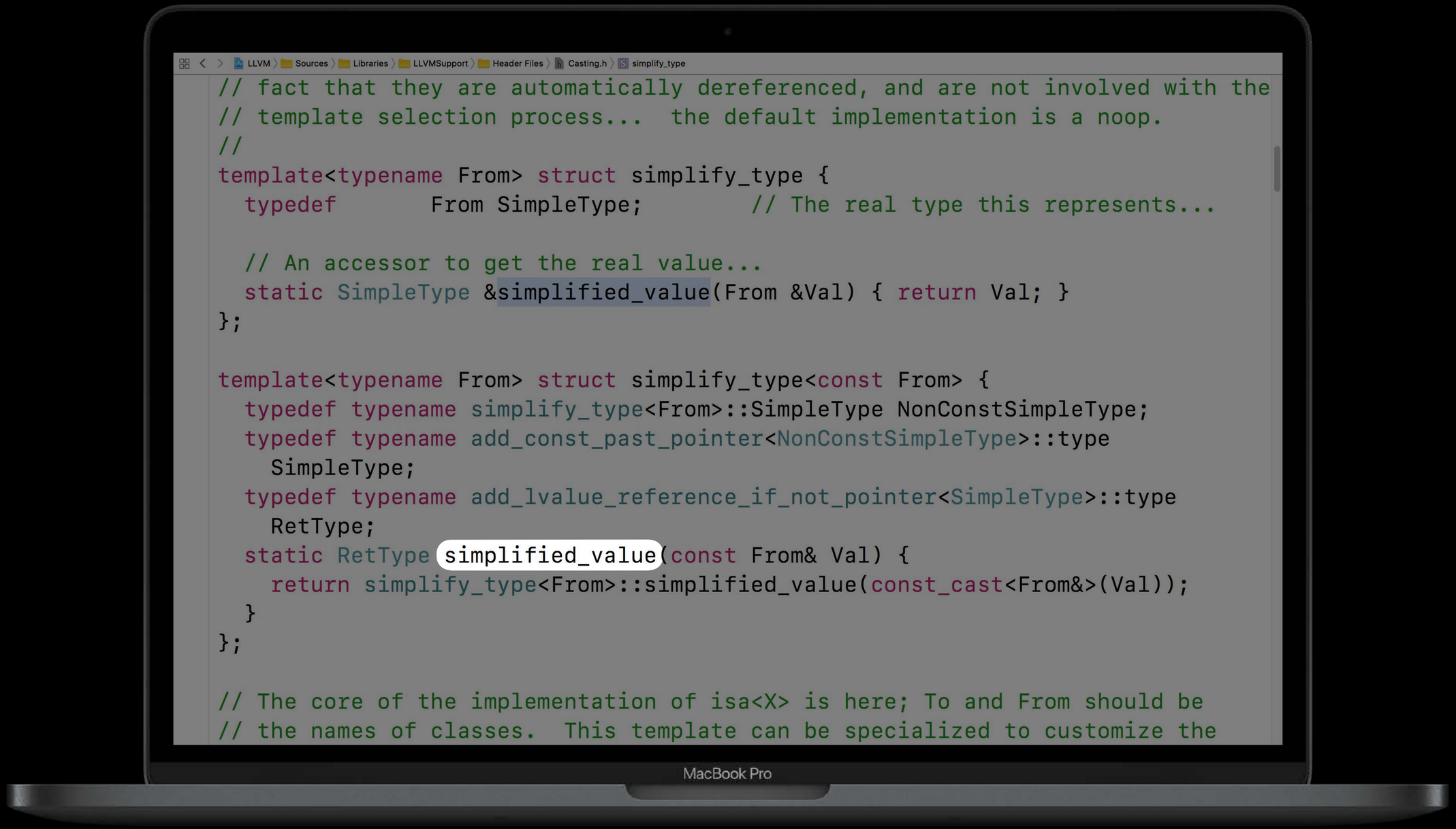

LLVM 〉 Sources 〉 Libraries 〉 LLVMXRay 〉 Header Files 〉 ilist_iterator.h 〉 getSimplifiedValue(const iterator &Node)
/// Check for end. Only valid if ilist_sentinel_tracking<true>.
bool isEnd() const { return NodePtr ? NodePtr->isSentinel() : false; }
};
template <typename From> struct simplify_type;
/// Allow ilist_iterators to convert into pointers to a node automatically when
/// used by the dyn_cast, cast, isa mechanisms...
template <class OptionsT, bool IsConst>
struct simplify_type<ilist_iterator<OptionsT, false, IsConst>> {
typedef ilist_iterator<OptionsT, false, IsConst> iterator;
typedef typename iterator::pointe
static SimpleType getSimplifiedV
erator &Node) { return &*Node; }
};
template <class OptionsT, bool IsCo
struct simplify_type<const ilist_iterator<OptionsT, false, IsConst>>
: simplify_type<ilist_iterator<OptionsT, false, IsConst>> {};
} // end namespace llvm
#endif // LLVM_ADT_ILIST_ITERATOR_H
Actions
Jump to Definition ^⌘
Show Quick Help ⌥
Edit All in Scope
Rename...
MacBook Pro

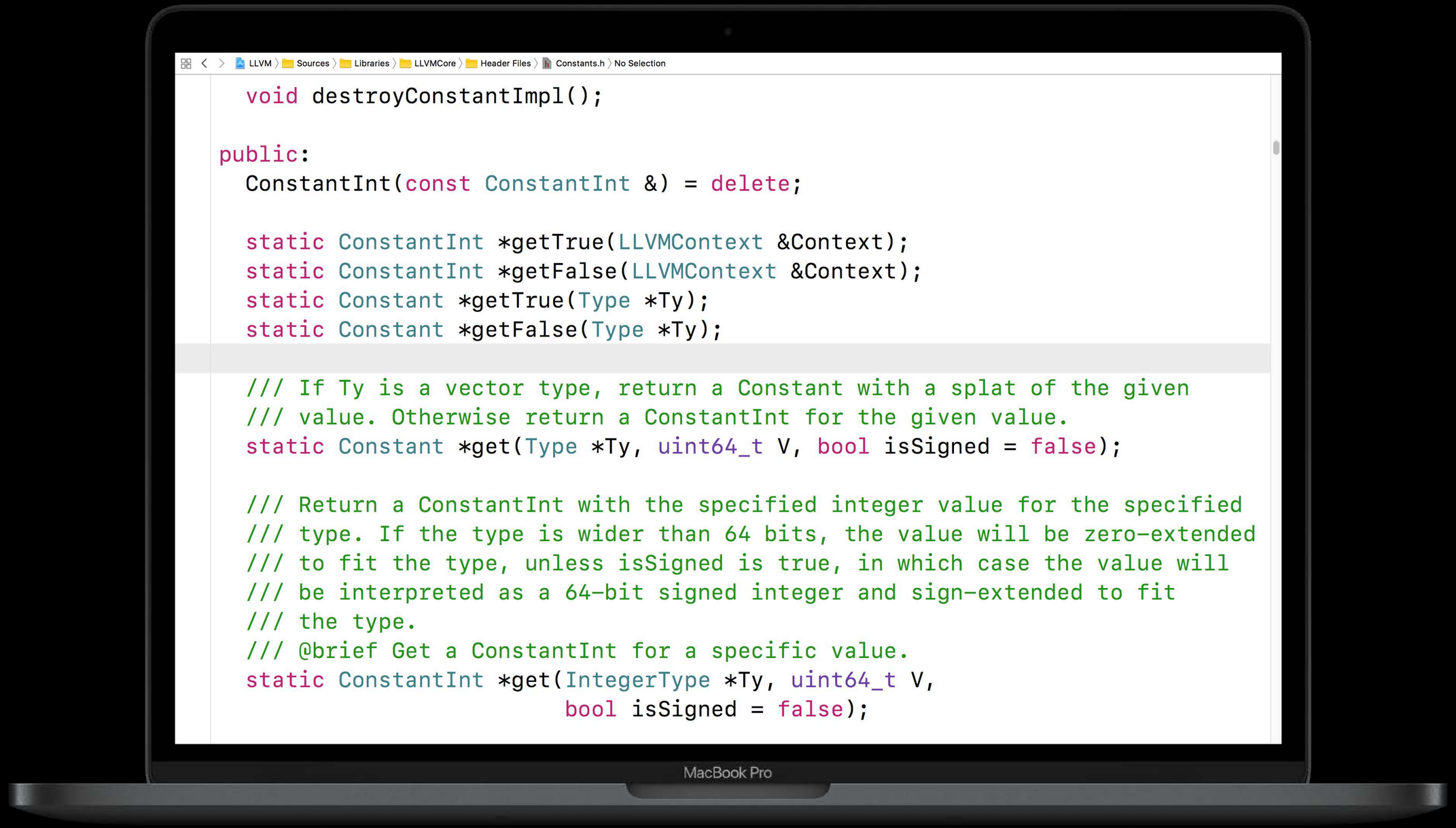

Rename
All (27)
Code (27)
File Names (0)
Comments (0)
Other (0)
Cancel
Rename
ilist_iterator.h
static SimpleType getSimplifiedValue(const iterator &Node) { return &*Node; }
};
Casting.h
// An accessor to get the real value...
static SimpleType &getSimplifiedValue(From &Val) { return Val; }
};
RetType;
static RetType getSimplifiedValue(const From& Val) {
return simplify_type<From>::getSimplifiedValue(const_cast<From&>(Val));
}
typename simplify_type<SimpleFrom>::SimpleType>::doit(
simplify_type<const From>::getSimplifiedValue(Val));
}
typename simplify_type<SimpleFrom>::SimpleType>::doit(
simplify_type<From>::getSimplifiedValue(Val));
}
TrackingMDRef.h
static SimpleType getSimplifiedValue(TrackingMDRef &MD) { return MD.get(); }
MacBook Pro

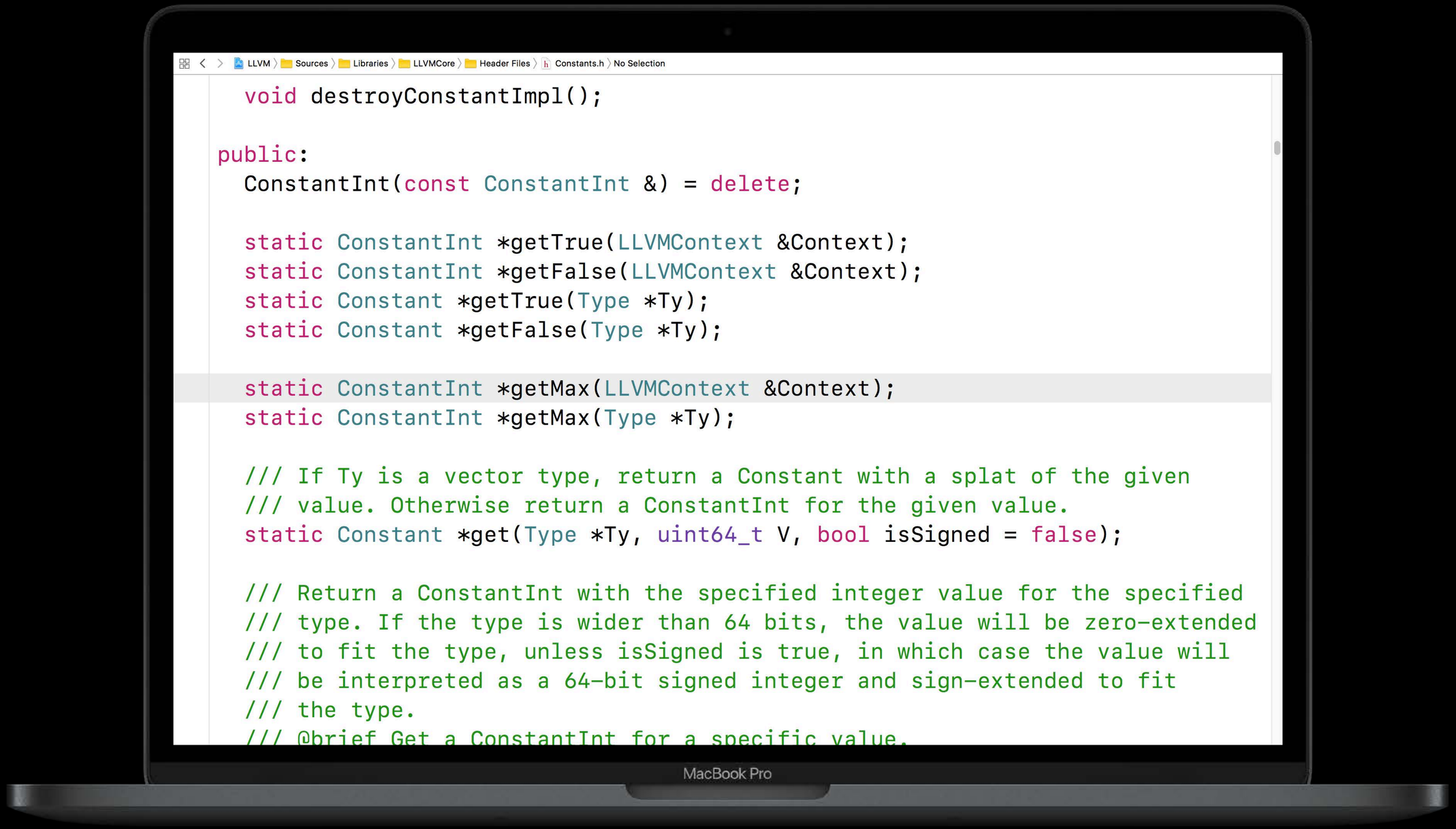

Rename
All (27)
Code (27)
File Names (0)
Comments (0)
Other (0)
Cancel
Rename
ilist_iterator.h
static SimpleType simplified_value(const iterator &Node) { return &*Node; }
};
Casting.h
// An accessor to get the real value...
static SimpleType &simplified_value(From &Val) { return Val; }
};
RetType;
static RetType simplified_value(const From& Val) {
return simplify_type<From>::simplified_value(const_cast<From&>(Val));
}
typename simplify_type<SimpleFrom>::SimpleType>::doit(
simplify_type<const From>::simplified_value(Val));
}
typename simplify_type<SimpleFrom>::SimpleType>::doit(
simplify_type<From>::simplified_value(Val));
}
TrackingMDRef.h
static SimpleType simplified_value(TrackingMDRef &MD) { return MD.get(); }
MacBook Pro


LLVM 〉 Sources 〉 Libraries 〉 LLVMXRay 〉 Header Files 〉 ilist_iterator.h 〉 simplified_value(const iterator &Node)
/// Check for end. Only valid if ilist_sentinel_tracking<true>.
bool isEnd() const { return NodePtr ? NodePtr->isSentinel() : false; }
};
template <typename From> struct simplify_type;
/// Allow ilist_iterators to convert into pointers to a node automatically when
/// used by the dyn_cast, cast, isa mechanisms...
template <class OptionsT, bool IsConst>
struct simplify_type<ilist_iterator<OptionsT, false, IsConst>> {
typedef ilist_iterator<OptionsT, false, IsConst> iterator;
typedef typename iterator::pointer SimpleType;
static SimpleType simplified_value(const iterator &Node) { return &*Node; }
};
template <class OptionsT, bool IsConst>
struct simplify_type<const ilist_iterator<OptionsT, false, IsConst>>
: simplify_type<ilist_iterator<OptionsT, false, IsConst>> {};
} // end namespace llvm
#endif // LLVM_ADT_ILIST_ITERATOR_H
MacBook Pro

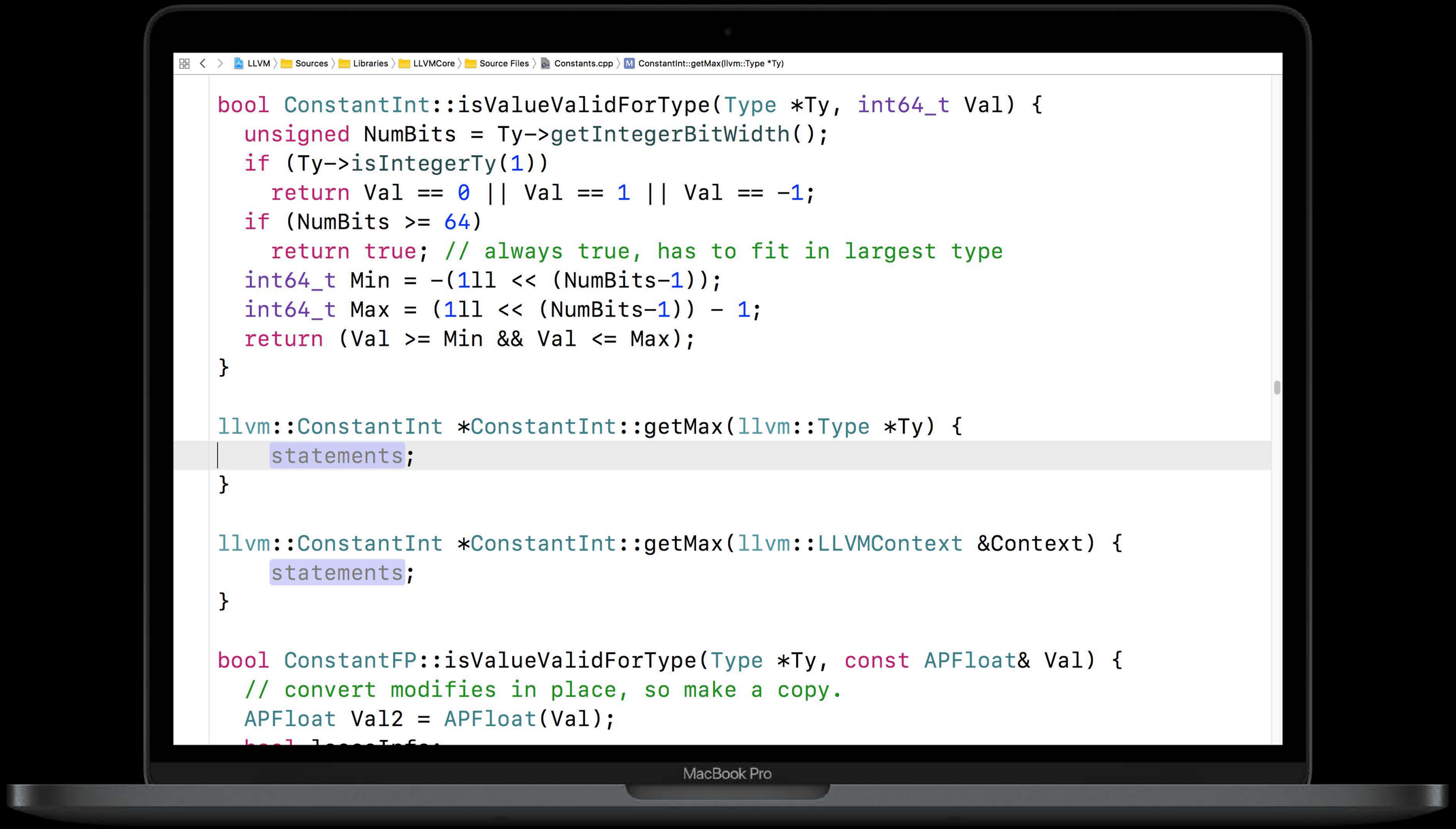

LLVM 〉 Sources 〉 Libraries 〉 LLVMSupport 〉 Header Files 〉 Casting.h 〉 simplify_type
// fact that they are automatically dereferenced, and are not involved with the
// template selection process... the default implementation is a noop.
//
template<typename From> struct simplify_type {
typedef From SimpleType; // The real type this represents...
// An accessor to get the real value...
static SimpleType &simplified_value(From &Val) { return Val; }
};
template<typename From> struct simplify_type<const From> {
typedef typename simplify_type<From>::SimpleType NonConstSimpleType;
typedef typename add_const_past_pointer<NonConstSimpleType>::type
SimpleType;
typedef typename add_lvalue_reference_if_not_pointer<SimpleType>::type
RetType;
static RetType simplified_value(const From& Val) {
return simplify_type<From>::simplified_value(const_cast<From&>(Val));
}
};
// The core of the implementation of isa<X> is here; To and From should be
// the names of classes. This template can be specialized to customize the
MacBook Pro


LLVM
Sources
Libraries
LLVMSupport
Header Files
Casting.h
simplify_type
// fact that they are automatically dereferenced, and are not involved with the
// template selection process... the default implementation is a noop.
//
template<typename From> struct simplify_type {
typedef From SimpleType; // The real type this represents...
// An accessor to get the real value...
static SimpleType &simplified_value(From &Val) { return Val; }
};
template<typename From> struct simplify_type<const From> {
typedef typename simplify_type<From>::SimpleType NonConstSimpleType;
typedef typename add_const_past_pointer<NonConstSimpleType>::type
SimpleType;
typedef typename add_lvalue_reference_if_not_pointer<SimpleType>::type
RetType;
static RetType simplified_value(const From& Val) {
return simplify_type<From>::simplified_value(const_cast<From&>(Val));
}
};
// The core of the implementation of isa<X> is here; To and From should be
// the names of classes. This template can be specialized to customize the
MacBook Pro


LLVM 〉 Sources 〉 Libraries 〉 LLVMCore 〉 Header Files 〉 Constants.h 〉 No Selection
void destroyConstantImpl();
public:
ConstantInt(const ConstantInt &) = delete;
static ConstantInt *getTrue(LLVMContext &Context);
static ConstantInt *getFalse(LLVMContext &Context);
static Constant *getTrue(Type *Ty);
static Constant *getFalse(Type *Ty);
/// If Ty is a vector type, return a Constant with a splat of the given
/// value. Otherwise return a ConstantInt for the given value.
static Constant *get(Type *Ty, uint64_t V, bool isSigned = false);
/// Return a ConstantInt with the specified integer value for the specified
/// type. If the type is wider than 64 bits, the value will be zero-extended
/// to fit the type, unless isSigned is true, in which case the value will
/// be interpreted as a 64-bit signed integer and sign-extended to fit
/// the type.
/// @brief Get a ConstantInt for a specific value.
static ConstantInt *get(IntegerType *Ty, uint64_t V,
bool isSigned = false);
MacBook Pro

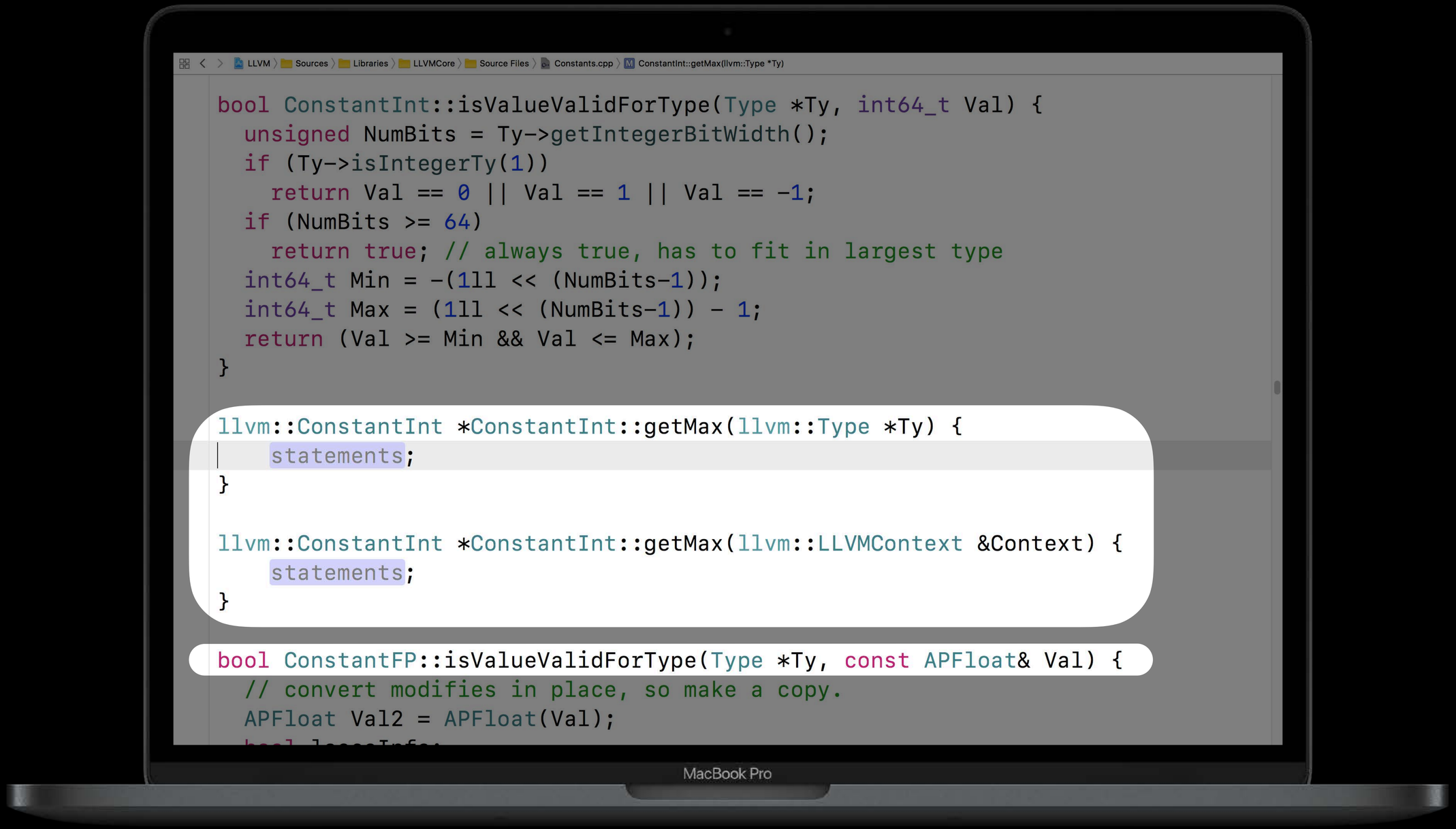

LLVM 〉 Sources 〉 Libraries 〉 LLVMCore 〉 Header Files 〉 Constants.h 〉 No Selection
void destroyConstantImpl();
public:
ConstantInt(const ConstantInt &) = delete;
static ConstantInt *getTrue(LLVMContext &Context);
static ConstantInt *getFalse(LLVMContext &Context);
static Constant *getTrue(Type *Ty);
static Constant *getFalse(Type *Ty);
static ConstantInt *getMax(LLVMContext &Context);
static ConstantInt *getMax(Type *Ty);
/// If Ty is a vector type, return a Constant with a splat of the given
/// value. Otherwise return a ConstantInt for the given value.
static Constant *get(Type *Ty, uint64_t V, bool isSigned = false);
/// Return a ConstantInt with the specified integer value for the specified
/// type. If the type is wider than 64 bits, the value will be zero-extended
/// to fit the type, unless isSigned is true, in which case the value will
/// be interpreted as a 64-bit signed integer and sign-extended to fit
/// the type.
/// @brief Get a ConstantInt for a specific value.
MacBook Pro

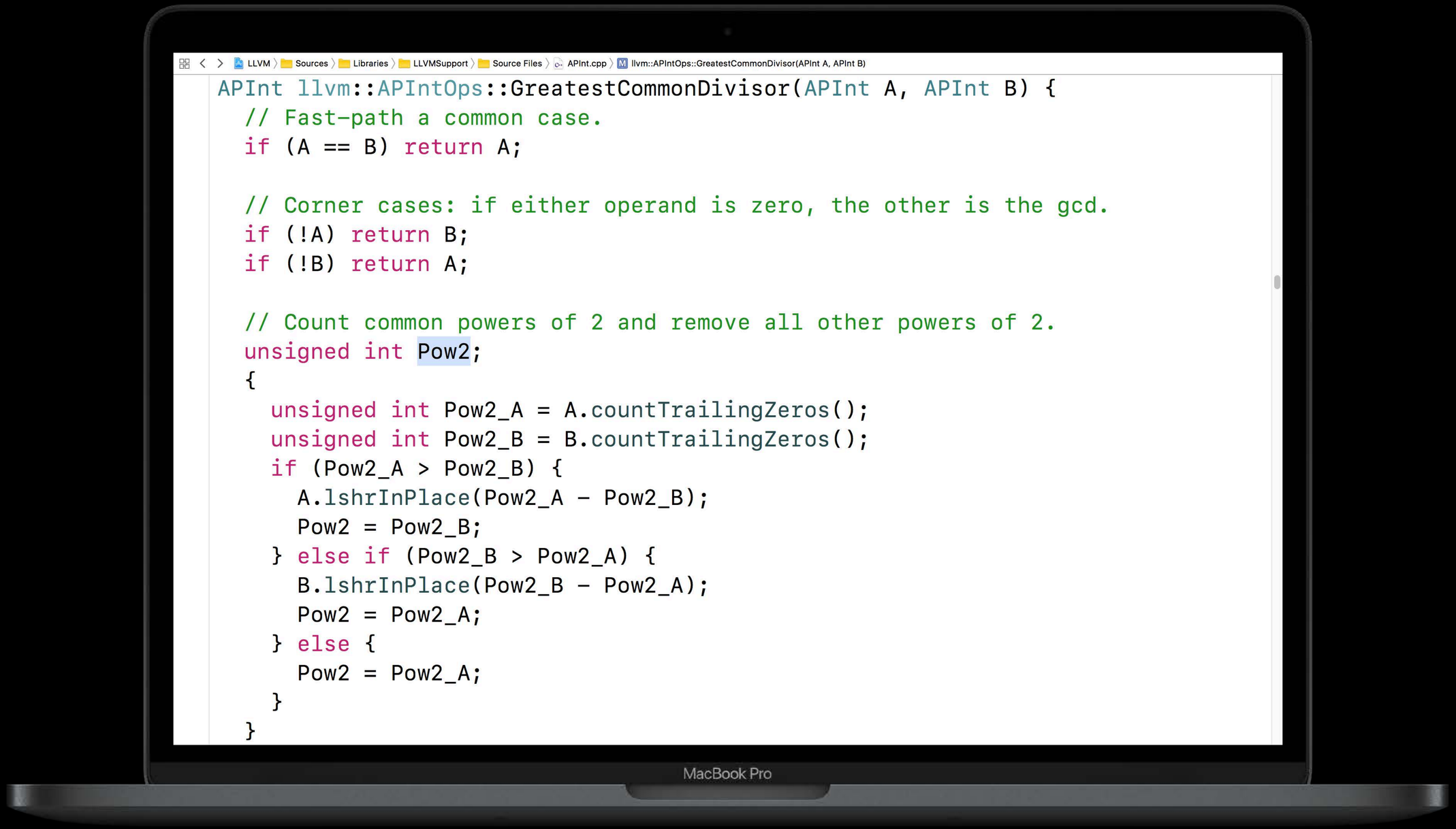

LLVM 〉 Sources 〉 Libraries 〉 LLVMCore 〉 Header Files 〉 Constants.h 〉 No Selection
void destroyConstantImpl();
public:
ConstantInt(const ConstantInt &) = delete;
static ConstantInt *getTrue(LLVMContext &Context);
static ConstantInt *getFalse(LLVMContext &Context);
static Constant *getTrue(Type *Ty);
static Constant *getFalse(Ty
static ConstantInt *getMax(
static ConstantInt *getMax(
Actions
Jump to Definition ^⌘
Show Quick Help ⌥
Edit All in Scope
Rename...
Generate Missing Function Definitions
Generate Missing Function Definitions
t);
/// If Ty is a vector type, return a Constant with a splat of the given
/// value. Otherwise return a ConstantInt for the given value.
static Constant *get(Type *Ty, uint64_t V, bool isSigned = false);
/// Return a ConstantInt with the specified integer value for the specified
/// type. If the type is wider than 64 bits, the value will be zero-extended
/// to fit the type, unless isSigned is true, in which case the value will
/// be interpreted as a 64-bit signed integer and sign-extended to fit
/// the type.
/// @brief Get a ConstantInt for a specific value.
MacBook Pro

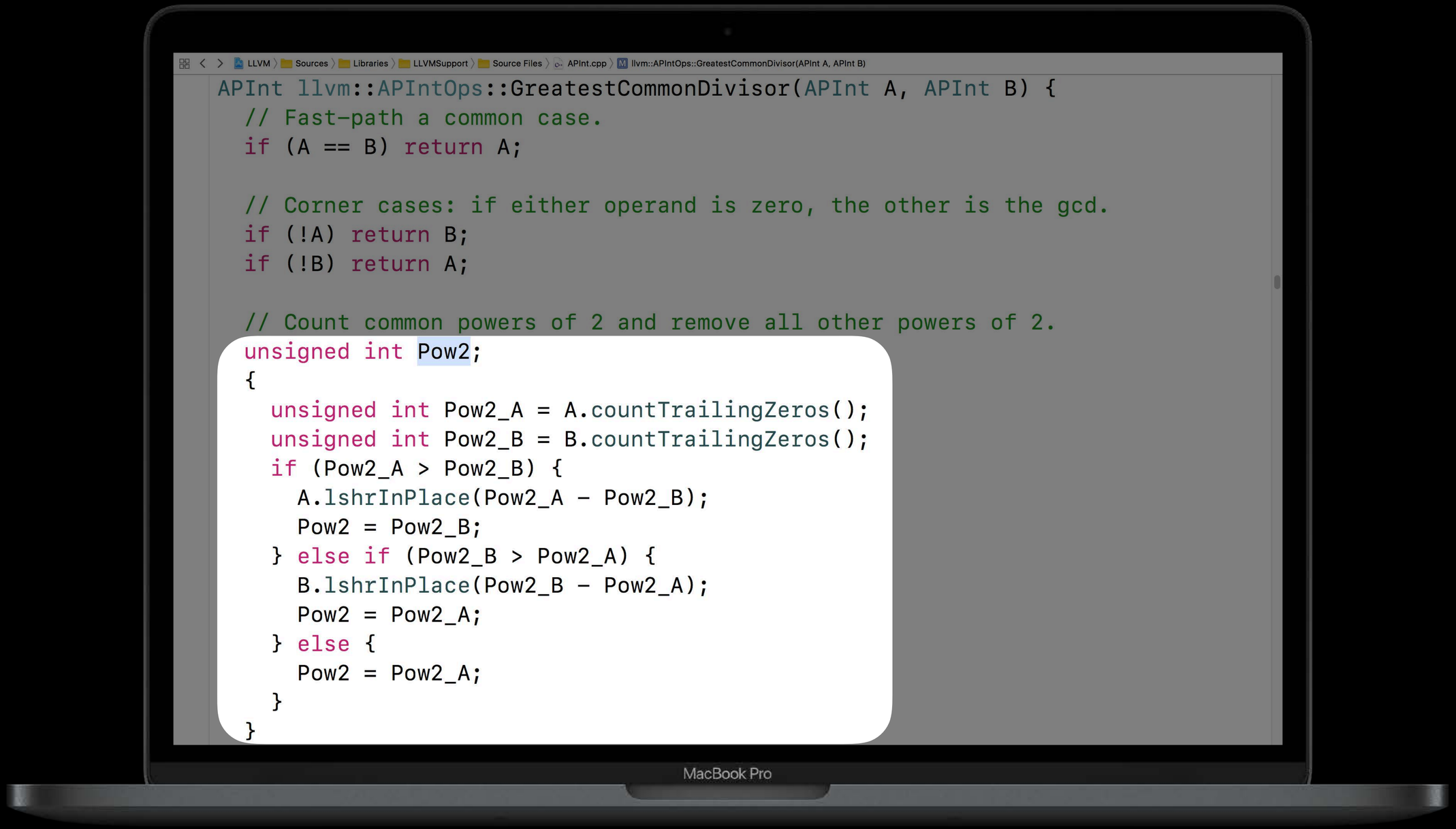

LLVM 〉 Sources 〉 Libraries 〉 LLVMCore 〉 Source Files 〉 Constants.cpp 〉 ConstantInt::getMax(llvm::Type *Ty)
bool ConstantInt::isValueValidForType(Type *Ty, int64_t Val) {
unsigned NumBits = Ty->getIntegerBitWidth();
if (Ty->isIntegerTy(1))
return Val == 0 || Val == 1 || Val == -1;
if (NumBits >= 64)
return true; // always true, has to fit in largest type
int64_t Min = -(1ll << (NumBits-1));
int64_t Max = (1ll << (NumBits-1)) - 1;
return (Val >= Min && Val <= Max);
}
llvm::ConstantInt *ConstantInt::getMax(llvm::Type *Ty) {
statements;
}
llvm::ConstantInt *ConstantInt::getMax(llvm::LLVMContext &Context) {
statements;
}
bool ConstantFP::isValueValidForType(Type *Ty, const APFloat& Val) {
// convert modifies in place, so make a copy.
APFloat Val2 = APFloat(Val);
MacBook Pro

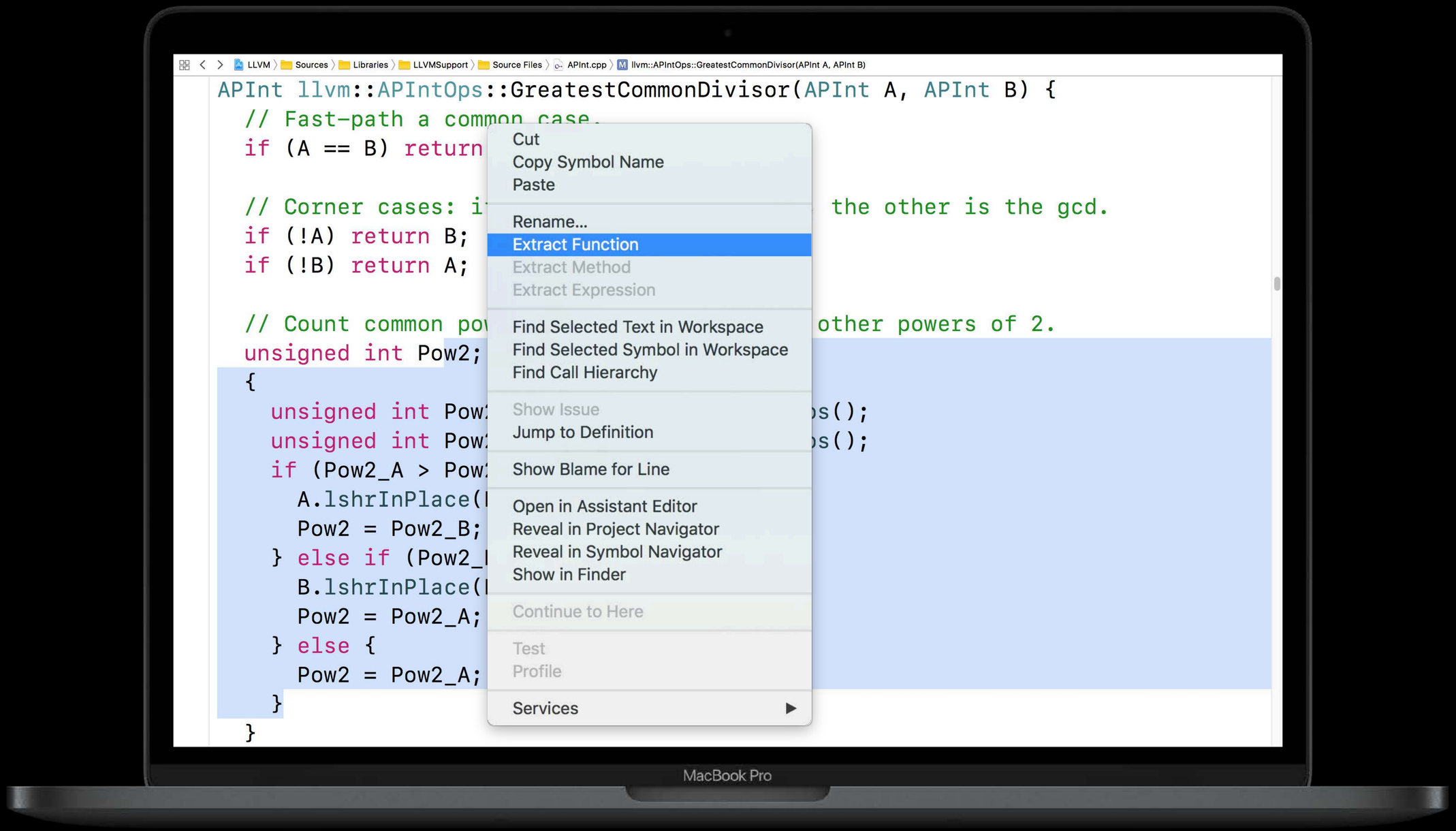

LLVM Sources Libraries LLVMCore Source Files Constants.cpp ConstantInt::getMax(llvm::Type *Ty)
bool ConstantInt::isValueValidForType(Type *Ty, int64_t Val) {
unsigned NumBits = Ty->getIntegerBitWidth();
if (Ty->isIntegerTy(1))
return Val == 0 || Val == 1 || Val == -1;
if (NumBits >= 64)
return true; // always true, has to fit in largest type
int64_t Min = -(1ll << (NumBits-1));
int64_t Max = (1ll << (NumBits-1)) - 1;
return (Val >= Min && Val <= Max);
}
llvm::ConstantInt *ConstantInt::getMax(llvm::Type *Ty) {
statements;
}
llvm::ConstantInt *ConstantInt::getMax(llvm::LLVMContext &Context) {
statements;
}
bool ConstantFP::isValueValidForType(Type *Ty, const APFloat& Val) {
// convert modifies in place, so make a copy.
APFloat Val2 = APFloat(Val);
MacBook Pro

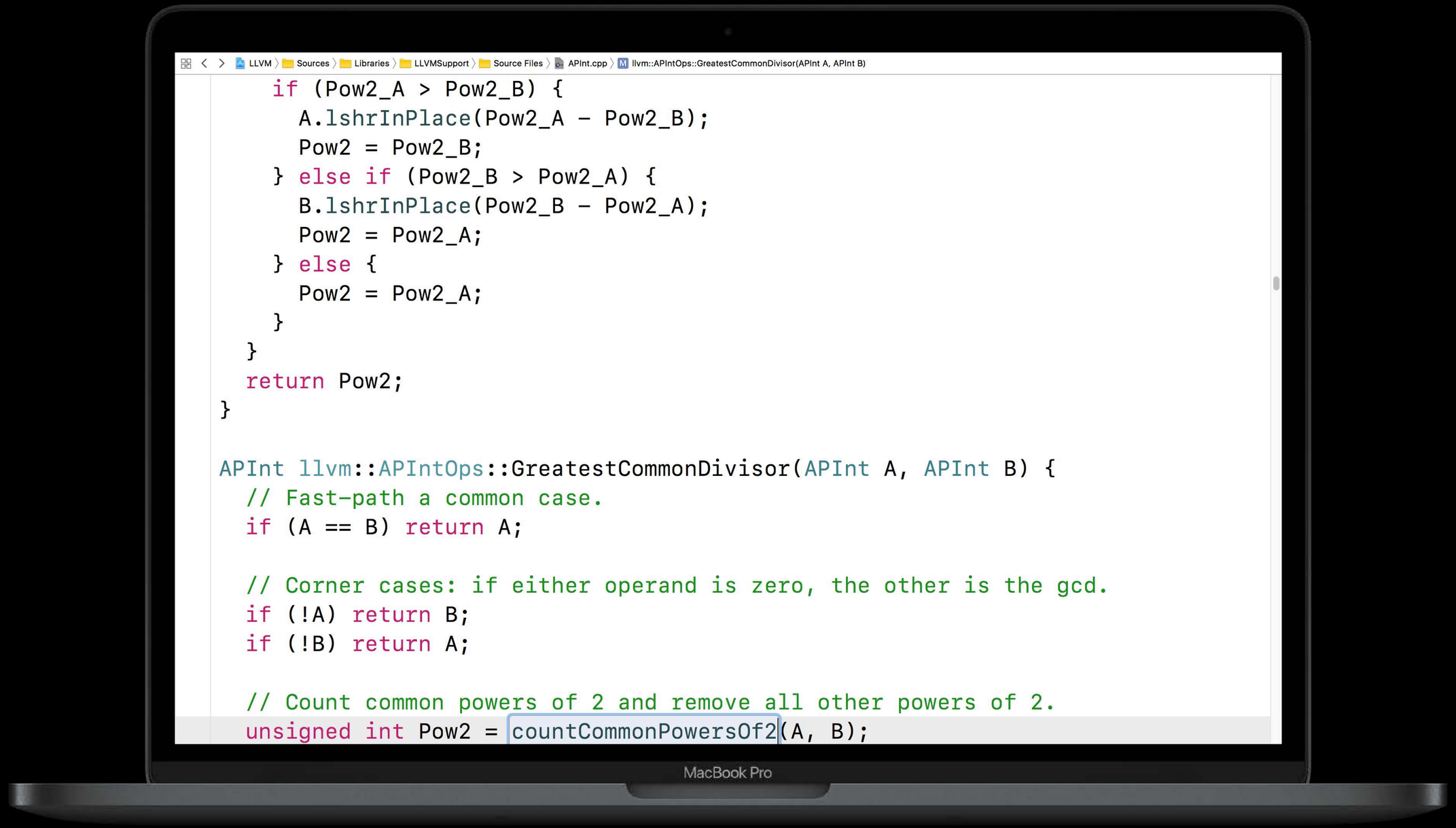

LLVM Sources Libraries LLVMCore Source Files Constants.cpp ConstantInt::getMax(llvm::Type *Ty)
bool ConstantInt::isValueValidForType(Type *Ty, int64_t Val) {
unsigned NumBits = Ty->getIntegerBitWidth();
if (Ty->isIntegerTy(1))
return Val == 0 || Val == 1 || Val == -1;
if (NumBits >= 64)
return true; // always true, has to fit in largest type
int64_t Min = -(1ll << (NumBits-1));
int64_t Max = (1ll << (NumBits-1)) - 1;
return (Val >= Min && Val <= Max);
}
llvm::ConstantInt *ConstantInt::getMax(llvm::Type *Ty) {
statements;
}
llvm::ConstantInt *ConstantInt::getMax(llvm::LLVMContext &Context) {
statements;
}
bool ConstantFP::isValueValidForType(Type *Ty, const APFloat& Val) {
// convert modifies in place, so make a copy.
APFloat Val2 = APFloat(Val);
MacBook Pro

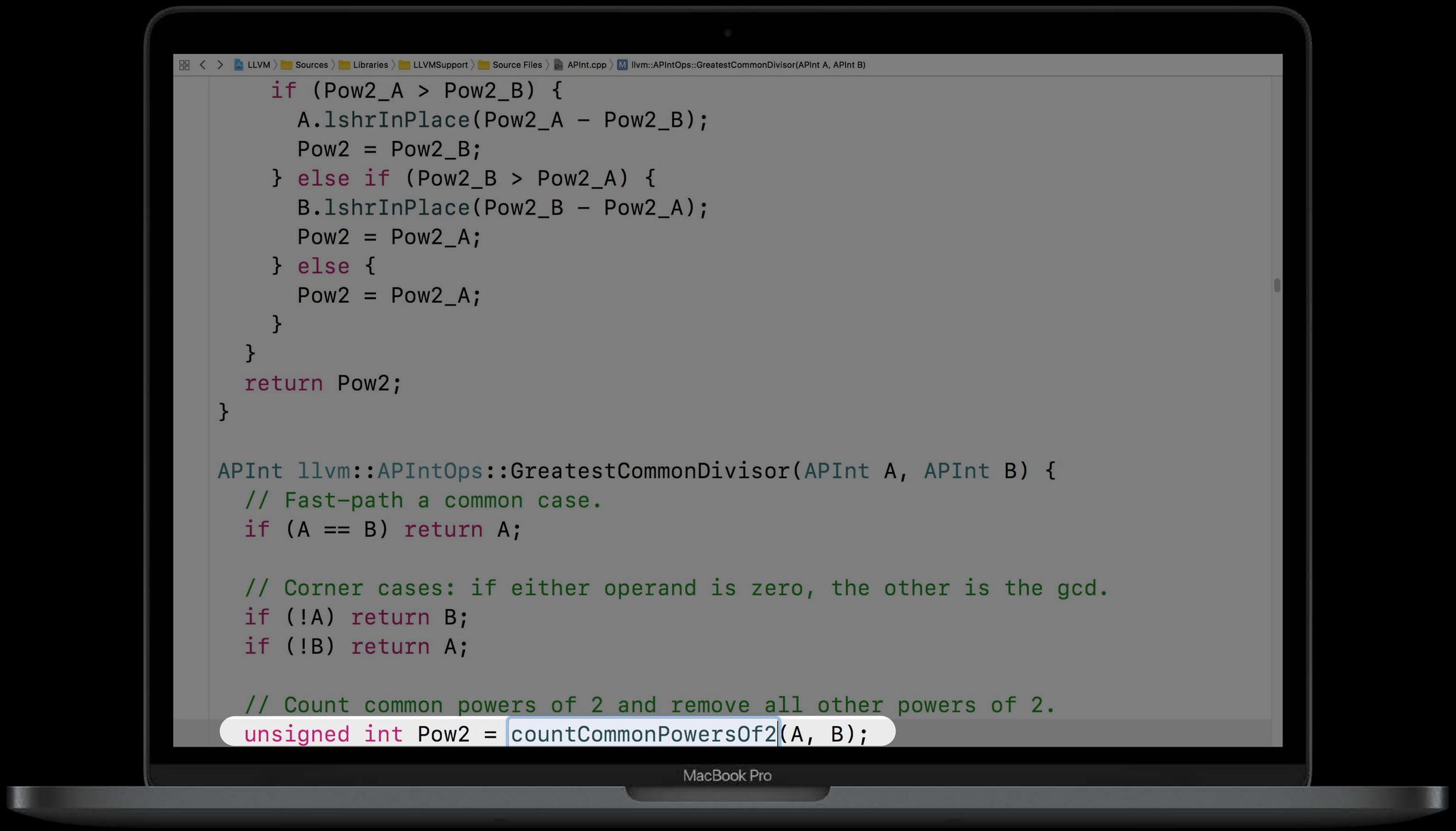

LLVM Sources Libraries LLVMCore Source Files Constants.cpp ConstantInt::getMax(llvm::Type *Ty)
bool ConstantInt::isValueValidForType(Type *Ty, int64_t Val) {
unsigned NumBits = Ty->getIntegerBitWidth();
if (Ty->isIntegerTy(1))
return Val == 0 || Val == 1 || Val == -1;
if (NumBits >= 64)
return true; // always true, has to fit in largest type
int64_t Min = -(1ll << (NumBits-1));
int64_t Max = (1ll << (NumBits-1)) - 1;
return (Val >= Min && Val <= Max);
}
llvm::ConstantInt *ConstantInt::getMax(llvm::Type *Ty) {
statements;
}
llvm::ConstantInt *ConstantInt::getMax(llvm::LLVMContext &Context) {
statements;
}
bool ConstantFP::isValueValidForType(Type *Ty, const APFloat& Val) {
// convert modifies in place, so make a copy.
APFloat Val2 = APFloat(Val);
MacBook Pro

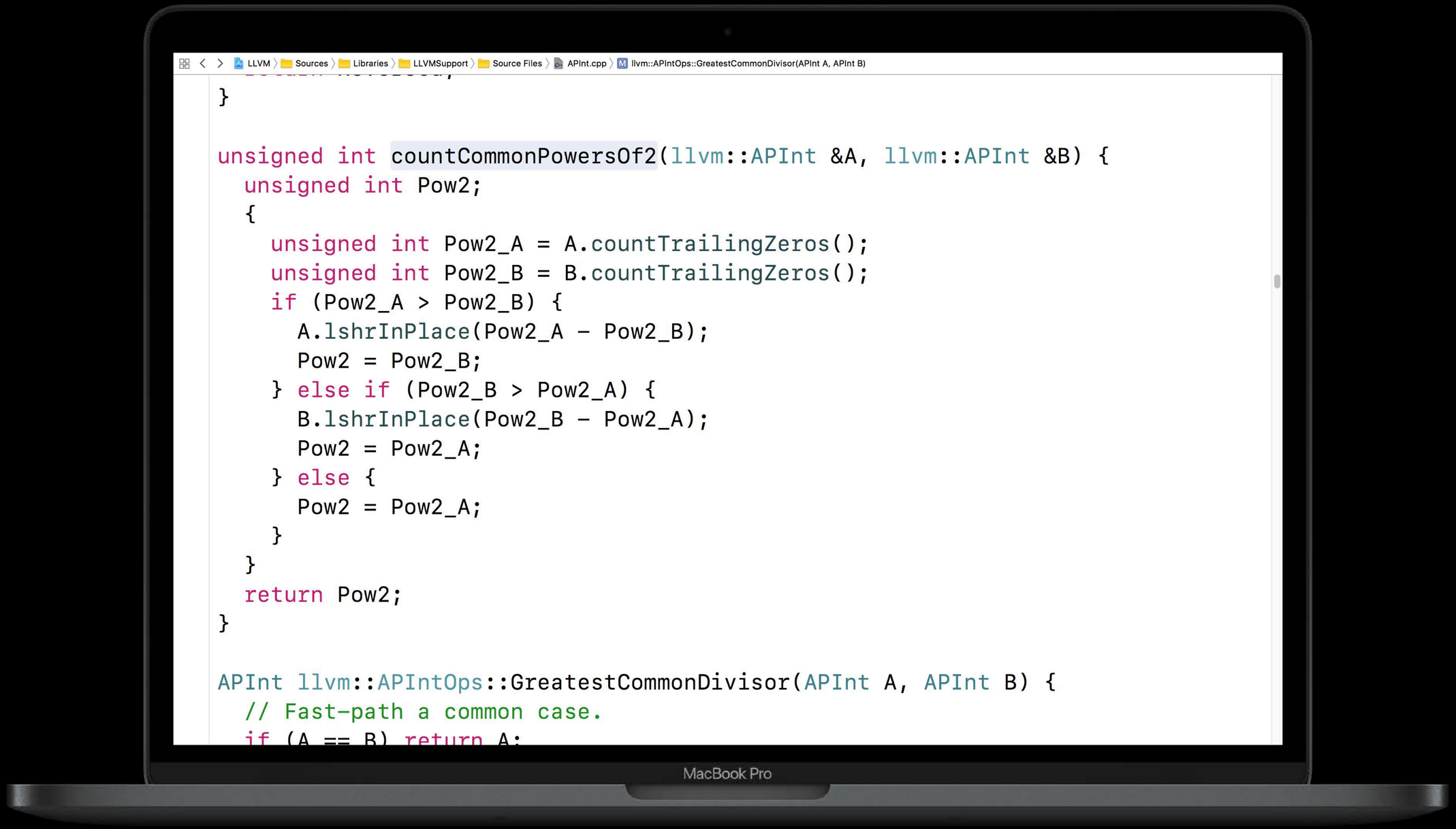

LLVM
Sources
Libraries
LLVMSupport
Source Files
APInt.cpp
llvm::APIntOps::GreatestCommonDivisor(APInt A, APInt B)
APInt llvm::APIntOps::GreatestCommonDivisor(APInt A, APInt B) {
// Fast-path a common case.
if (A == B) return A;
// Corner cases: if either operand is zero, the other is the gcd.
if (!A) return B;
if (!B) return A;
// Count common powers of 2 and remove all other powers of 2.
unsigned int Pow2;
{
unsigned int Pow2_A = A.countTrailingZeros();
unsigned int Pow2_B = B.countTrailingZeros();
if (Pow2_A > Pow2_B) {
A.lshrInPlace(Pow2_A - Pow2_B);
Pow2 = Pow2_B;
} else if (Pow2_B > Pow2_A) {
B.lshrInPlace(Pow2_B - Pow2_A);
Pow2 = Pow2_A;
} else {
Pow2 = Pow2_A;
}
}
MacBook Pro

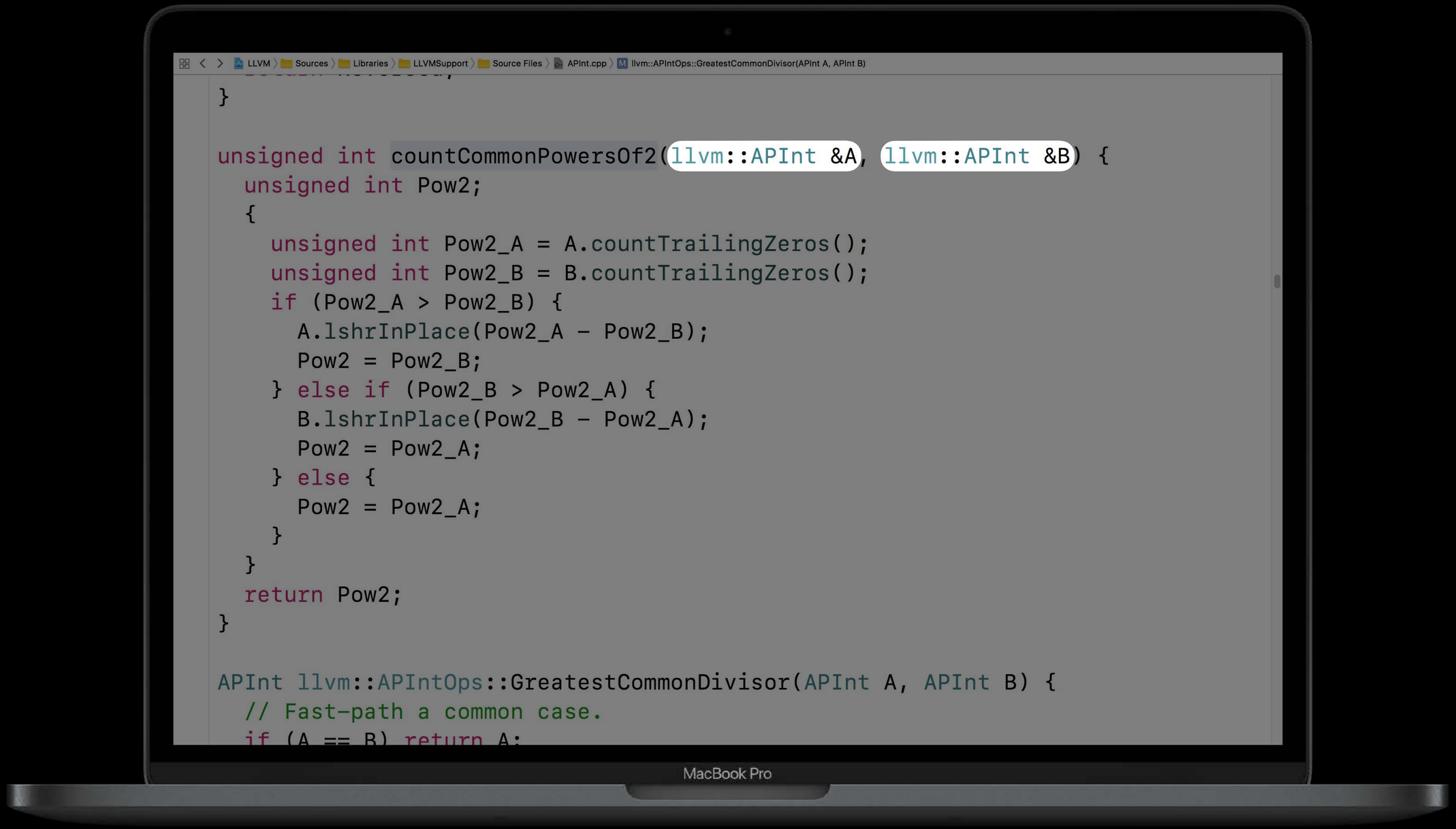

LLVM Sources Libraries LLVMSupport Source Files APInt.cpp llvm::APIntOps::GreatestCommonDivisor(APInt A, APInt B)
APInt llvm::APIntOps::GreatestCommonDivisor(APInt A, APInt B) {
// Fast-path a common case.
if (A == B) return A;
// Corner cases: if either operand is zero, the other is the gcd.
if (!A) return B;
if (!B) return A;
// Count common powers of 2 and remove all other powers of 2.
unsigned int Pow2;
{
unsigned int Pow2_A = A.countTrailingZeros();
unsigned int Pow2_B = B.countTrailingZeros();
if (Pow2_A > Pow2_B) {
A.lshrInPlace(Pow2_A - Pow2_B);
Pow2 = Pow2_B;
} else if (Pow2_B > Pow2_A) {
B.lshrInPlace(Pow2_B - Pow2_A);
Pow2 = Pow2_A;
} else {
Pow2 = Pow2_A;
}
}
MacBook Pro

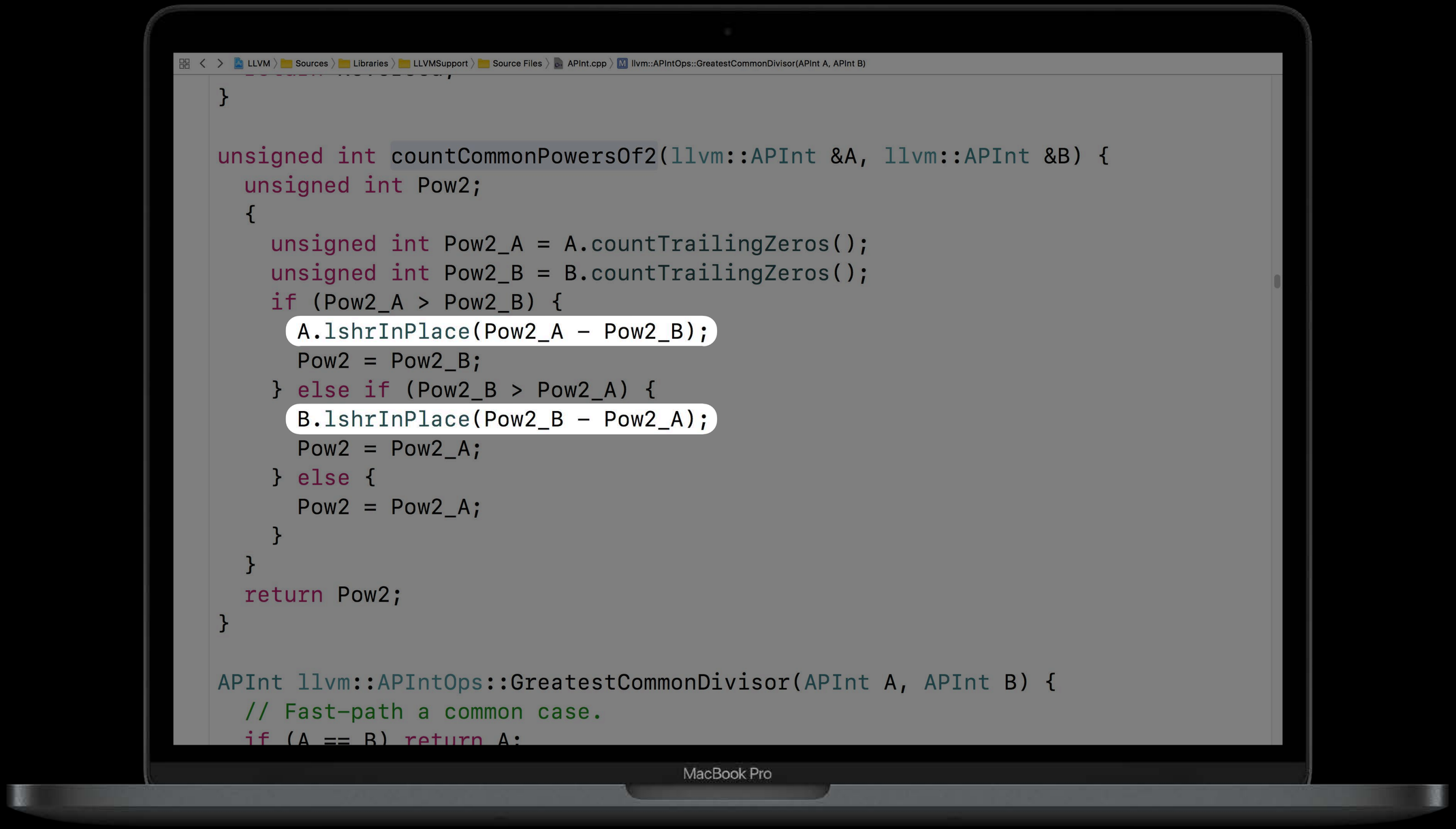

LLVM 〉 Sources 〉 Libraries 〉 LLVMSupport 〉 Source Files 〉 APInt.cpp 〉 llvm::APIntOps::GreatestCommonDivisor(APInt A, APInt B)
APInt llvm::APIntOps::GreatestCommonDivisor(APInt A, APInt B) {
// Fast-path a common case.
if (A == B) return
// Corner cases: i
the other is the gcd.
if (!A) return B;
if (!B) return A;
// Count common pow
other powers of 2.
unsigned int Pow2;
{
unsigned int Pow
os();
unsigned int Pow
os();
if (Pow2_A > Pow
A.lshrInPlace(
Pow2 = Pow2_B;
} else if (Pow2_
B.lshrInPlace(
Pow2 = Pow2_A;
} else {
Pow2 = Pow2_A;
}
}
Cut
Copy Symbol Name
Paste
Rename...
Extract Function
Extract Method
Extract Expression
Find Selected Text in Workspace
Find Selected Symbol in Workspace
Find Call Hierarchy
Show Issue
Jump to Definition
Show Blame for Line
Open in Assistant Editor
Reveal in Project Navigator
Reveal in Symbol Navigator
Show in Finder
Continue to Here
Test
Profile
Services
MacBook Pro

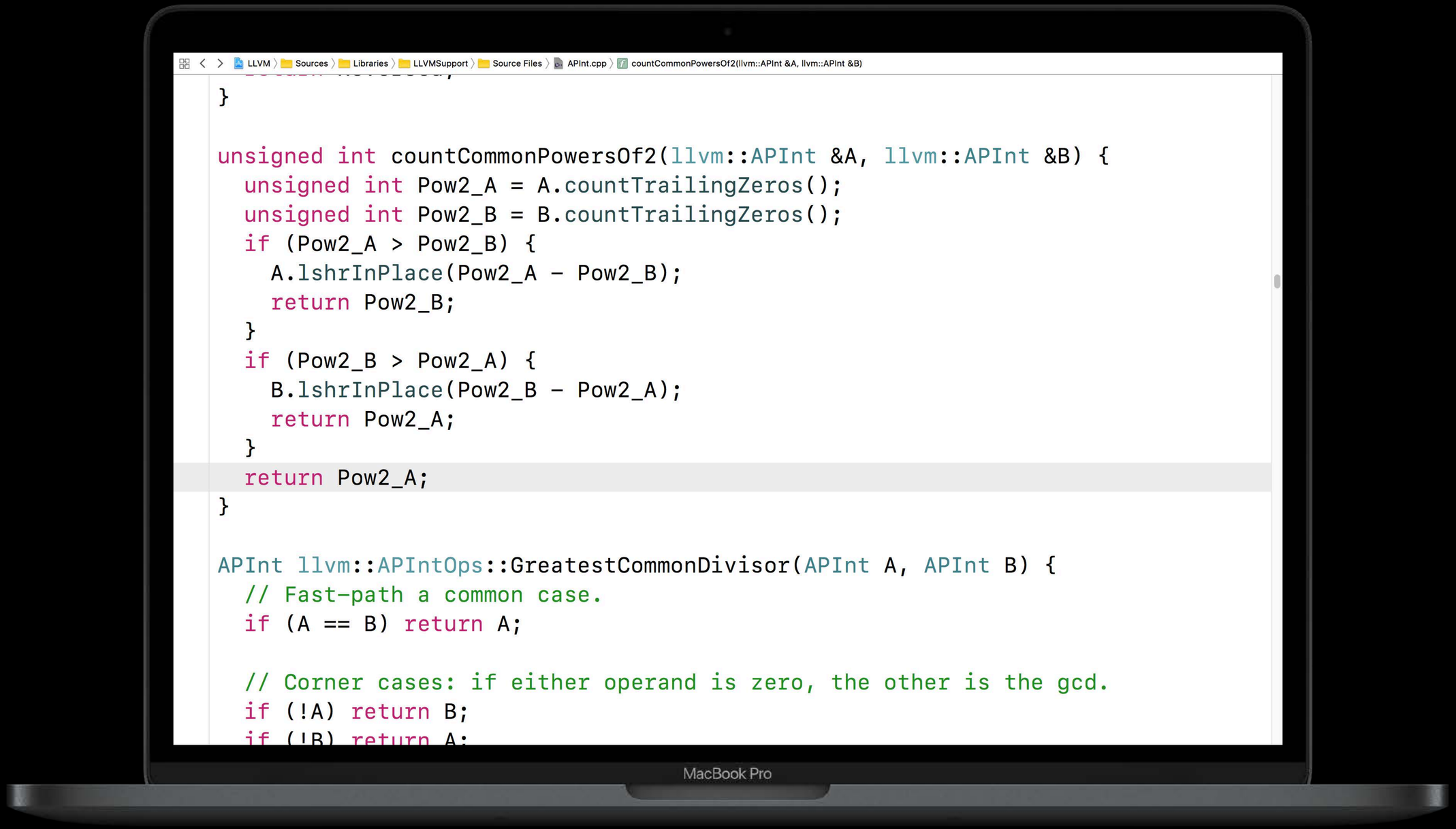

LLVM 〉 Sources 〉 Libraries 〉 LLVMSupport 〉 Source Files 〉 APInt.cpp 〉 llvm::APIntOps::GreatestCommonDivisor(APInt A, APInt B)
if (Pow2_A > Pow2_B) {
A.lshrInPlace(Pow2_A - Pow2_B);
Pow2 = Pow2_B;
} else if (Pow2_B > Pow2_A) {
B.lshrInPlace(Pow2_B - Pow2_A);
Pow2 = Pow2_A;
} else {
Pow2 = Pow2_A;
}
}
return Pow2;
}
APInt llvm::APIntOps::GreatestCommonDivisor(APInt A, APInt B) {
// Fast-path a common case.
if (A == B) return A;
// Corner cases: if either operand is zero, the other is the gcd.
if (!A) return B;
if (!B) return A;
// Count common powers of 2 and remove all other powers of 2.
unsigned int Pow2 = countCommonPowersOf2(A, B);
MacBook Pro

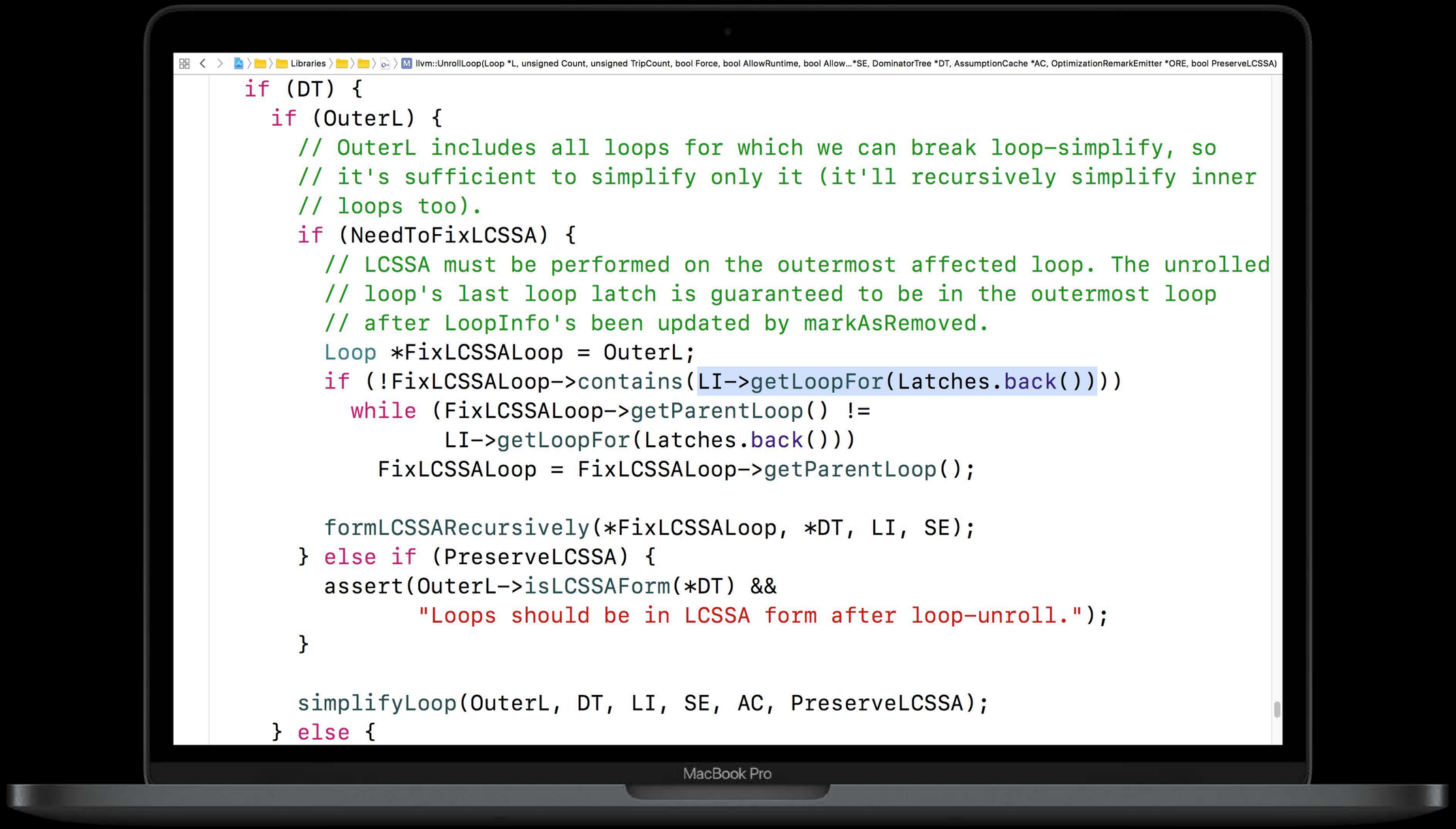

LLVM Sources Libraries LLVMSupport Source Files APInt.cpp llvm::APIntOps::GreatestCommonDivisor(APInt A, APInt B)
if (Pow2_A > Pow2_B) {
A.lshrInPlace(Pow2_A - Pow2_B);
Pow2 = Pow2_B;
} else if (Pow2_B > Pow2_A) {
B.lshrInPlace(Pow2_B - Pow2_A);
Pow2 = Pow2_A;
} else {
Pow2 = Pow2_A;
}
}
return Pow2;
}
APInt llvm::APIntOps::GreatestCommonDivisor(APInt A, APInt B) {
// Fast-path a common case.
if (A == B) return A;
// Corner cases: if either operand is zero, the other is the gcd.
if (!A) return B;
if (!B) return A;
// Count common powers of 2 and remove all other powers of 2.
unsigned int Pow2 = countCommonPowersOf2(A, B);
MacBook Pro

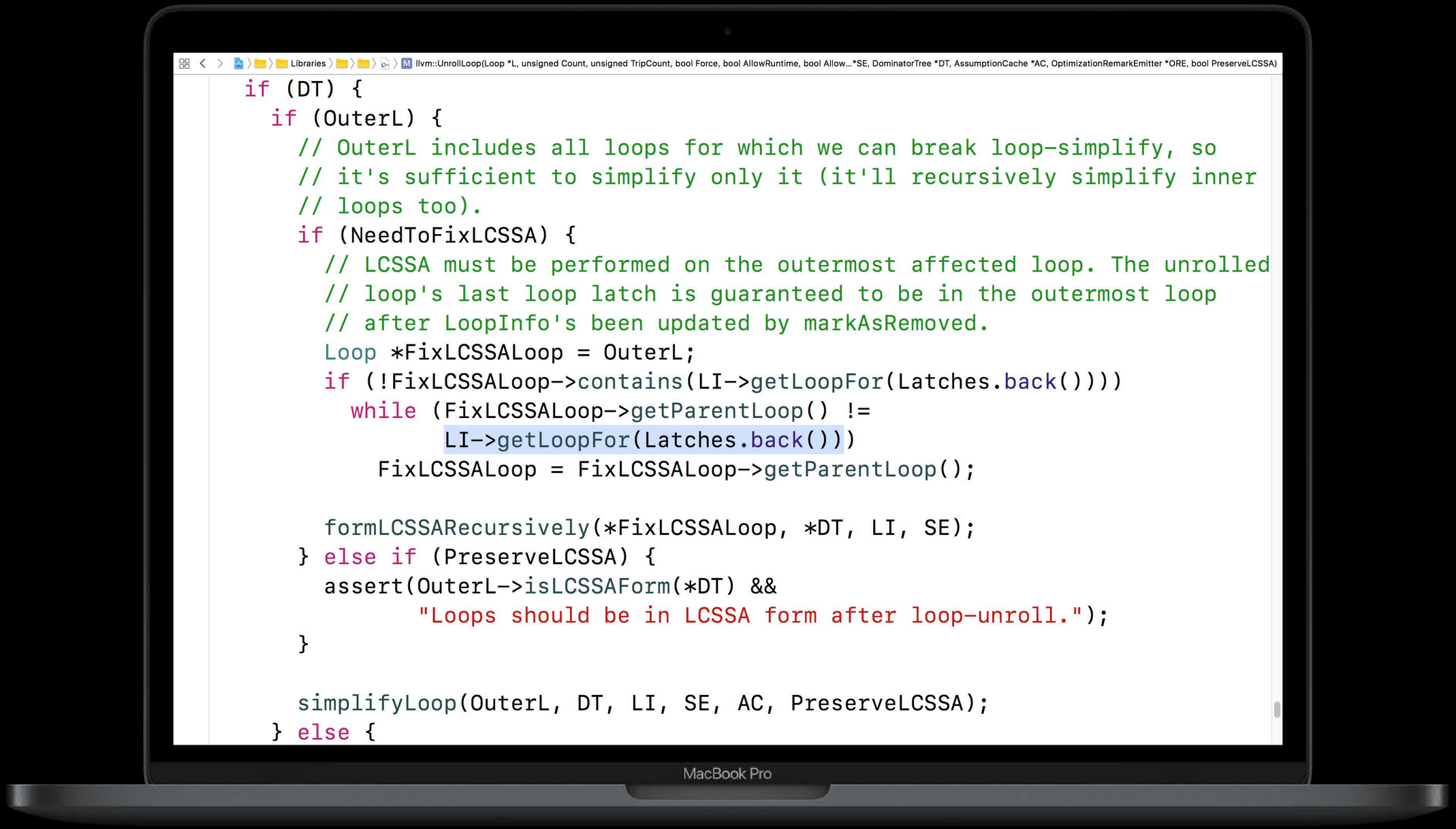

LLVM
Sources
Libraries
LLVMSupport
Source Files
APInt.cpp
llvm::APIntOps::GreatestCommonDivisor(APInt A, APInt B)
}
unsigned int countCommonPowersOf2(llvm::APInt &A, llvm::APInt &B) {
unsigned int Pow2;
{
unsigned int Pow2_A = A.countTrailingZeros();
unsigned int Pow2_B = B.countTrailingZeros();
if (Pow2_A > Pow2_B) {
A.lshrInPlace(Pow2_A - Pow2_B);
Pow2 = Pow2_B;
} else if (Pow2_B > Pow2_A) {
B.lshrInPlace(Pow2_B - Pow2_A);
Pow2 = Pow2_A;
} else {
Pow2 = Pow2_A;
}
}
return Pow2;
}
APInt llvm::APIntOps::GreatestCommonDivisor(APInt A, APInt B) {
// Fast-path a common case.
MacBook Pro

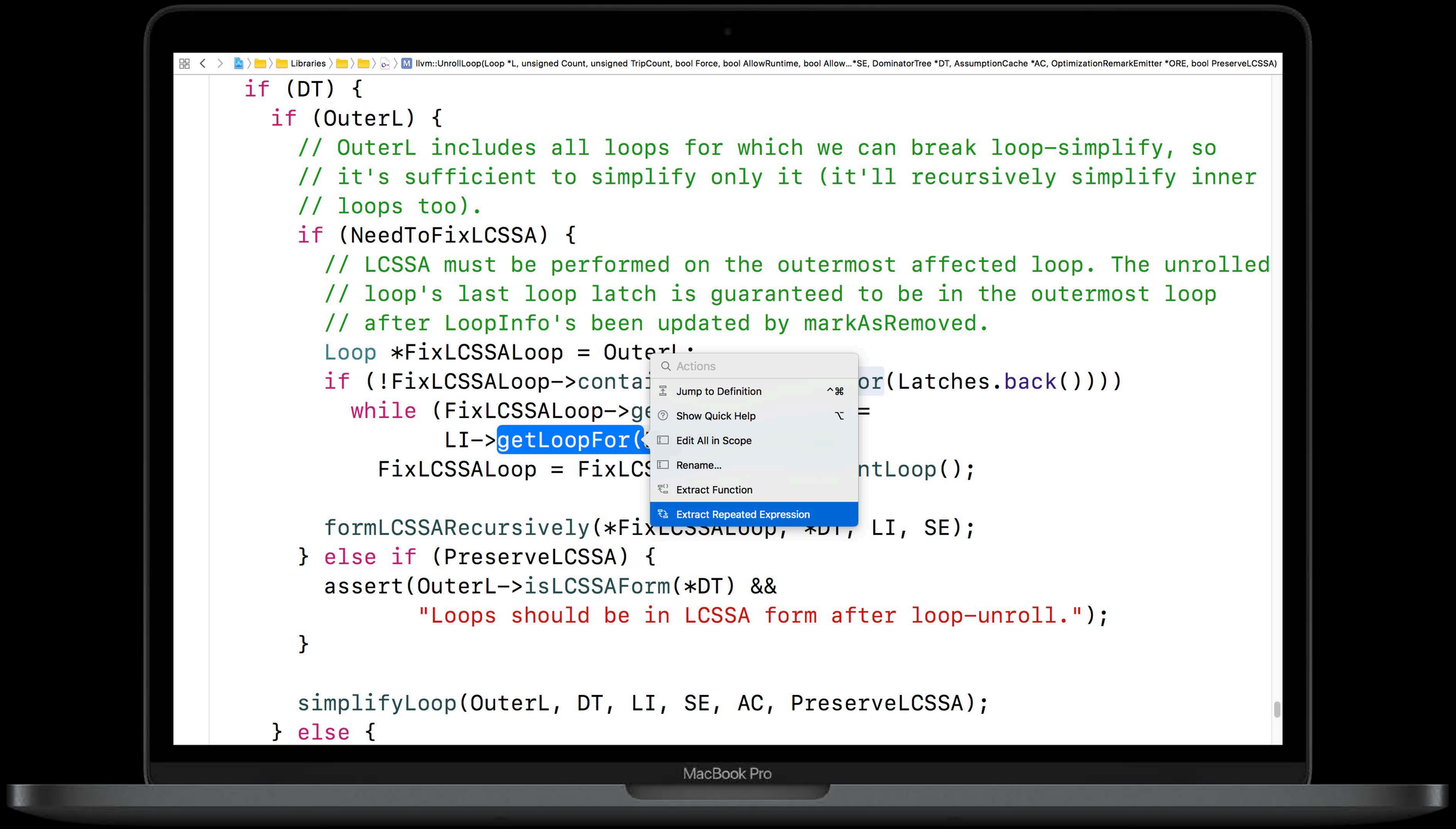

LLVM
Sources
Libraries
LLVMSupport
Source Files
APInt.cpp
llvm::APIntOps::GreatestCommonDivisor(APInt A, APInt B)
}
unsigned int countCommonPowersOf2(llvm::APInt &A, llvm::APInt &B) {
unsigned int Pow2;
{
unsigned int Pow2_A = A.countTrailingZeros();
unsigned int Pow2_B = B.countTrailingZeros();
if (Pow2_A > Pow2_B) {
A.lshrInPlace(Pow2_A - Pow2_B);
Pow2 = Pow2_B;
} else if (Pow2_B > Pow2_A) {
B.lshrInPlace(Pow2_B - Pow2_A);
Pow2 = Pow2_A;
} else {
Pow2 = Pow2_A;
}
}
return Pow2;
}
APInt llvm::APIntOps::GreatestCommonDivisor(APInt A, APInt B) {
// Fast-path a common case.
MacBook Pro

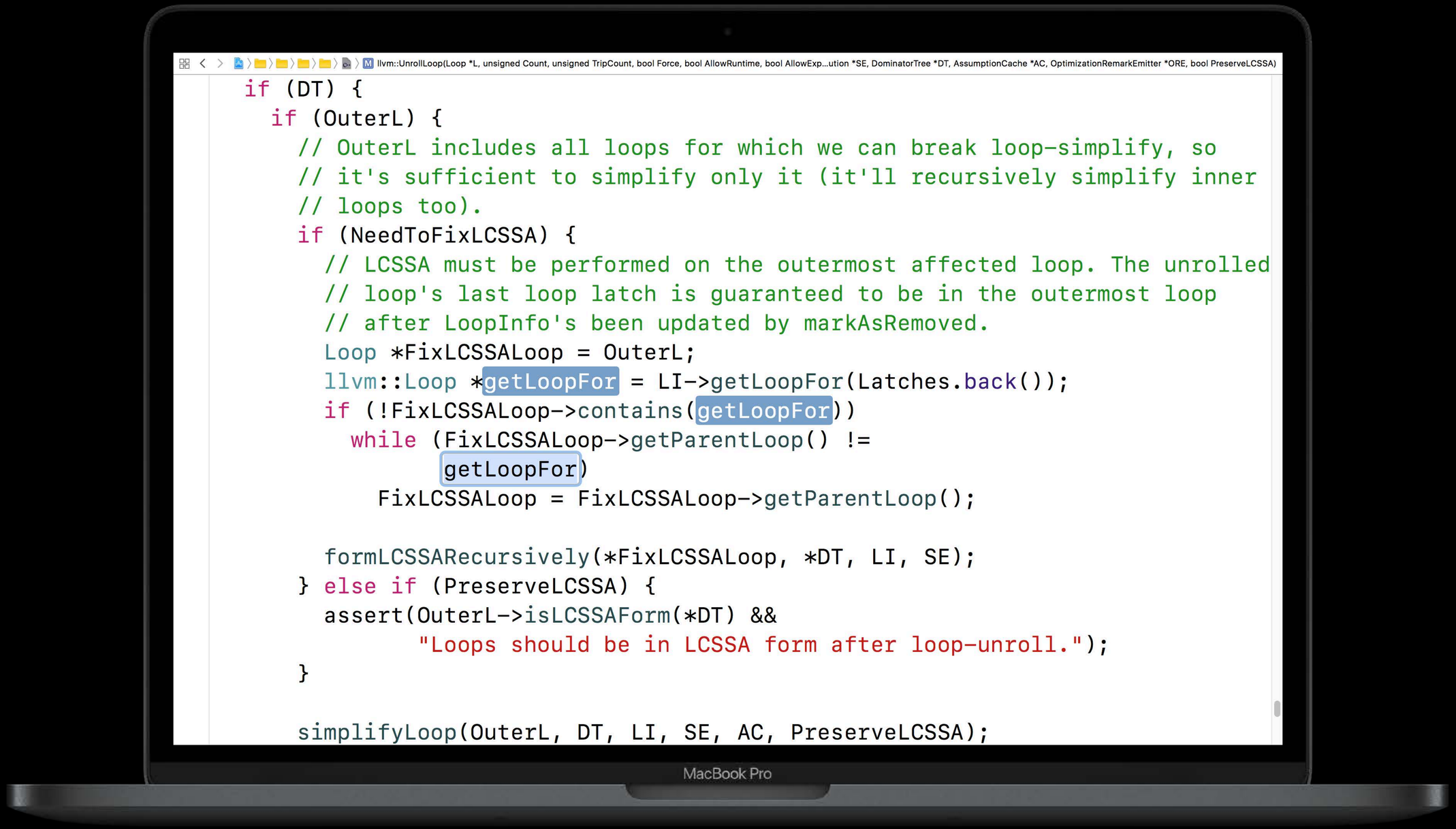

LLVM Sources Libraries LLVMSupport Source Files APInt.cpp llvm::APIntOps::GreatestCommonDivisor(APInt A, APInt B)
}
unsigned int countCommonPowersOf2(llvm::APInt &A, llvm::APInt &B) {
unsigned int Pow2;
{
unsigned int Pow2_A = A.countTrailingZeros();
unsigned int Pow2_B = B.countTrailingZeros();
if (Pow2_A > Pow2_B) {
A.lshrInPlace(Pow2_A - Pow2_B);
Pow2 = Pow2_B;
} else if (Pow2_B > Pow2_A) {
B.lshrInPlace(Pow2_B - Pow2_A);
Pow2 = Pow2_A;
} else {
Pow2 = Pow2_A;
}
}
return Pow2;
}
APInt llvm::APIntOps::GreatestCommonDivisor(APInt A, APInt B) {
// Fast-path a common case.
MacBook Pro

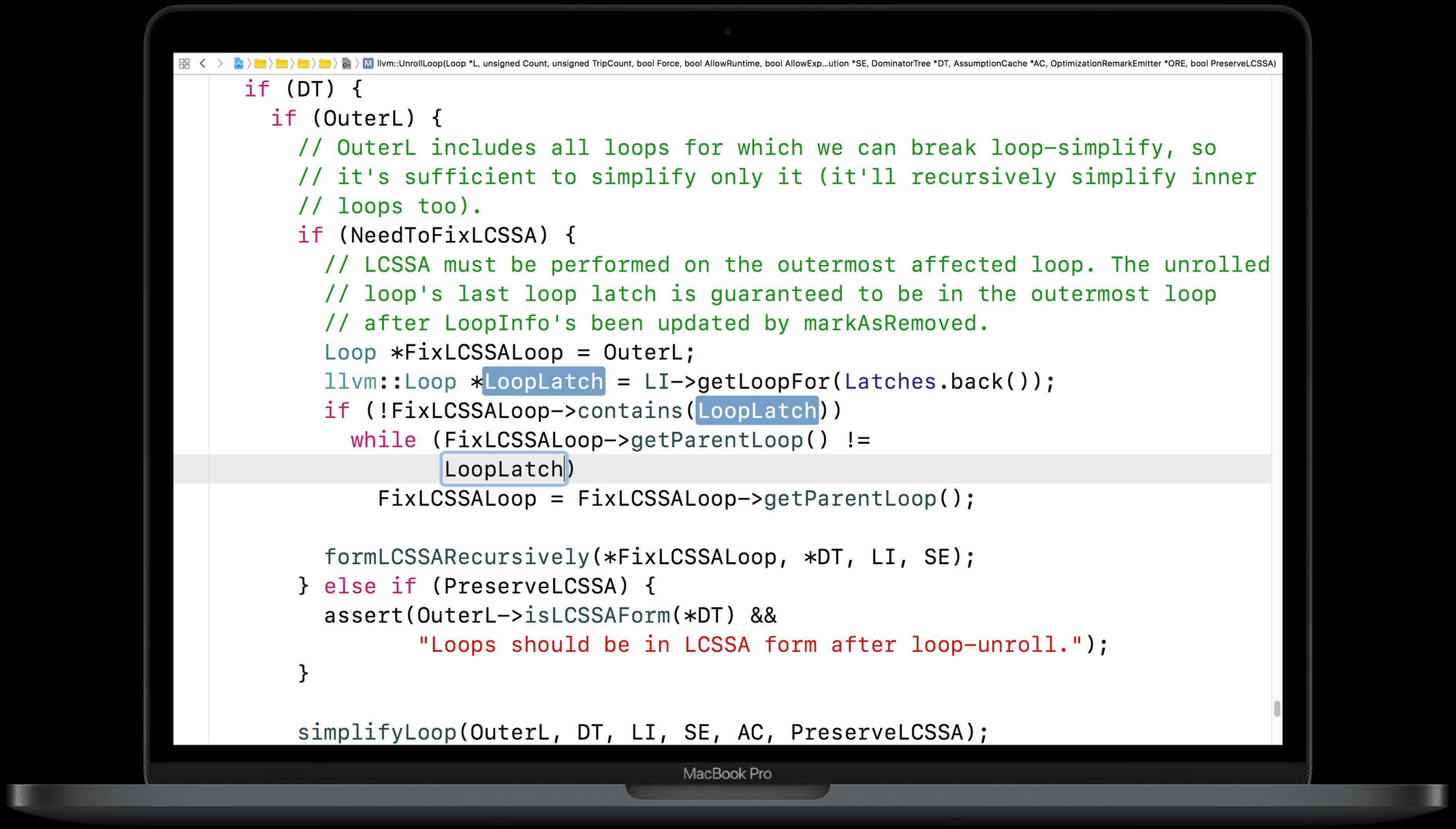

LLVM 〉 Sources 〉 Libraries 〉 LLVMSupport 〉 Source Files 〉 APInt.cpp 〉 countCommonPowersOf2(llvm::APInt &A, llvm::APInt &B)
}
unsigned int countCommonPowersOf2(llvm::APInt &A, llvm::APInt &B) {
unsigned int Pow2_A = A.countTrailingZeros();
unsigned int Pow2_B = B.countTrailingZeros();
if (Pow2_A > Pow2_B) {
A.lshrInPlace(Pow2_A - Pow2_B);
return Pow2_B;
}
if (Pow2_B > Pow2_A) {
B.lshrInPlace(Pow2_B - Pow2_A);
return Pow2_A;
}
return Pow2_A;
}
APInt llvm::APIntOps::GreatestCommonDivisor(APInt A, APInt B) {
// Fast-path a common case.
if (A == B) return A;
// Corner cases: if either operand is zero, the other is the gcd.
if (!A) return B;
MacBook Pro

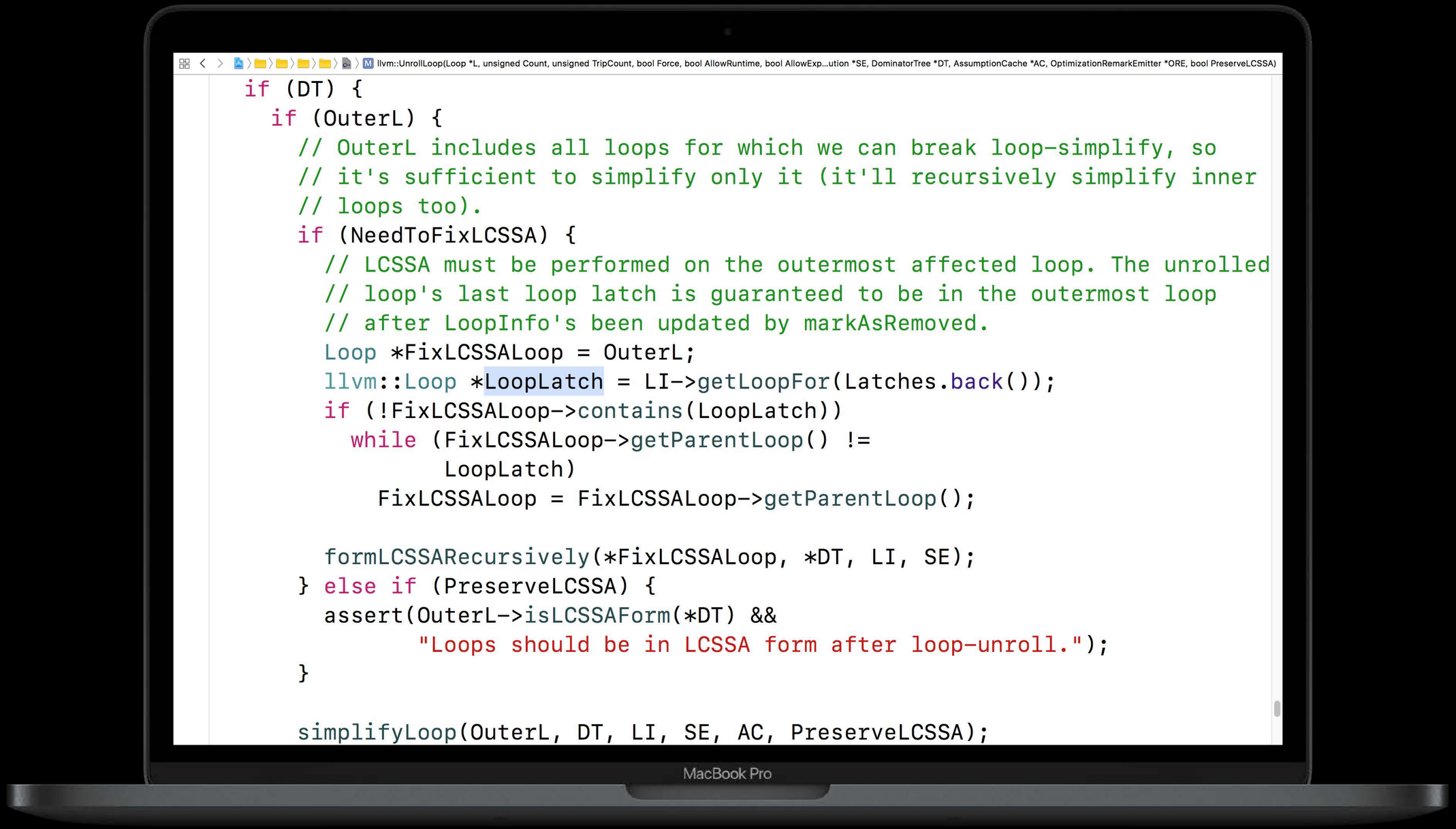

Libraries
llvm::UnrollLoop(Loop *L, unsigned Count, unsigned TripCount, bool Force, bool AllowRuntime, bool Allow...*SE, DominatorTree *DT, AssumptionCache *AC, OptimizationRemarkEmitter *ORE, bool PreserveLCSSA)
if (DT) {
if (OuterL) {
// OuterL includes all loops for which we can break loop-simplify, so
// it's sufficient to simplify only it (it'll recursively simplify inner
// loops too).
if (NeedToFixLCSSA) {
// LCSSA must be performed on the outermost affected loop. The unrolled
// loop's last loop latch is guaranteed to be in the outermost loop
// after LoopInfo's been updated by markAsRemoved.
Loop *FixLCSSALoop = OuterL;
if (!FixLCSSALoop->contains(LI->getLoopFor(Latches.back())))
while (FixLCSSALoop->getParentLoop() !=
LI->getLoopFor(Latches.back()))
FixLCSSALoop = FixLCSSALoop->getParentLoop();
formLCSSARecursively(*FixLCSSALoop, *DT, LI, SE);
} else if (PreserveLCSSA) {
assert(OuterL->isLCSSAForm(*DT) &&
"Loops should be in LCSSA form after loop-unroll.");
}
simplifyLoop(OuterL, DT, LI, SE, AC, PreserveLCSSA);
} else {
MacBook Pro

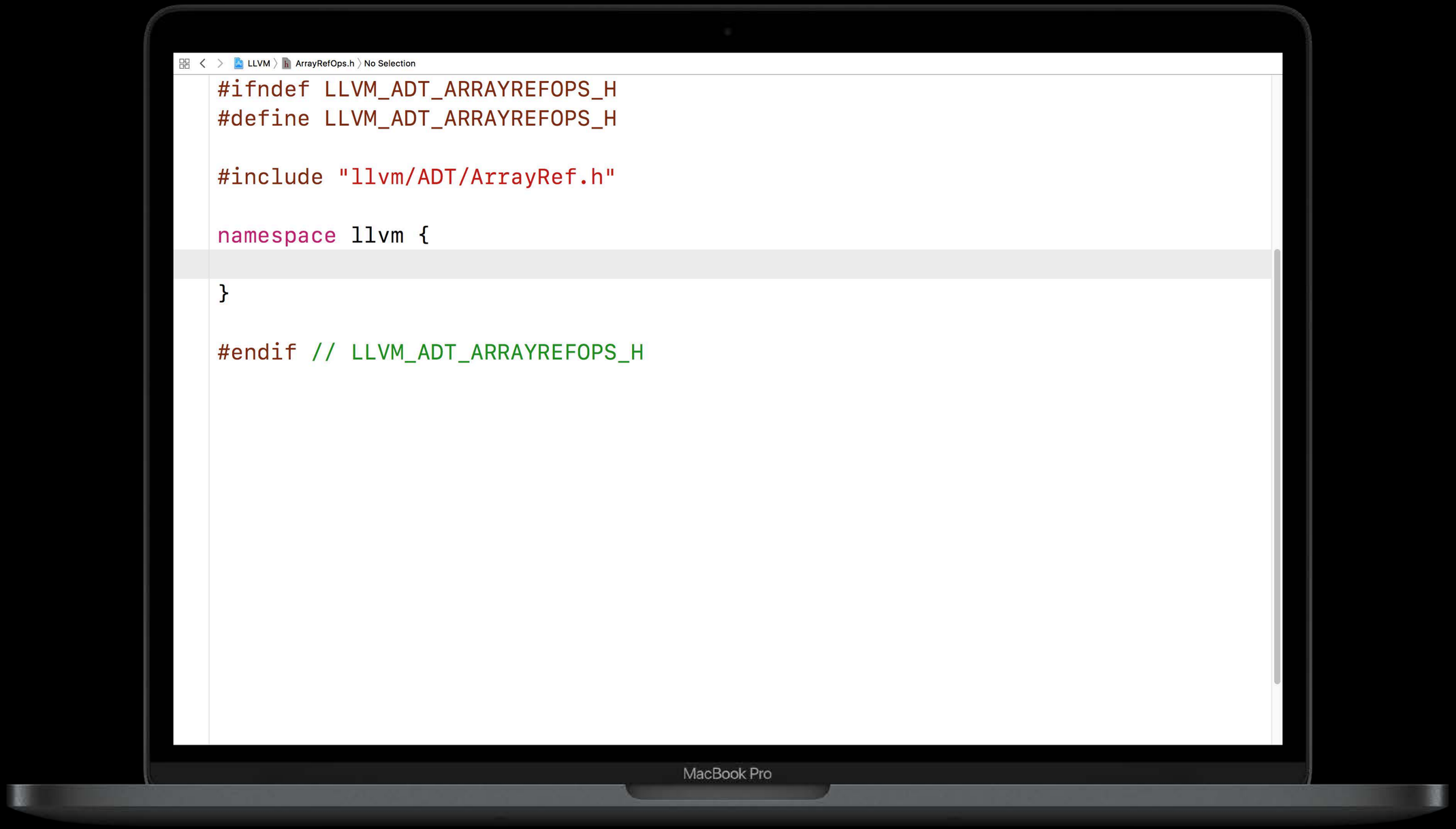

Libraries
llvm::UnrollLoop(Loop *L, unsigned Count, unsigned TripCount, bool Force, bool AllowRuntime, bool Allow...*SE, DominatorTree *DT, AssumptionCache *AC, OptimizationRemarkEmitter *ORE, bool PreserveLCSSA)
if (DT) {
if (OuterL) {
// OuterL includes all loops for which we can break loop-simplify, so
// it's sufficient to simplify only it (it'll recursively simplify inner
// loops too).
if (NeedToFixLCSSA) {
// LCSSA must be performed on the outermost affected loop. The unrolled
// loop's last loop latch is guaranteed to be in the outermost loop
// after LoopInfo's been updated by markAsRemoved.
Loop *FixLCSSALoop = OuterL;
if (!FixLCSSALoop->contains(LI->getLoopFor(Latches.back())))
while (FixLCSSALoop->getParentLoop() !=
LI->getLoopFor(Latches.back()))
FixLCSSALoop = FixLCSSALoop->getParentLoop();
formLCSSARecursively(*FixLCSSALoop, *DT, LI, SE);
} else if (PreserveLCSSA) {
assert(OuterL->isLCSSAForm(*DT) &&
"Loops should be in LCSSA form after loop-unroll.");
}
simplifyLoop(OuterL, DT, LI, SE, AC, PreserveLCSSA);
} else {
MacBook Pro

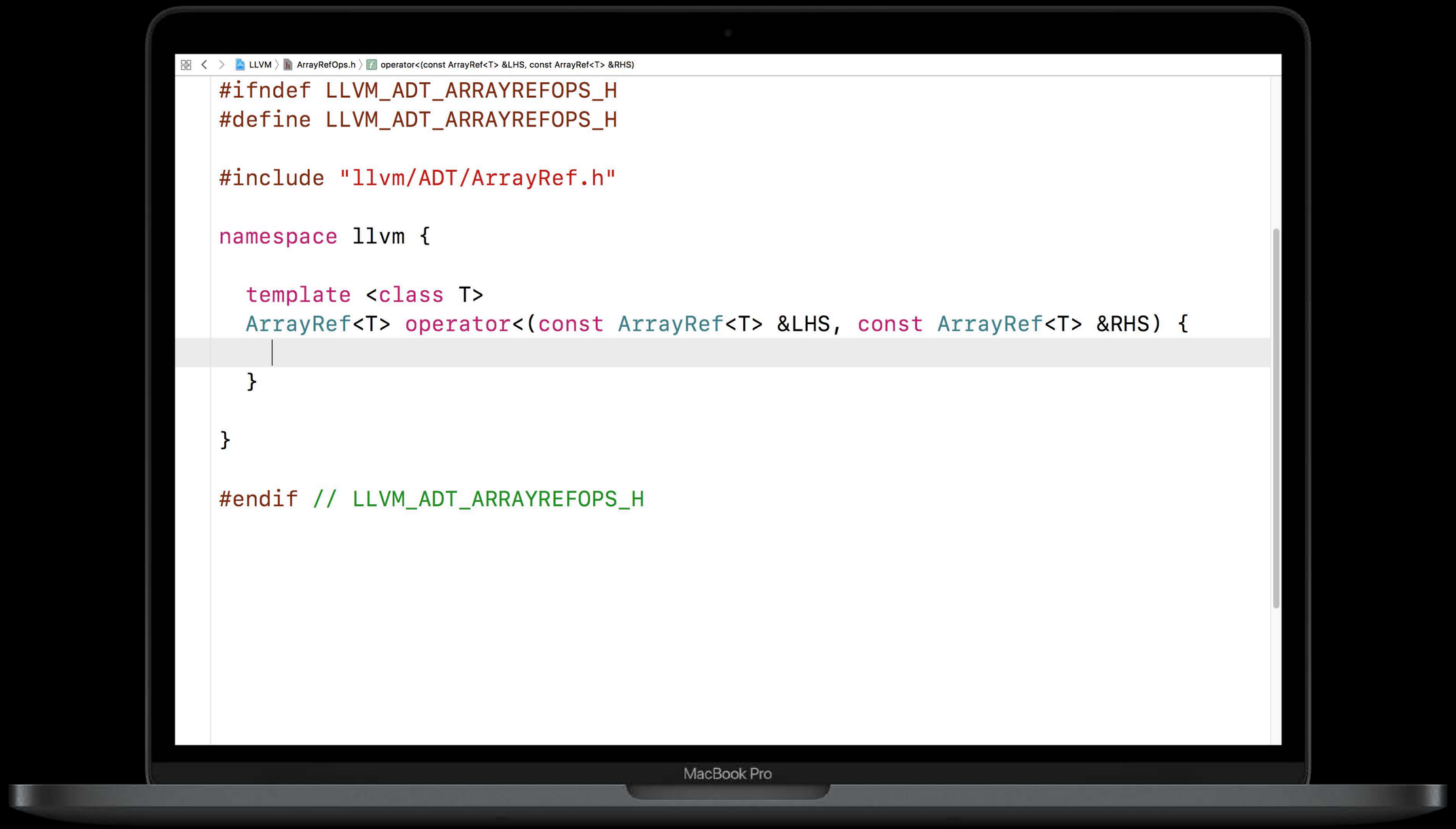

Libraries
llvm::UnrollLoop(Loop *L, unsigned Count, unsigned TripCount, bool Force, bool AllowRuntime, bool Allow...*SE, DominatorTree *DT, AssumptionCache *AC, OptimizationRemarkEmitter *ORE, bool PreserveLCSSA)
if (DT) {
if (OuterL) {
// OuterL includes all loops for which we can break loop-simplify, so
// it's sufficient to simplify only it (it'll recursively simplify inner
// loops too).
if (NeedToFixLCSSA) {
// LCSSA must be performed on the outermost affected loop. The unrolled
// loop's last loop latch is guaranteed to be in the outermost loop
// after LoopInfo's been updated by markAsRemoved.
Loop *FixLCSSALoop = Outer
if (!FixLCSSALoop->contai
or(Latches.back()))
while (FixLCSSALoop->ge
=
LI->getLoopFor(
FixLCSSALoop = FixLCS
ntLoop();
formLCSSARecursively(*FixLCSSALoop, *DT, LI, SE);
} else if (PreserveLCSSA) {
assert(OuterL->isLCSSAForm(*DT) &&
"Loops should be in LCSSA form after loop-unroll.");
}
simplifyLoop(OuterL, DT, LI, SE, AC, PreserveLCSSA);
} else {
Actions
Jump to Definition ^⌘
Show Quick Help ⌥
Edit All in Scope
Rename...
Extract Function
Extract Repeated Expression
MacBook Pro

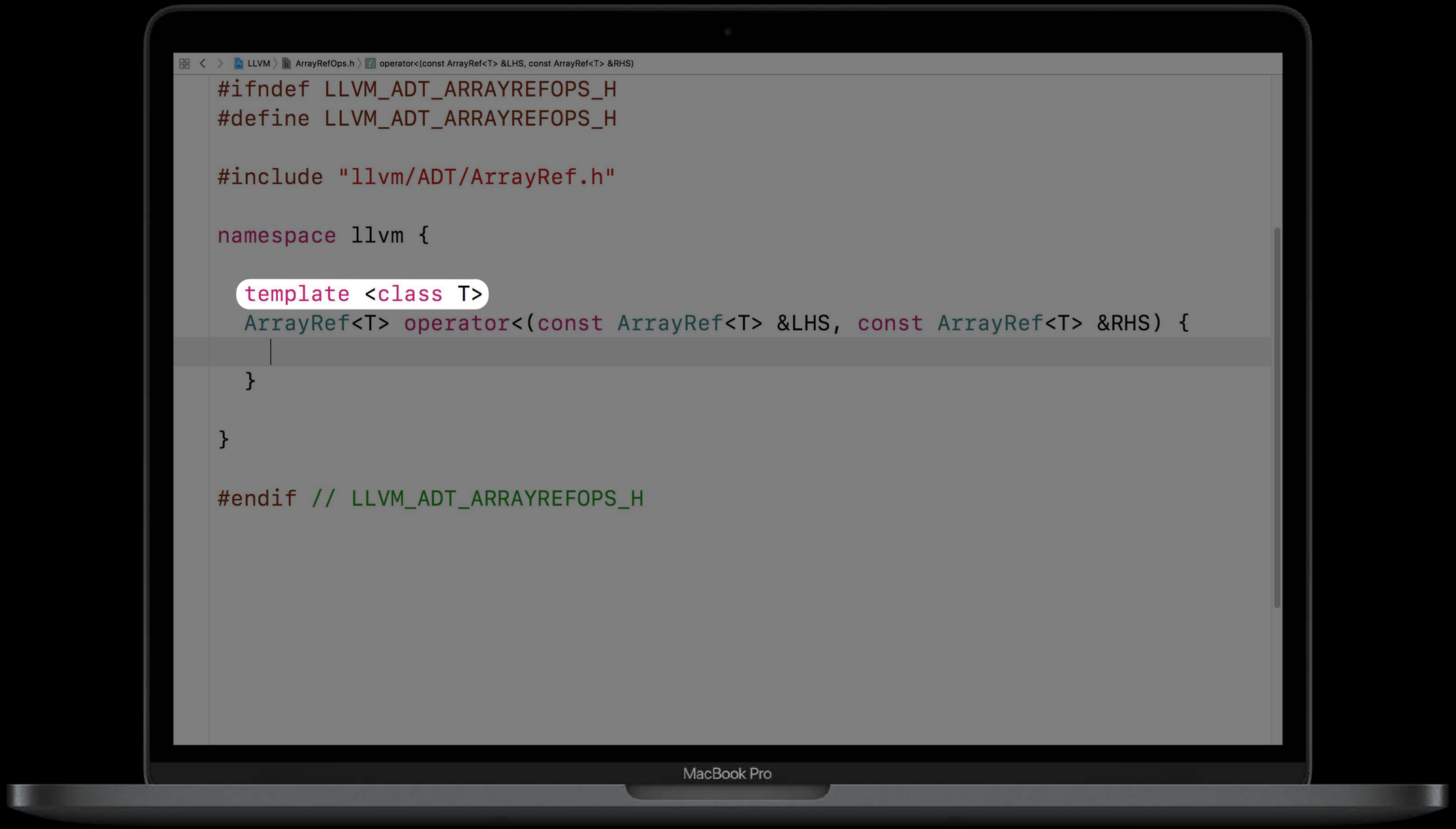

llvm::UnrollLoop(Loop *L, unsigned Count, unsigned TripCount, bool Force, bool AllowRuntime, bool AllowExp...ution *SE, DominatorTree *DT, AssumptionCache *AC, OptimizationRemarkEmitter *ORE, bool PreserveLCSSA)
if (DT) {
if (OuterL) {
// OuterL includes all loops for which we can break loop-simplify, so
// it's sufficient to simplify only it (it'll recursively simplify inner
// loops too).
if (NeedToFixLCSSA) {
// LCSSA must be performed on the outermost affected loop. The unrolled
// loop's last loop latch is guaranteed to be in the outermost loop
// after LoopInfo's been updated by markAsRemoved.
Loop *FixLCSSALoop = OuterL;
llvm::Loop *getLoopFor = LI->getLoopFor(Latches.back());
if (!FixLCSSALoop->contains(getLoopFor))
while (FixLCSSALoop->getParentLoop() !=
getLoopFor)
FixLCSSALoop = FixLCSSALoop->getParentLoop();
formLCSSARecursively(*FixLCSSALoop, *DT, LI, SE);
} else if (PreserveLCSSA) {
assert(OuterL->isLCSSAForm(*DT) &&
"Loops should be in LCSSA form after loop-unroll.");
}
simplifyLoop(OuterL, DT, LI, SE, AC, PreserveLCSSA);
MacBook Pro

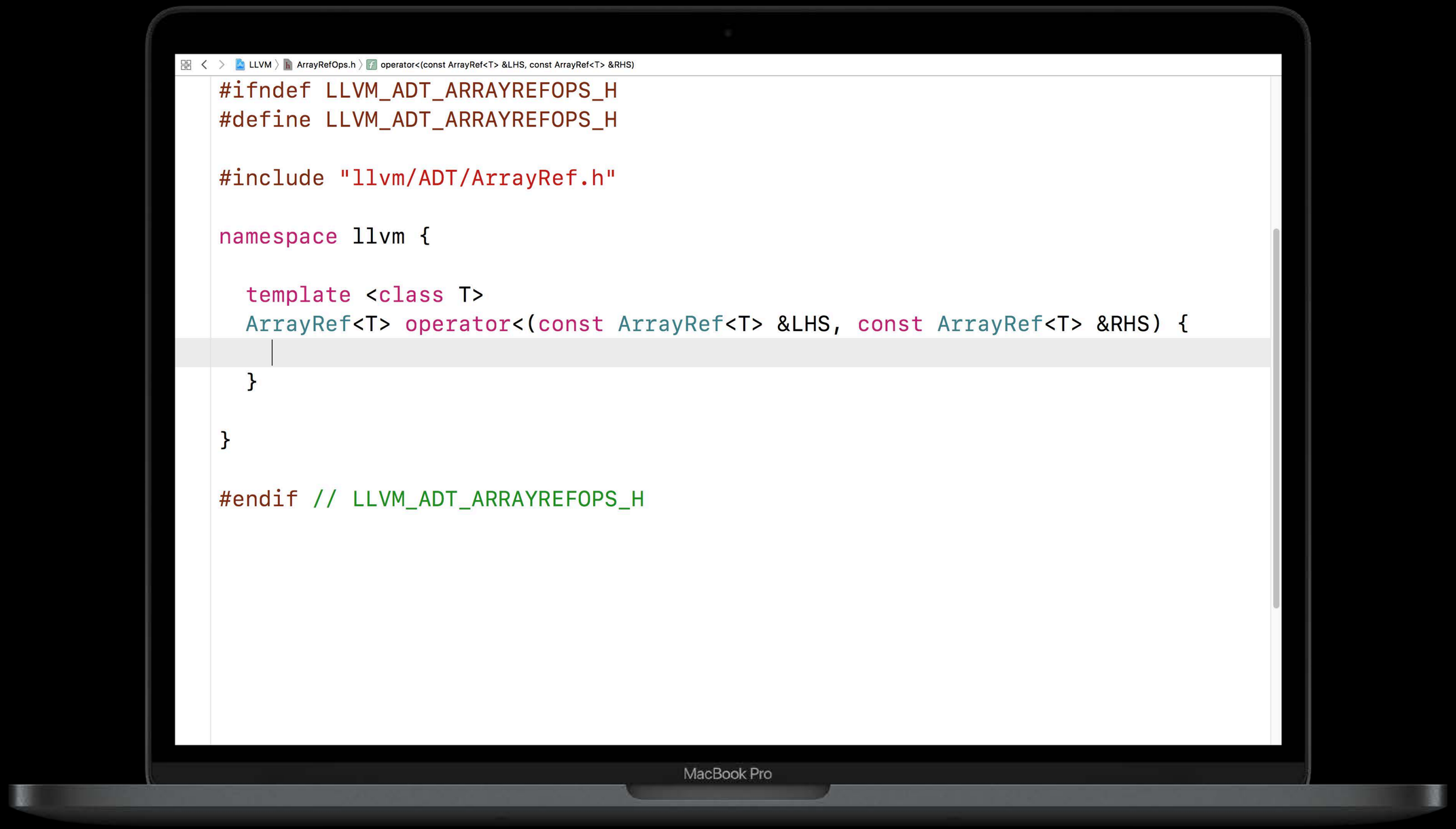

llvm::UnrollLoop(Loop *L, unsigned Count, unsigned TripCount, bool Force, bool AllowRuntime, bool AllowExp...ution *SE, DominatorTree *DT, AssumptionCache *AC, OptimizationRemarkEmitter *ORE, bool PreserveLCSSA)
if (DT) {
if (OuterL) {
// OuterL includes all loops for which we can break loop-simplify, so
// it's sufficient to simplify only it (it'll recursively simplify inner
// loops too).
if (NeedToFixLCSSA) {
// LCSSA must be performed on the outermost affected loop. The unrolled
// loop's last loop latch is guaranteed to be in the outermost loop
// after LoopInfo's been updated by markAsRemoved.
Loop *FixLCSSALoop = OuterL;
llvm::Loop *LoopLatch = LI->getLoopFor(Latches.back());
if (!FixLCSSALoop->contains(LoopLatch))
while (FixLCSSALoop->getParentLoop() !=
LoopLatch)
FixLCSSALoop = FixLCSSALoop->getParentLoop();
formLCSSARecursively(*FixLCSSALoop, *DT, LI, SE);
} else if (PreserveLCSSA) {
assert(OuterL->isLCSSAForm(*DT) &&
"Loops should be in LCSSA form after loop-unroll.");
}
simplifyLoop(OuterL, DT, LI, SE, AC, PreserveLCSSA);
MacBook Pro

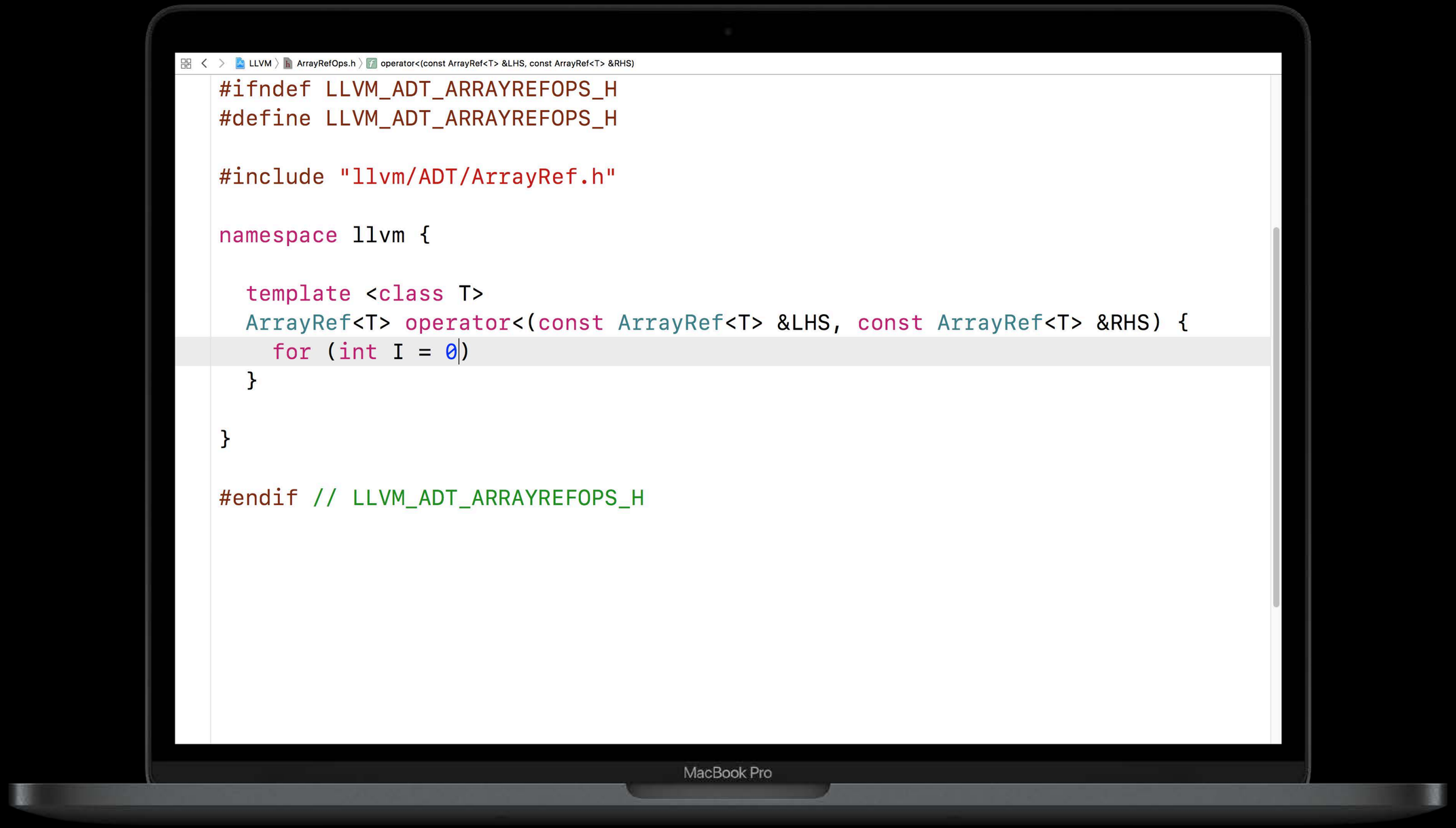

llvm::UnrollLoop(Loop *L, unsigned Count, unsigned TripCount, bool Force, bool AllowRuntime, bool AllowExp...ution *SE, DominatorTree *DT, AssumptionCache *AC, OptimizationRemarkEmitter *ORE, bool PreserveLCSSA)
if (DT) {
if (OuterL) {
// OuterL includes all loops for which we can break loop-simplify, so
// it's sufficient to simplify only it (it'll recursively simplify inner
// loops too).
if (NeedToFixLCSSA) {
// LCSSA must be performed on the outermost affected loop. The unrolled
// loop's last loop latch is guaranteed to be in the outermost loop
// after LoopInfo's been updated by markAsRemoved.
Loop *FixLCSSALoop = OuterL;
llvm::Loop *LoopLatch = LI->getLoopFor(Latches.back());
if (!FixLCSSALoop->contains(LoopLatch))
while (FixLCSSALoop->getParentLoop() !=
LoopLatch)
FixLCSSALoop = FixLCSSALoop->getParentLoop();
formLCSSARecursively(*FixLCSSALoop, *DT, LI, SE);
} else if (PreserveLCSSA) {
assert(OuterL->isLCSSAForm(*DT) &&
"Loops should be in LCSSA form after loop-unroll.");
}
simplifyLoop(OuterL, DT, LI, SE, AC, PreserveLCSSA);
MacBook Pro

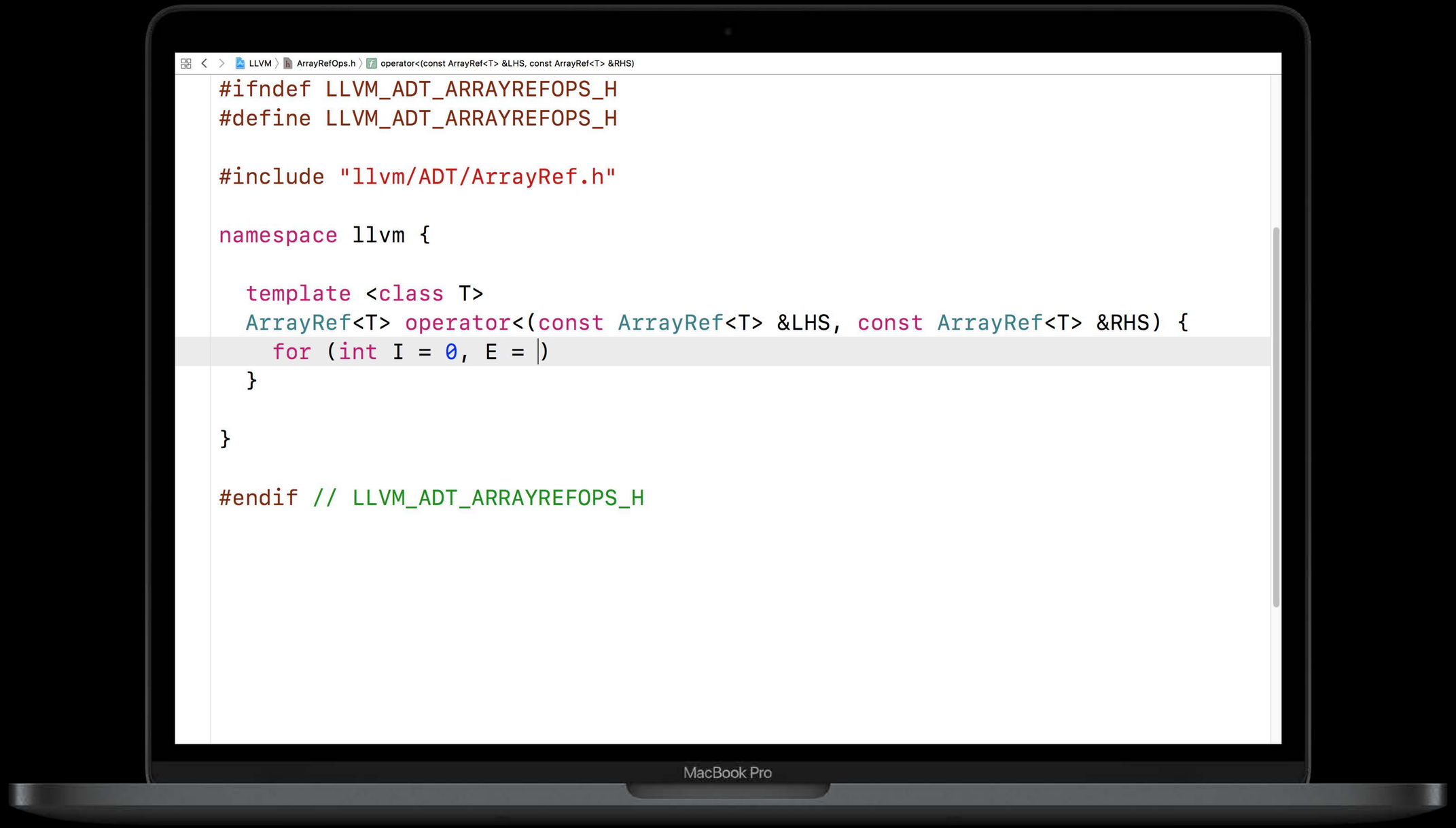

LLVM
ArrayRefOps.h
No Selection
#ifndef LLVM_ADT_ARRAYREFOPS_H
#define LLVM_ADT_ARRAYREFOPS_H
#include "llvm/ADT/ArrayRef.h"
namespace llvm {
}
#endif // LLVM_ADT_ARRAYREFOPS_H
MacBook Pro

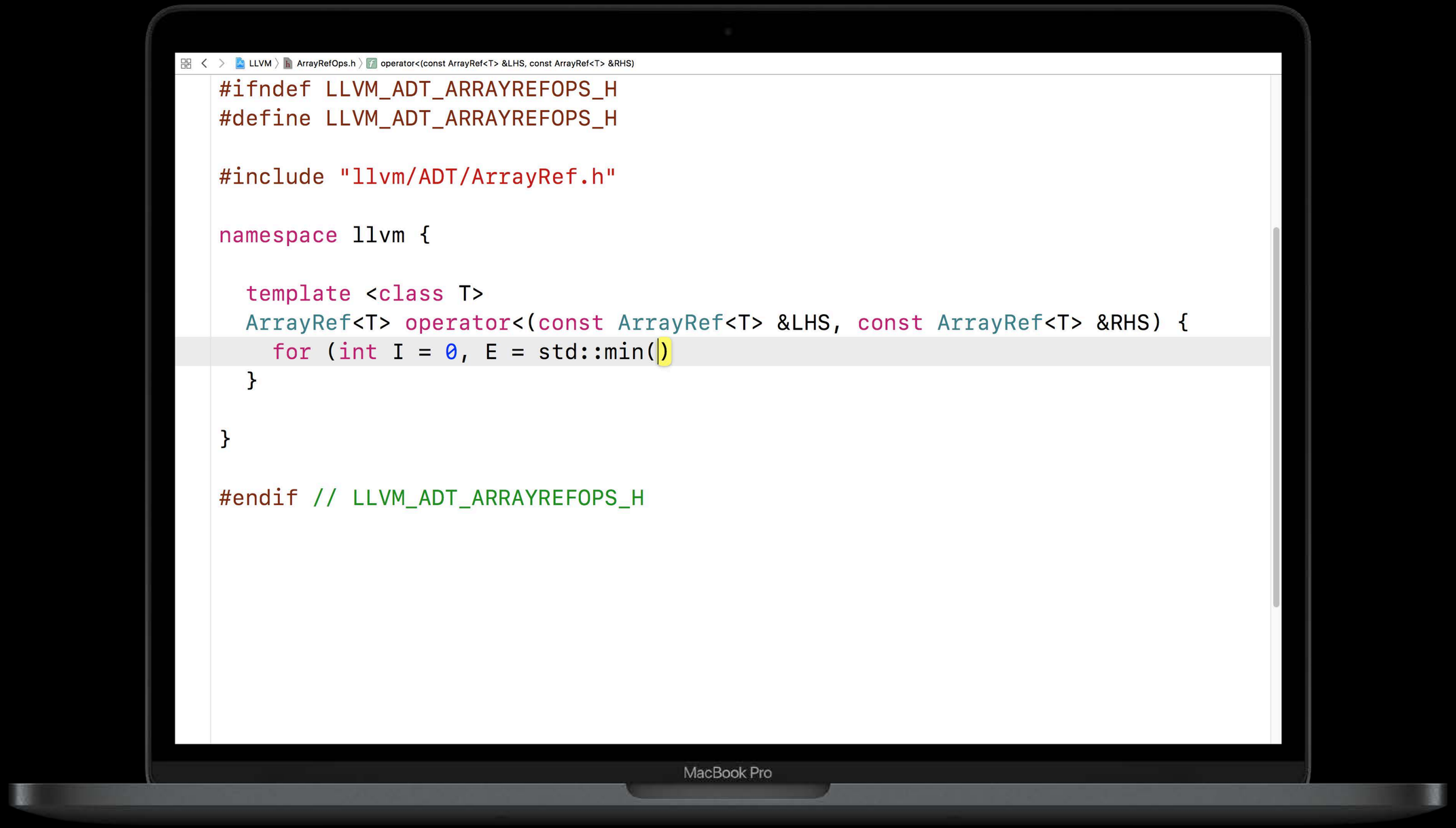

LLVM
ArrayRefOps.h
operator<(const ArrayRef<T> &LHS, const ArrayRef<T> &RHS)
#ifndef LLVM_ADT_ARRAYREFOPS_H
#define LLVM_ADT_ARRAYREFOPS_H
#include "llvm/ADT/ArrayRef.h"
namespace llvm {
template <class T>
ArrayRef<T> operator<(const ArrayRef<T> &LHS, const ArrayRef<T> &RHS) {
}
}
#endif // LLVM_ADT_ARRAYREFOPS_H
MacBook Pro

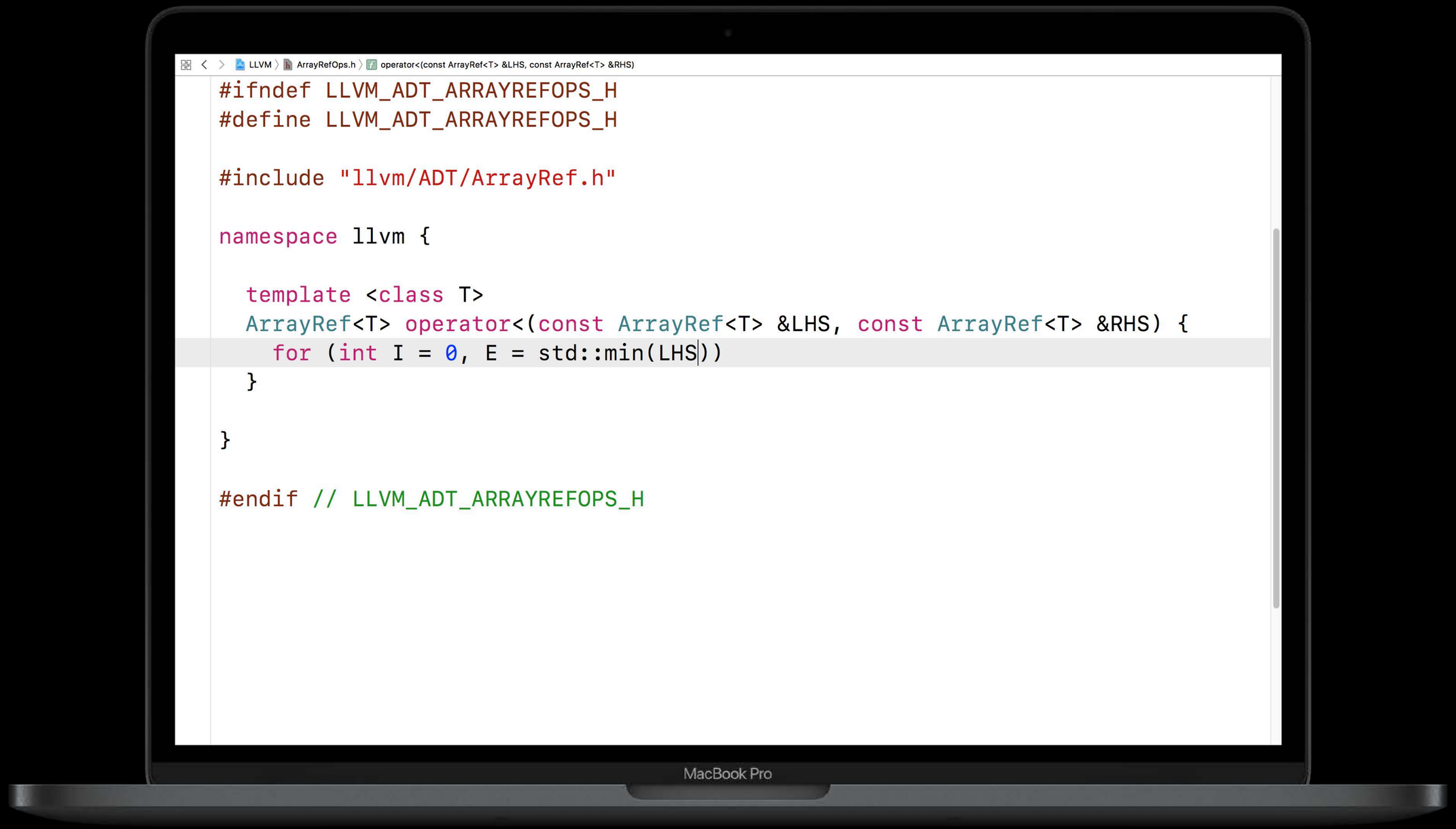

LLVM
ArrayRefOps.h
operator<(const ArrayRef<T> &LHS, const ArrayRef<T> &RHS)
#ifndef LLVM_ADT_ARRAYREFOPS_H
#define LLVM_ADT_ARRAYREFOPS_H
#include "llvm/ADT/ArrayRef.h"
namespace llvm {
template <class T>
ArrayRef<T> operator<(const ArrayRef<T> &LHS, const ArrayRef<T> &RHS) {
}
}
#endif // LLVM_ADT_ARRAYREFOPS_H
MacBook Pro

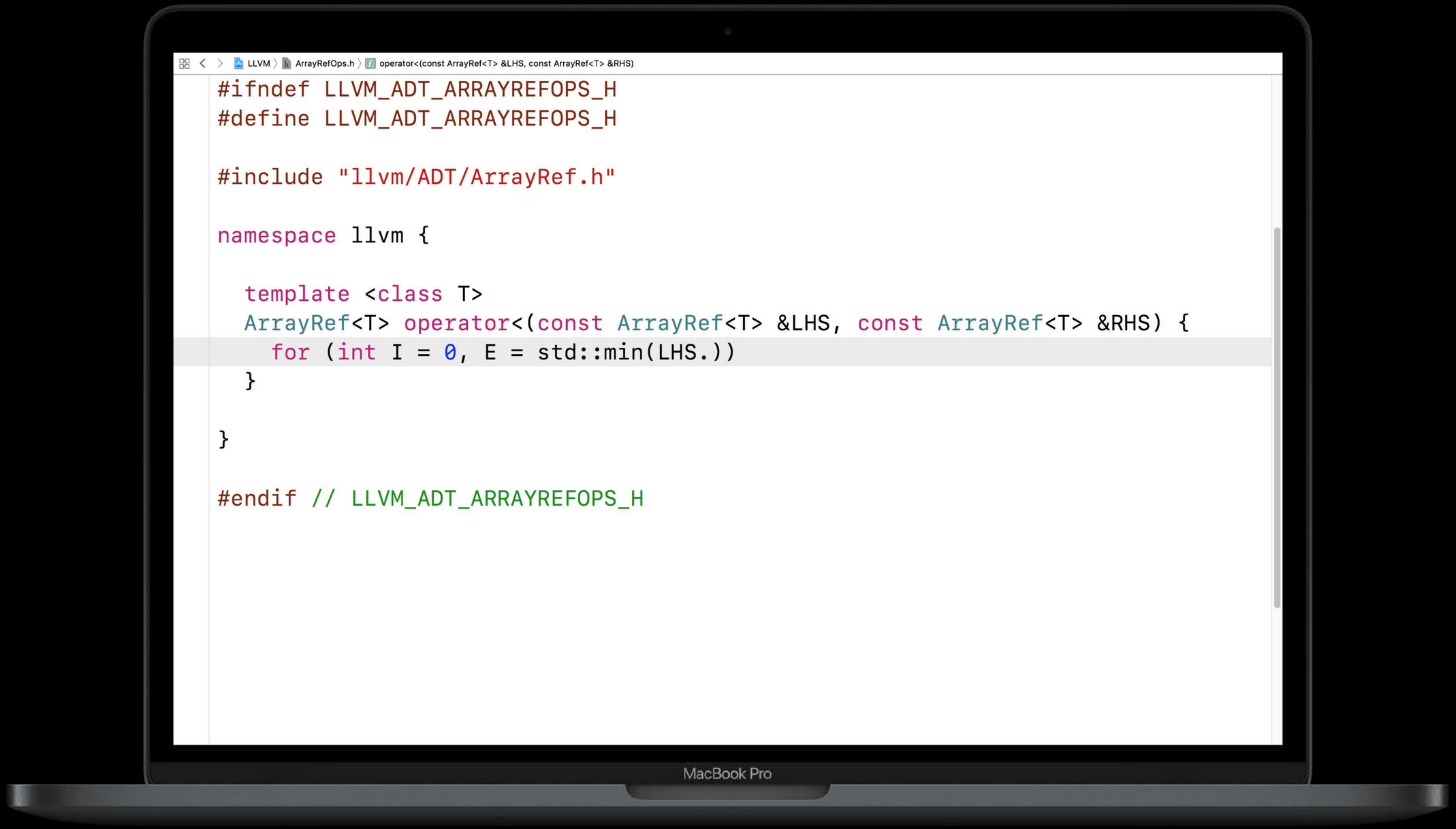

LLVM
ArrayRefOps.h
operator<(const ArrayRef<T> &LHS, const ArrayRef<T> &RHS)
#ifndef LLVM_ADT_ARRAYREFOPS_H
#define LLVM_ADT_ARRAYREFOPS_H
#include "llvm/ADT/ArrayRef.h"
namespace llvm {
template <class T>
ArrayRef<T> operator<(const ArrayRef<T> &LHS, const ArrayRef<T> &RHS) {
}
}
#endif // LLVM_ADT_ARRAYREFOPS_H
MacBook Pro

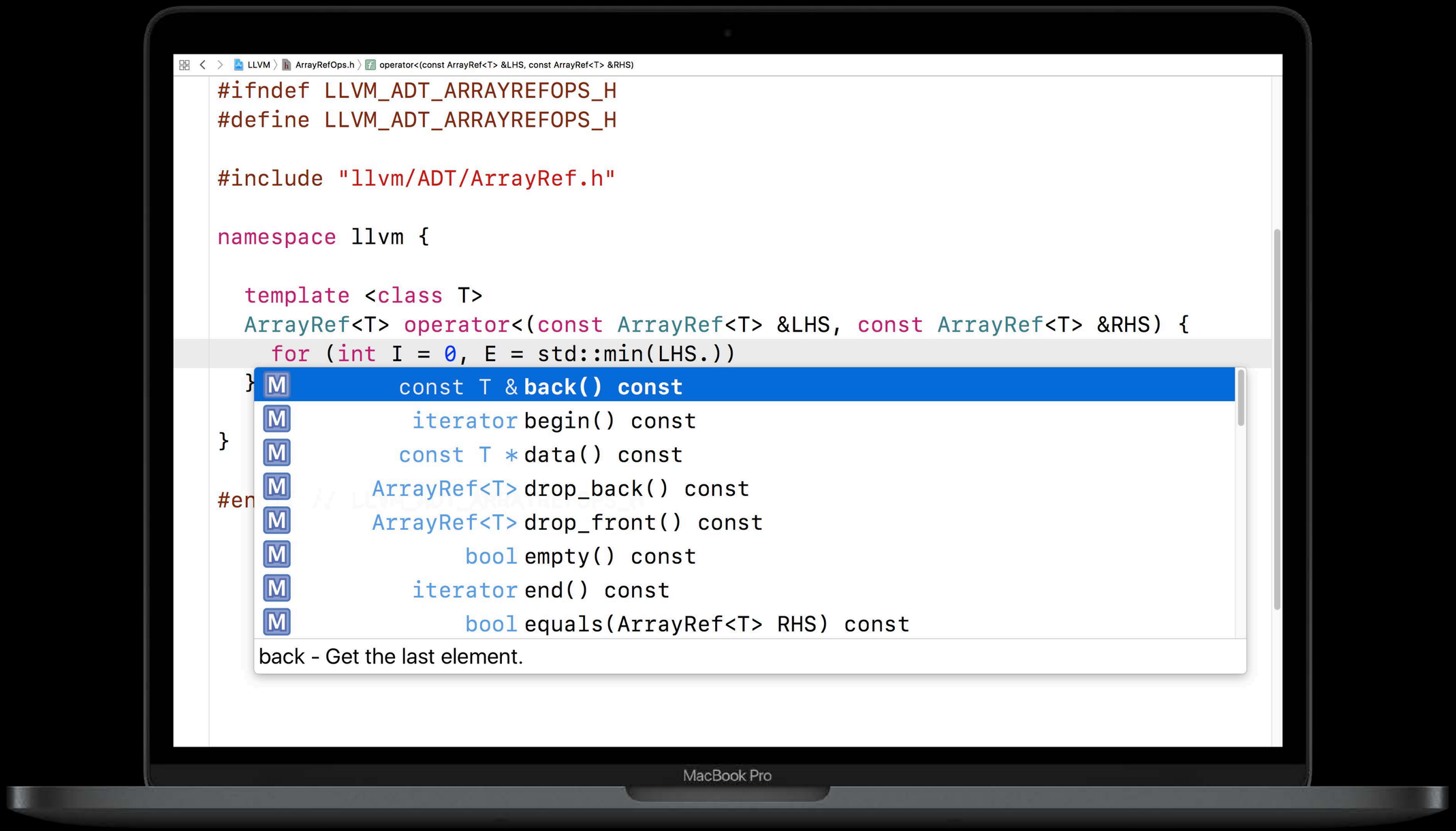

LLVM
ArrayRefOps.h
operator<(const ArrayRef<T> &LHS, const ArrayRef<T> &RHS)
#ifndef LLVM_ADT_ARRAYREFOPS_H
#define LLVM_ADT_ARRAYREFOPS_H
#include "llvm/ADT/ArrayRef.h"
namespace llvm {
template <class T>
ArrayRef<T> operator<(const ArrayRef<T> &LHS, const ArrayRef<T> &RHS) {
for (int I = 0)
}
}
#endif // LLVM_ADT_ARRAYREFOPS_H
MacBook Pro

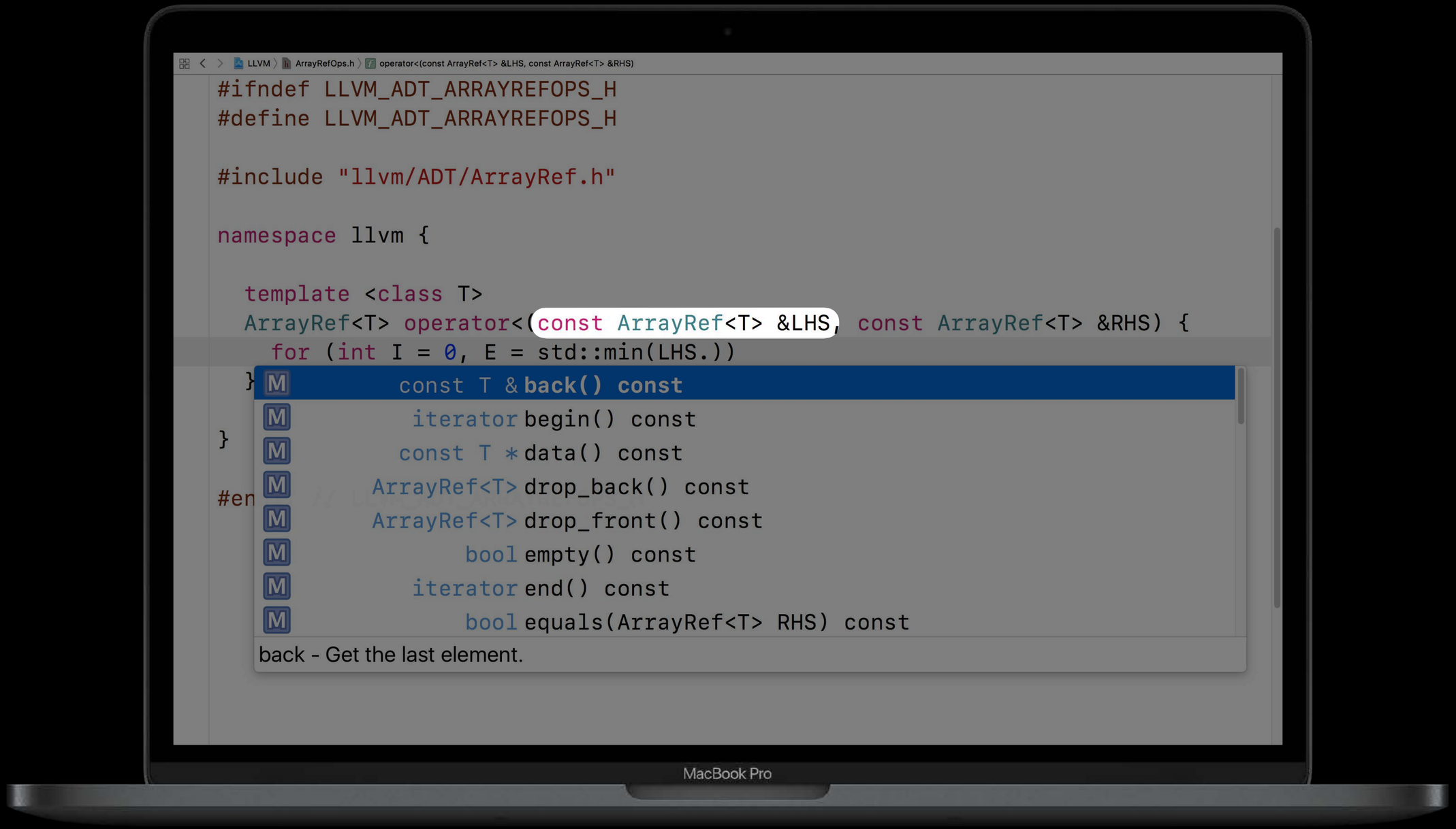

LLVM 〉 ArrayRefOps.h 〉 operator<(const ArrayRef<T> &LHS, const ArrayRef<T> &RHS)
#ifndef LLVM_ADT_ARRAYREFOPS_H
#define LLVM_ADT_ARRAYREFOPS_H
#include "llvm/ADT/ArrayRef.h"
namespace llvm {
template <class T>
ArrayRef<T> operator<(const ArrayRef<T> &LHS, const ArrayRef<T> &RHS) {
for (int I = 0, E =)
}
}
#endif // LLVM_ADT_ARRAYREFOPS_H
MacBook Pro

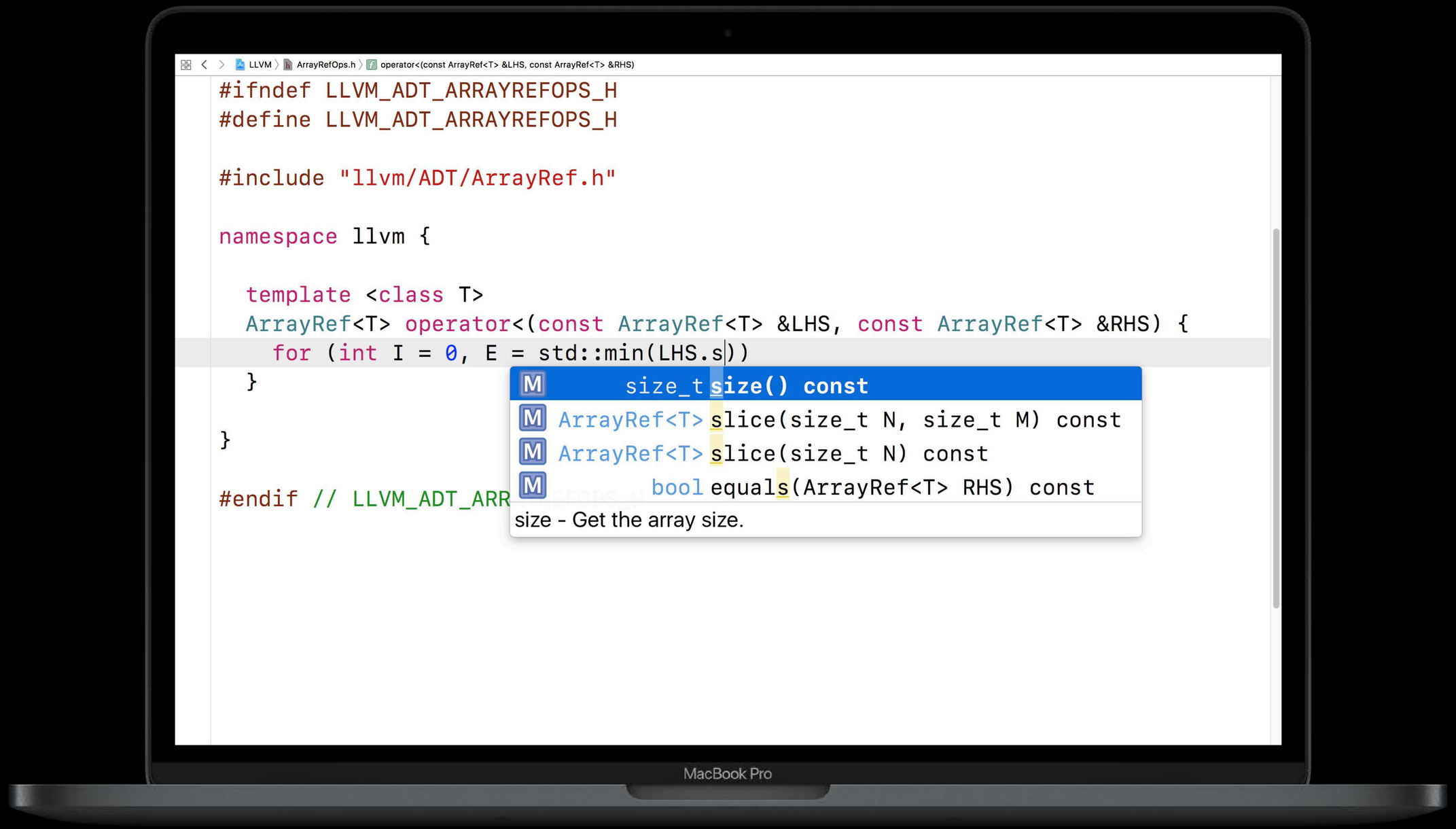

LLVM 〉 ArrayRefOps.h 〉 operator<(const ArrayRef<T> &LHS, const ArrayRef<T> &RHS)
#ifndef LLVM_ADT_ARRAYREFOPS_H
#define LLVM_ADT_ARRAYREFOPS_H
#include "llvm/ADT/ArrayRef.h"
namespace llvm {
template <class T>
ArrayRef<T> operator<(const ArrayRef<T> &LHS, const ArrayRef<T> &RHS) {
for (int I = 0, E = std::min()
}
}
#endif // LLVM_ADT_ARRAYREFOPS_H
MacBook Pro

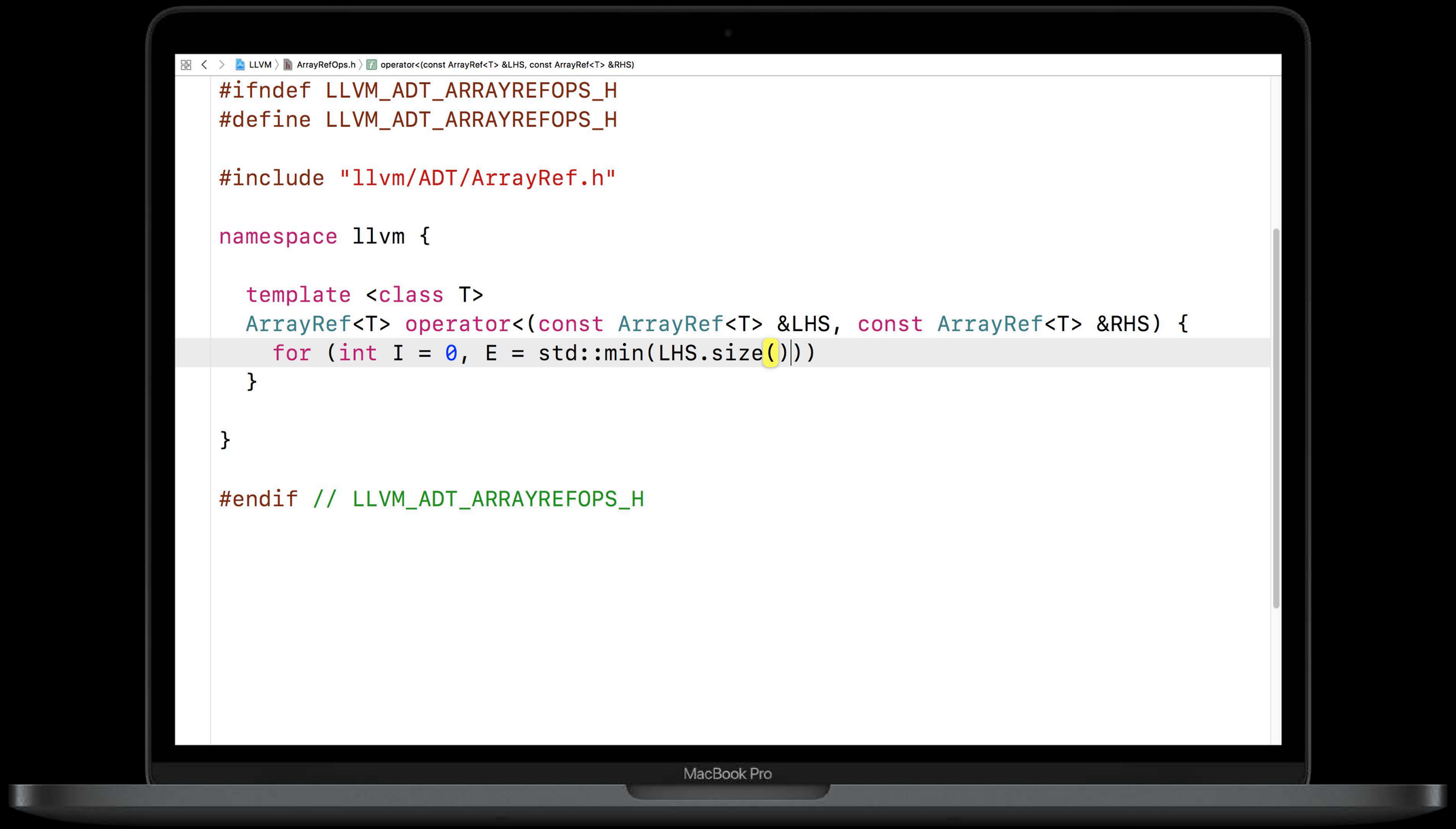

LLVM 〉 ArrayRefOps.h 〉 operator<(const ArrayRef<T> &LHS, const ArrayRef<T> &RHS)
#ifndef LLVM_ADT_ARRAYREFOPS_H
#define LLVM_ADT_ARRAYREFOPS_H
#include "llvm/ADT/ArrayRef.h"
namespace llvm {
template <class T>
ArrayRef<T> operator<(const ArrayRef<T> &LHS, const ArrayRef<T> &RHS) {
for (int I = 0, E = std::min(LHS))
}
}
#endif // LLVM_ADT_ARRAYREFOPS_H
MacBook Pro

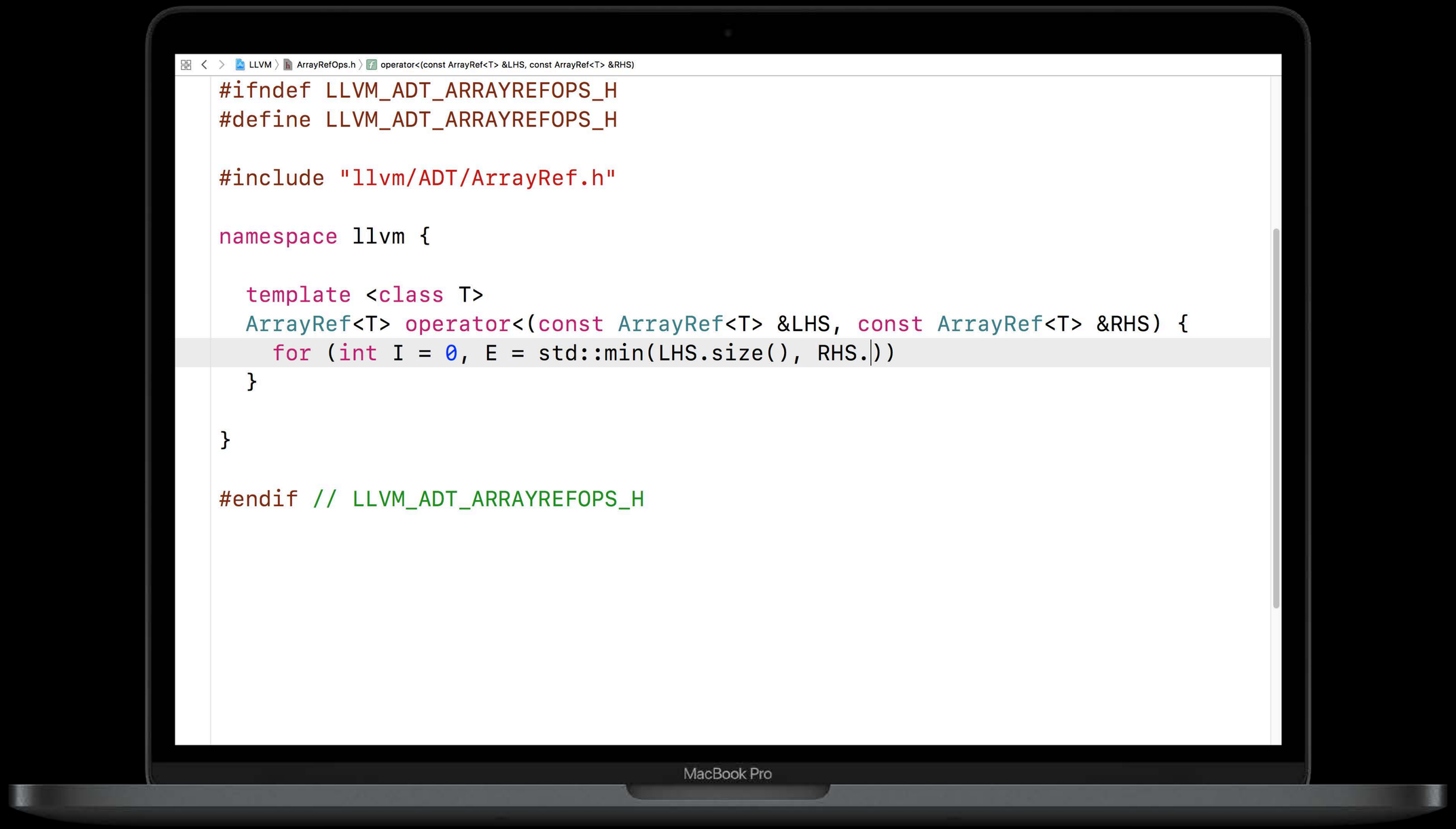

LLVM
ArrayRefOps.h
operator<(const ArrayRef<T> &LHS, const ArrayRef<T> &RHS)
#ifndef LLVM_ADT_ARRAYREFOPS_H
#define LLVM_ADT_ARRAYREFOPS_H
#include "llvm/ADT/ArrayRef.h"
namespace llvm {
template <class T>
ArrayRef<T> operator<(const ArrayRef<T> &LHS, const ArrayRef<T> &RHS) {
for (int I = 0, E = std::min(LHS.))
}
}
#endif // LLVM_ADT_ARRAYREFOPS_H
MacBook Pro

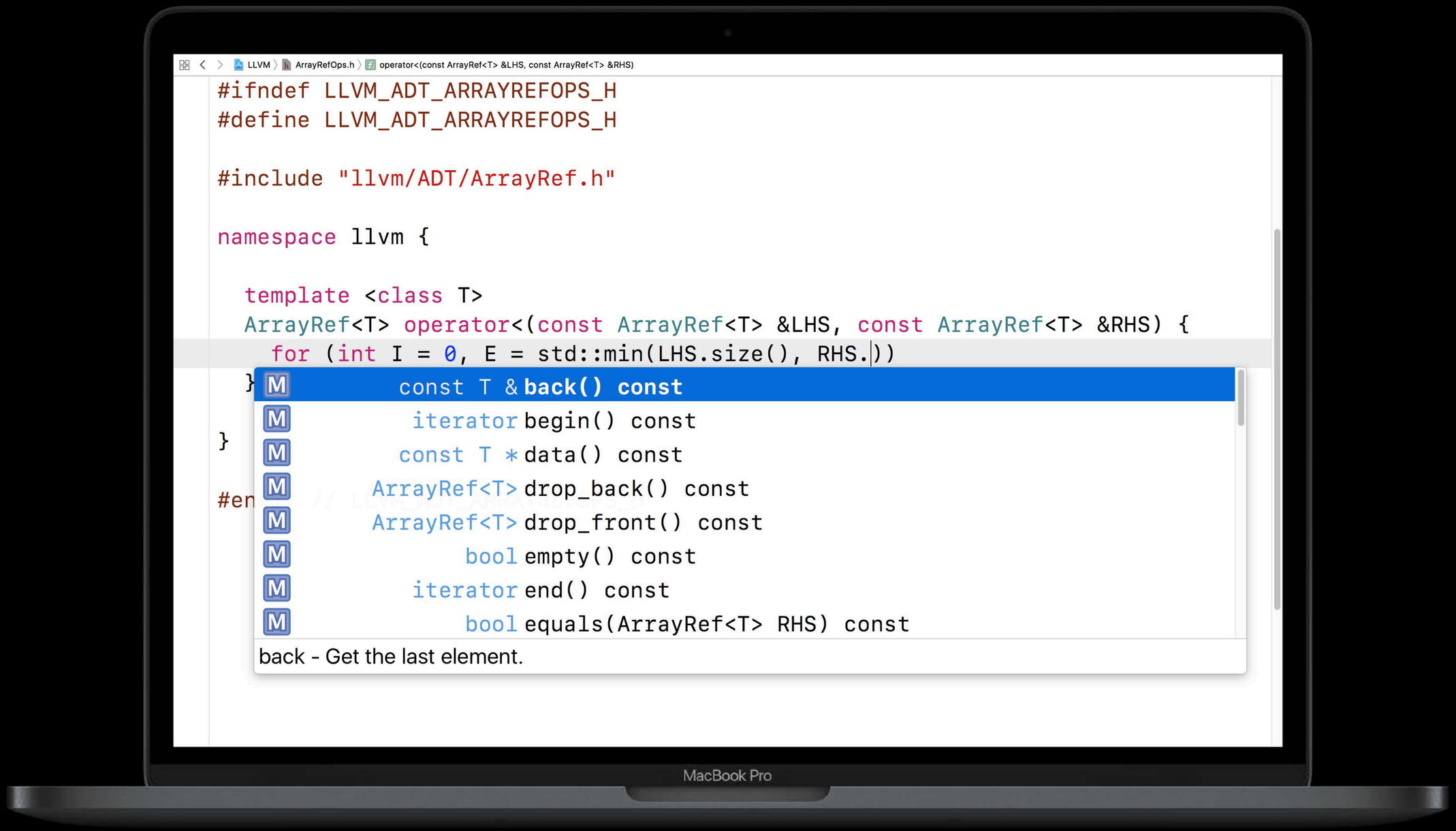

LLVM 〉 ArrayRefOps.h 〉 operator<(const ArrayRef<T> &LHS, const ArrayRef<T> &RHS)
#ifndef LLVM_ADT_ARRAYREFOPS_H
#define LLVM_ADT_ARRAYREFOPS_H
#include "llvm/ADT/ArrayRef.h"
namespace llvm {
template <class T>
ArrayRef<T> operator<(const ArrayRef<T> &LHS, const ArrayRef<T> &RHS) {
for (int I = 0, E = std::min(LHS.))
}
}
#en
const T &back() const
iterator begin() const
const T *data() const
ArrayRef<T> drop_back() const
ArrayRef<T> drop_front() const
bool empty() const
iterator end() const
bool equals(ArrayRef<T> RHS) const
back - Get the last element.
MacBook Pro

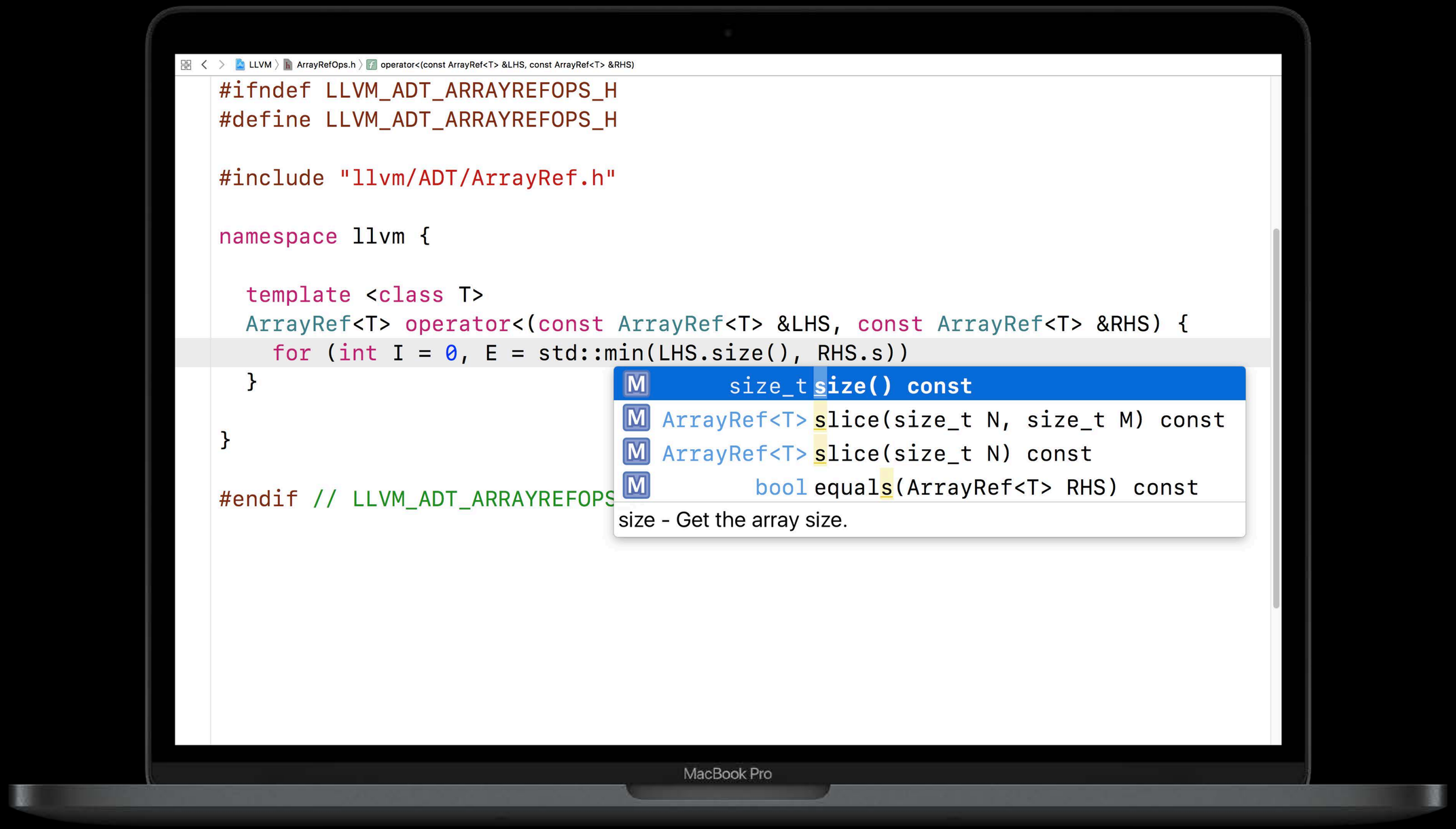

LLVM 〉 ArrayRefOps.h 〉 operator<(const ArrayRef<T> &LHS, const ArrayRef<T> &RHS)
#ifndef LLVM_ADT_ARRAYREFOPS_H
#define LLVM_ADT_ARRAYREFOPS_H
#include "llvm/ADT/ArrayRef.h"
namespace llvm {
template <class T>
ArrayRef<T> operator<(const ArrayRef<T> &LHS, const ArrayRef<T> &RHS) {
for (int I = 0, E = std::min(LHS.))
}
}
#en
const T &back() const
iterator begin() const
const T *data() const
ArrayRef<T> drop_back() const
ArrayRef<T> drop_front() const
bool empty() const
iterator end() const
bool equals(ArrayRef<T> RHS) const
back - Get the last element.
MacBook Pro


LLVM
ArrayRefOps.h
operator<(const ArrayRef<T> &LHS, const ArrayRef<T> &RHS)
#ifndef LLVM_ADT_ARRAYREFOPS_H
#define LLVM_ADT_ARRAYREFOPS_H
#include "llvm/ADT/ArrayRef.h"
namespace llvm {
template <class T>
ArrayRef<T> operator<(const ArrayRef<T> &LHS, const ArrayRef<T> &RHS) {
for (int I = 0, E = std::min(LHS.s))
}
}
#endif // LLVM_ADT_ARR
size_t size() const
ArrayRef<T> slice(size_t N, size_t M) const
ArrayRef<T> slice(size_t N) const
bool equals(ArrayRef<T> RHS) const
size - Get the array size.
MacBook Pro


LLVM
ArrayRefOps.h
operator<(const ArrayRef<T> &LHS, const ArrayRef<T> &RHS)
#ifndef LLVM_ADT_ARRAYREFOPS_H
#define LLVM_ADT_ARRAYREFOPS_H
#include "llvm/ADT/ArrayRef.h"
namespace llvm {
template <class T>
ArrayRef<T> operator<(const ArrayRef<T> &LHS, const ArrayRef<T> &RHS) {
for (int I = 0, E = std::min(LHS.size()))
}
}
#endif // LLVM_ADT_ARRAYREFOPS_H
MacBook Pro


LLVM
ArrayRefOps.h
operator<(const ArrayRef<T> &LHS, const ArrayRef<T> &RHS)
#ifndef LLVM_ADT_ARRAYREFOPS_H
#define LLVM_ADT_ARRAYREFOPS_H
#include "llvm/ADT/ArrayRef.h"
namespace llvm {
template <class T>
ArrayRef<T> operator<(const ArrayRef<T> &LHS, const ArrayRef<T> &RHS) {
for (int I = 0, E = std::min(LHS.size(), RHS.))
}
}
#endif // LLVM_ADT_ARRAYREFOPS_H
MacBook Pro


LLVM 〉 ArrayRefOps.h 〉 operator<(const ArrayRef<T> &LHS, const ArrayRef<T> &RHS)
#ifndef LLVM_ADT_ARRAYREFOPS_H
#define LLVM_ADT_ARRAYREFOPS_H
#include "llvm/ADT/ArrayRef.h"
namespace llvm {
template <class T>
ArrayRef<T> operator<(const ArrayRef<T> &LHS, const ArrayRef<T> &RHS) {
for (int I = 0, E = std::min(LHS.size(), RHS.))
}
}
#en
const T &back() const
iterator begin() const
const T *data() const
ArrayRef<T> drop_back() const
ArrayRef<T> drop_front() const
bool empty() const
iterator end() const
bool equals(ArrayRef<T> RHS) const
back - Get the last element.
MacBook Pro


LLVM
ArrayRefOps.h
operator<(const ArrayRef<T> &LHS, const ArrayRef<T> &RHS)
#ifndef LLVM_ADT_ARRAYREFOPS_H
#define LLVM_ADT_ARRAYREFOPS_H
#include "llvm/ADT/ArrayRef.h"
namespace llvm {
template <class T>
ArrayRef<T> operator<(const ArrayRef<T> &LHS, const ArrayRef<T> &RHS) {
for (int I = 0, E = std::min(LHS.size(), RHS.s))
}
}
#endif // LLVM_ADT_ARRAYREFOPS
size_t size() const
ArrayRef<T> slice(size_t N, size_t M) const
ArrayRef<T> slice(size_t N) const
bool equals(ArrayRef<T> RHS) const
size - Get the array size.
MacBook Pro

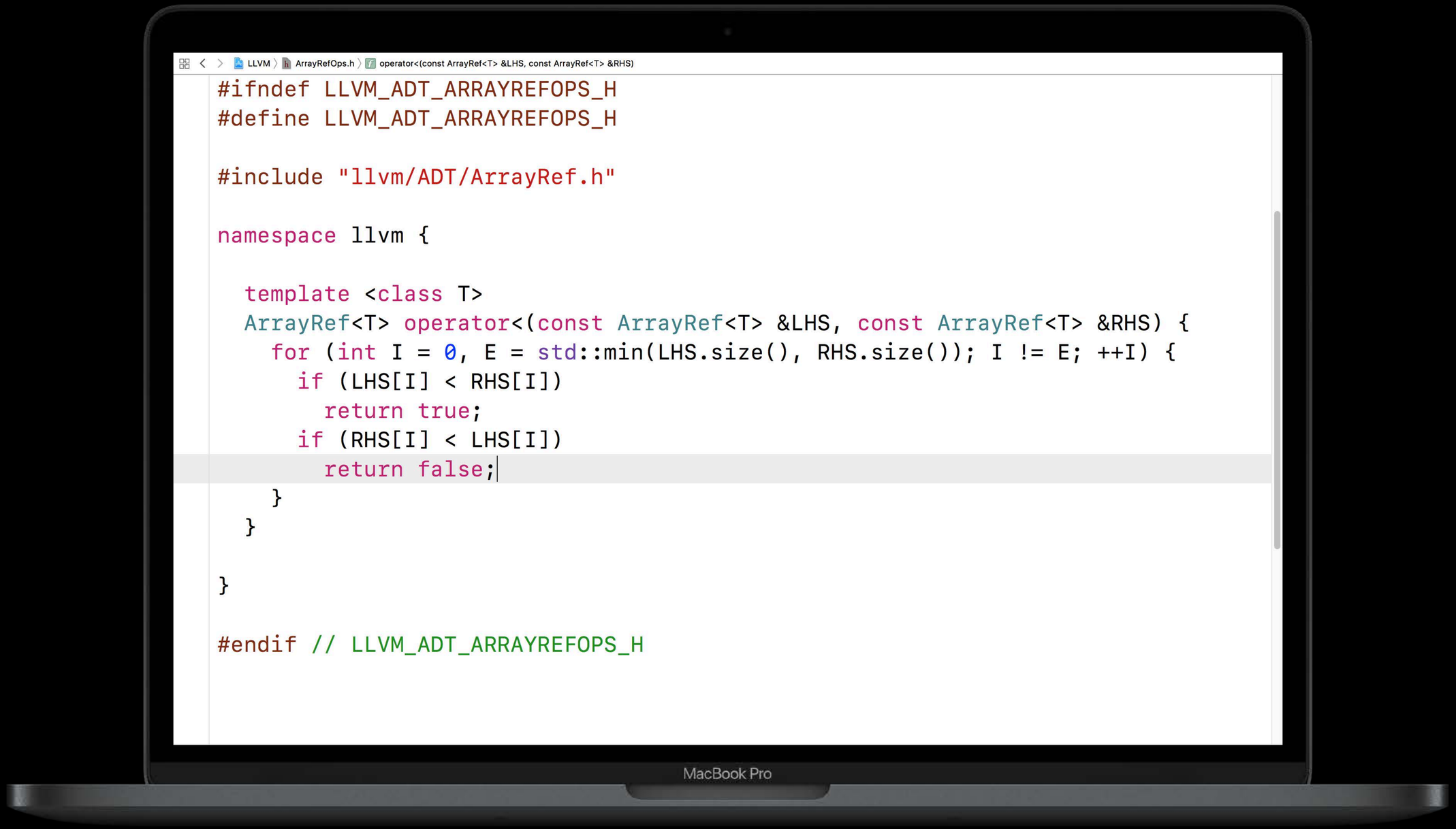

LLVM
ArrayRefOps.h
operator<(const ArrayRef<T> &LHS, const ArrayRef<T> &RHS)
#ifndef LLVM_ADT_ARRAYREFOPS_H
#define LLVM_ADT_ARRAYREFOPS_H
#include "llvm/ADT/ArrayRef.h"
namespace llvm {
template <class T>
ArrayRef<T> operator<(const ArrayRef<T> &LHS, const ArrayRef<T> &RHS) {
for (int I = 0, E = std::min(LHS.size(), RHS.size()))
}
}
#endif // LLVM_ADT_ARRAYREFOPS_H
MacBook Pro


LLVM 〉 ArrayRefOps.h 〉 operator<(const ArrayRef<T> &LHS, const ArrayRef<T> &RHS)
#ifndef LLVM_ADT_ARRAYREFOPS_H
#define LLVM_ADT_ARRAYREFOPS_H
#include "llvm/ADT/ArrayRef.h"
namespace llvm {
template <class T>
ArrayRef<T> operator<(const ArrayRef<T> &LHS, const ArrayRef<T> &RHS) {
for (int I = 0, E = std::min(LHS.size(), RHS.size()); I != E;
}
}
#endif // LLVM_ADT_ARRAYREFOPS_H
MacBook Pro

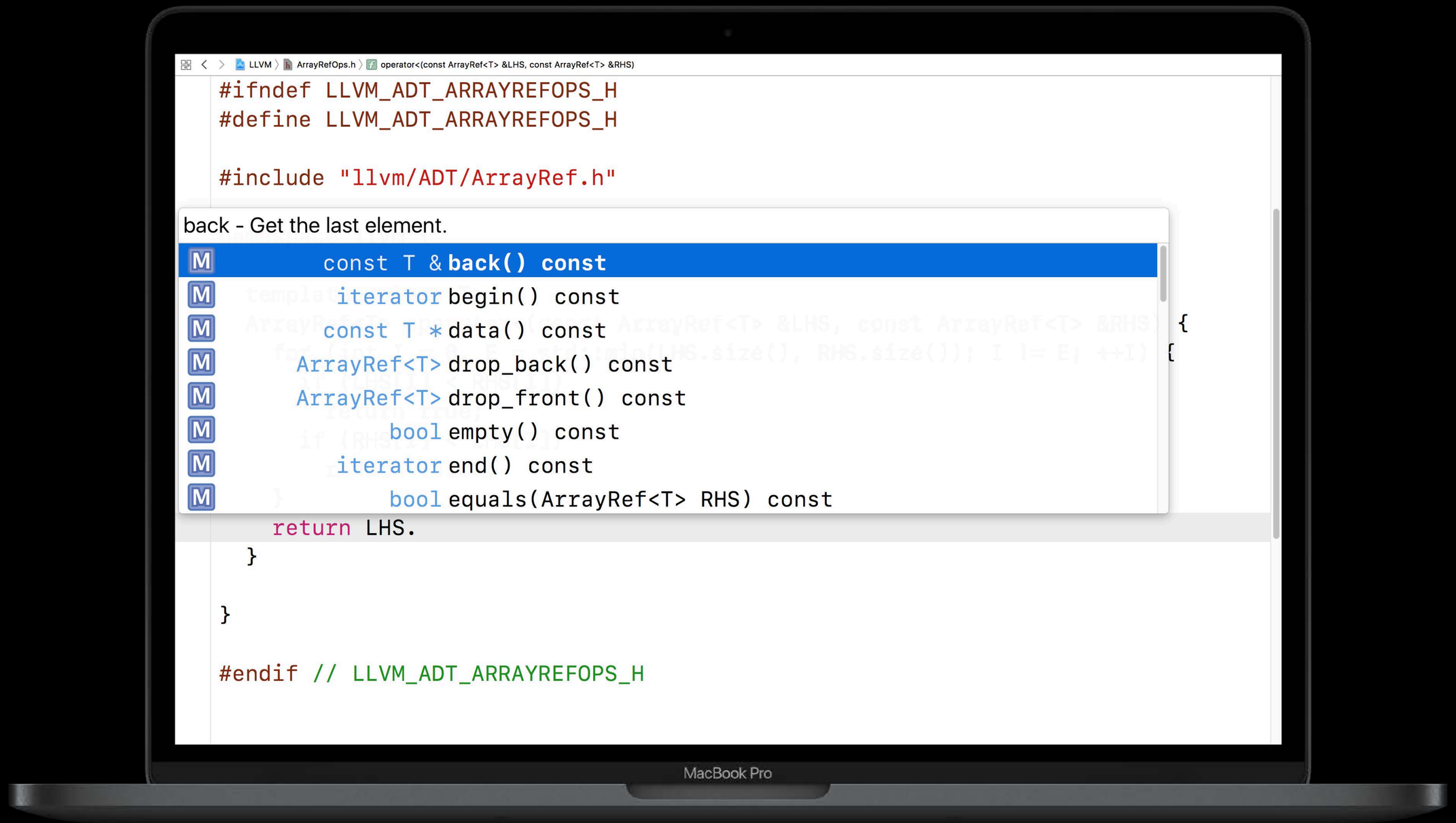

LLVM 〉 ArrayRefOps.h 〉 operator<(const ArrayRef<T> &LHS, const ArrayRef<T> &RHS)
#ifndef LLVM_ADT_ARRAYREFOPS_H
#define LLVM_ADT_ARRAYREFOPS_H
#include "llvm/ADT/ArrayRef.h"
namespace llvm {
template <class T>
ArrayRef<T> operator<(const ArrayRef<T> &LHS, const ArrayRef<T> &RHS) {
for (int I = 0, E = std::min(LHS.size(), RHS.size()); I != E; ++I)
}
}
#endif // LLVM_ADT_ARRAYREFOPS_H
MacBook Pro

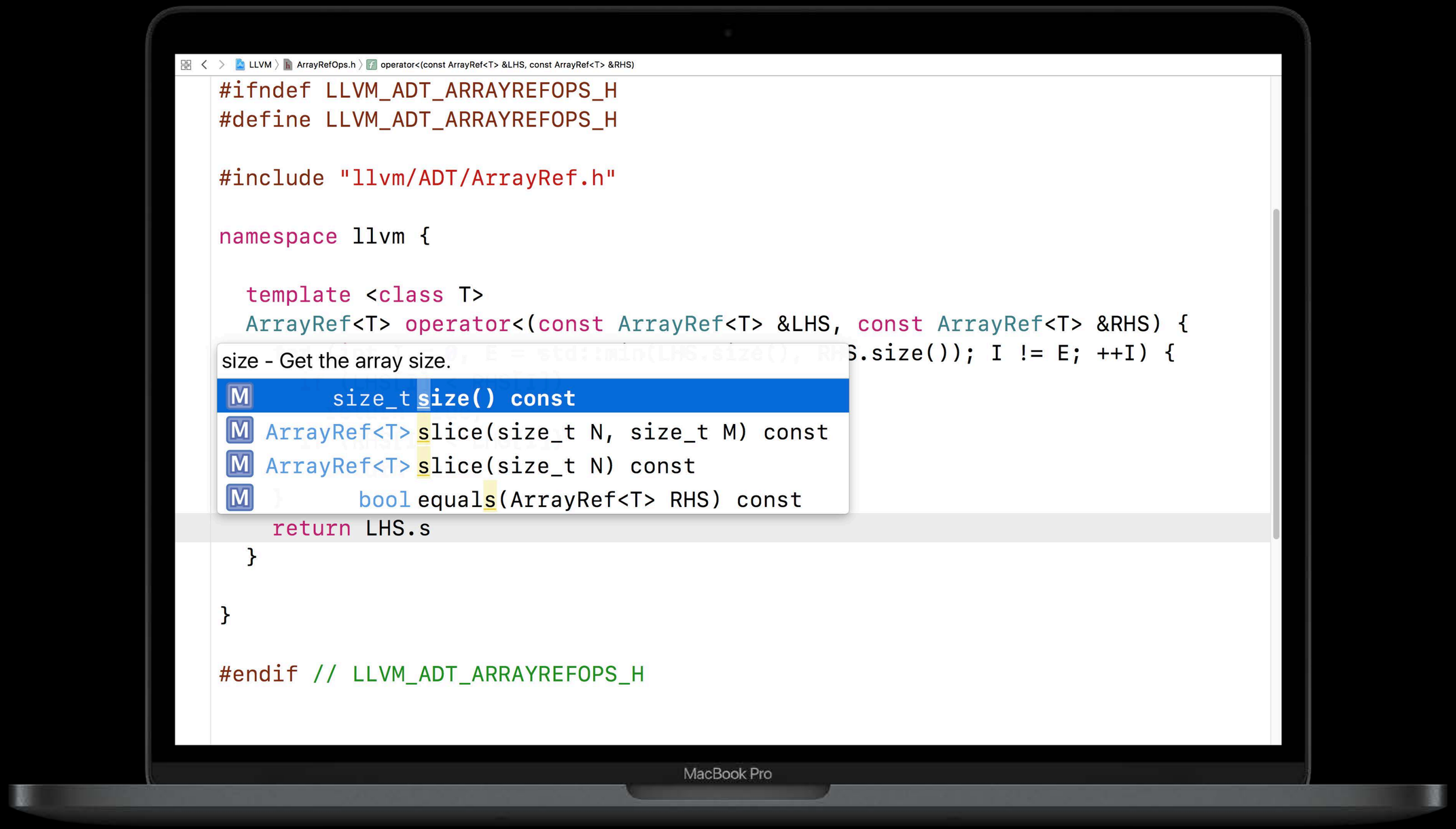

LLVM 〉 ArrayRefOps.h 〉 operator<(const ArrayRef<T> &LHS, const ArrayRef<T> &RHS)
#ifndef LLVM_ADT_ARRAYREFOPS_H
#define LLVM_ADT_ARRAYREFOPS_H
#include "llvm/ADT/ArrayRef.h"
namespace llvm {
template <class T>
ArrayRef<T> operator<(const ArrayRef<T> &LHS, const ArrayRef<T> &RHS) {
for (int I = 0, E = std::min(LHS.size(), RHS.size()); I != E; ++I) {
}
}
}
#endif // LLVM_ADT_ARRAYREFOPS_H
MacBook Pro

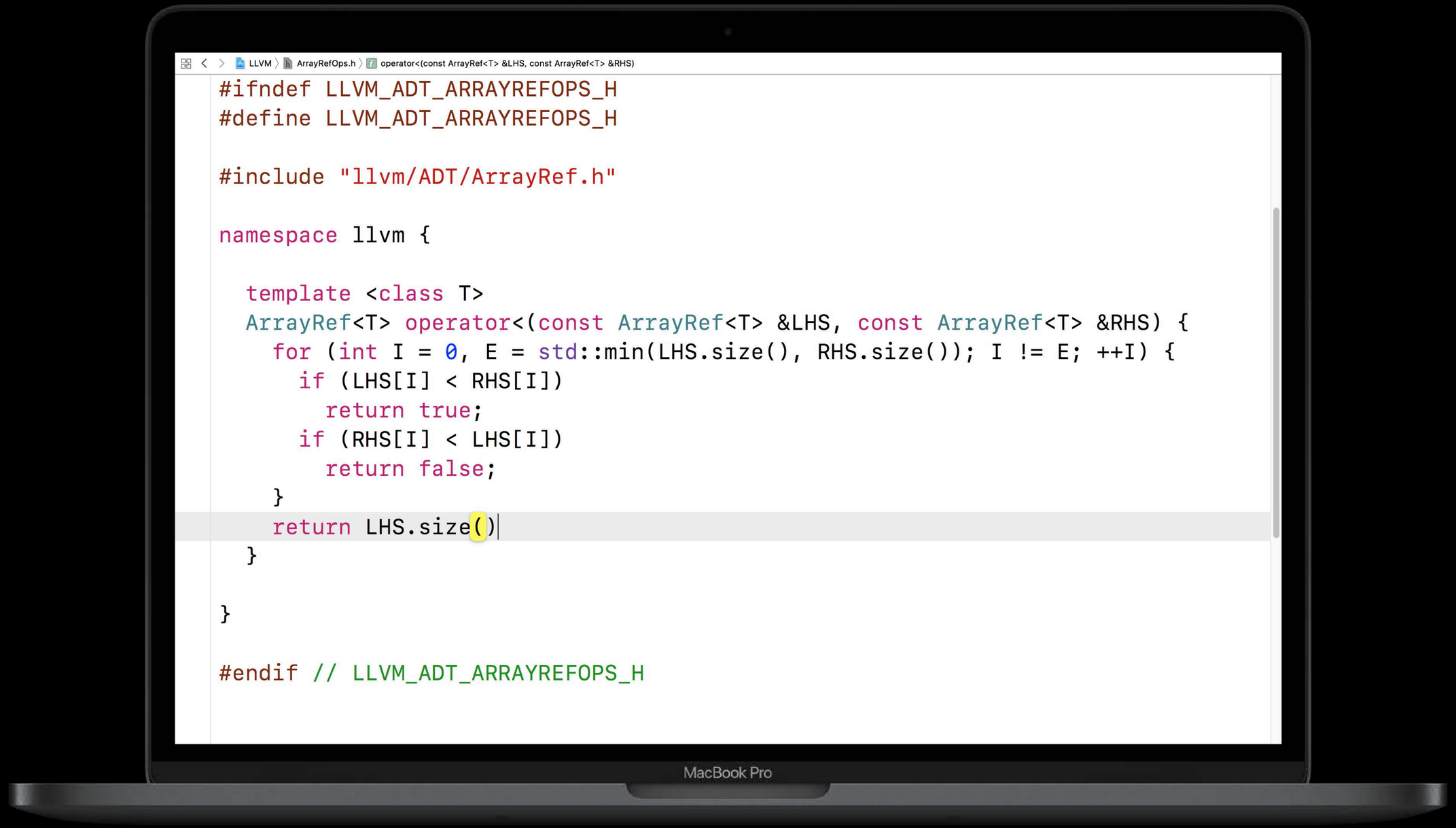

LLVM
ArrayRefOps.h
operator<(const ArrayRef<T> &LHS, const ArrayRef<T> &RHS)
#ifndef LLVM_ADT_ARRAYREFOPS_H
#define LLVM_ADT_ARRAYREFOPS_H
#include "llvm/ADT/ArrayRef.h"
namespace llvm {
template <class T>
ArrayRef<T> operator<(const ArrayRef<T> &LHS, const ArrayRef<T> &RHS) {
for (int I = 0, E = std::min(LHS.size(), RHS.size()); I != E; ++I) {
if (LHS[I] < RHS[I])
return true;
}
}
}
#endif // LLVM_ADT_ARRAYREFOPS_H
MacBook Pro

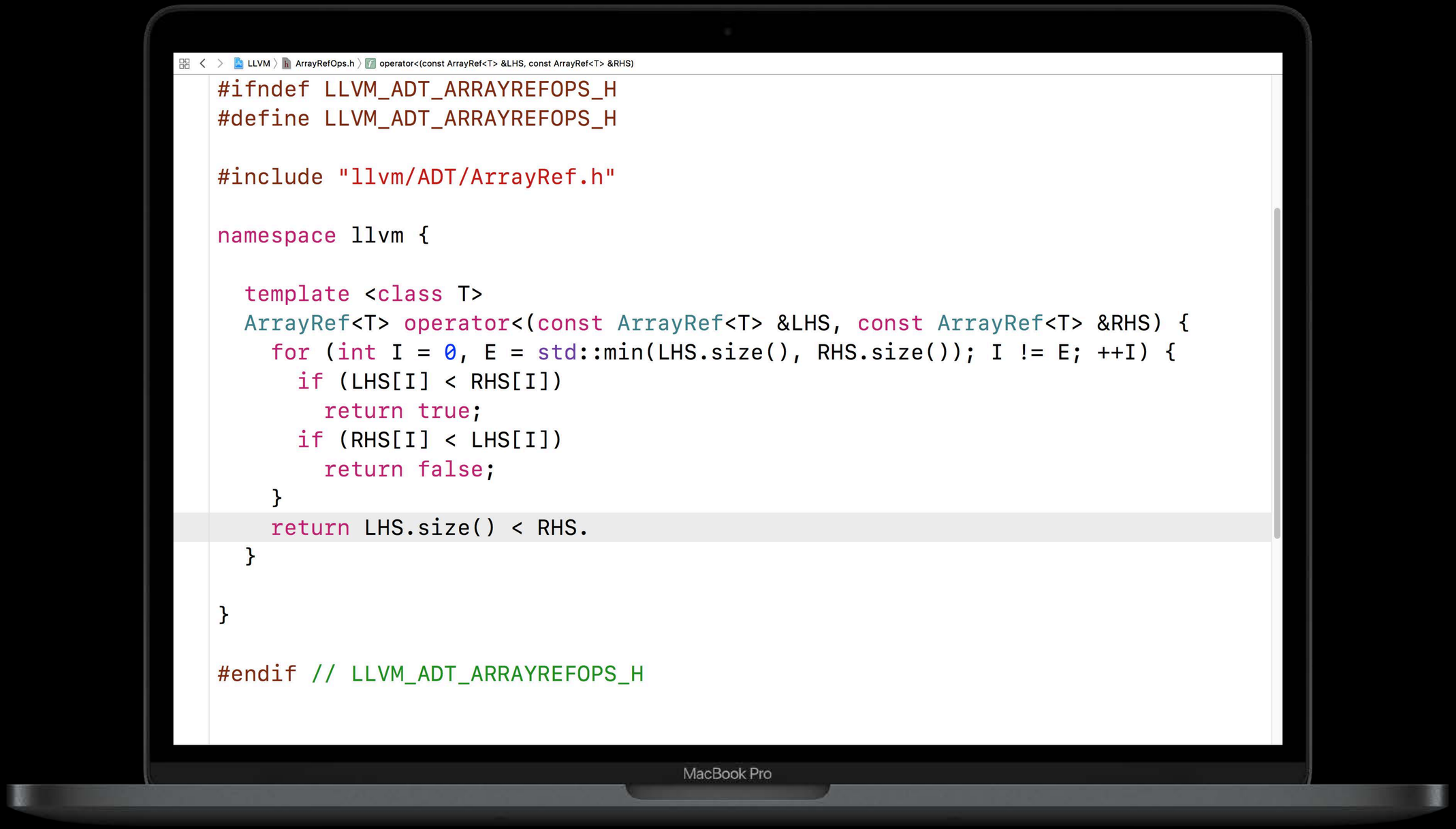

LLVM
ArrayRefOps.h
operator<(const ArrayRef<T> &LHS, const ArrayRef<T> &RHS)
#ifndef LLVM_ADT_ARRAYREFOPS_H
#define LLVM_ADT_ARRAYREFOPS_H
#include "llvm/ADT/ArrayRef.h"
namespace llvm {
template <class T>
ArrayRef<T> operator<(const ArrayRef<T> &LHS, const ArrayRef<T> &RHS) {
for (int I = 0, E = std::min(LHS.size(), RHS.size()); I != E; ++I) {
if (LHS[I] < RHS[I])
return true;
if (RHS[I] < LHS[I])
return false;
}
}
}
#endif // LLVM_ADT_ARRAYREFOPS_H
MacBook Pro

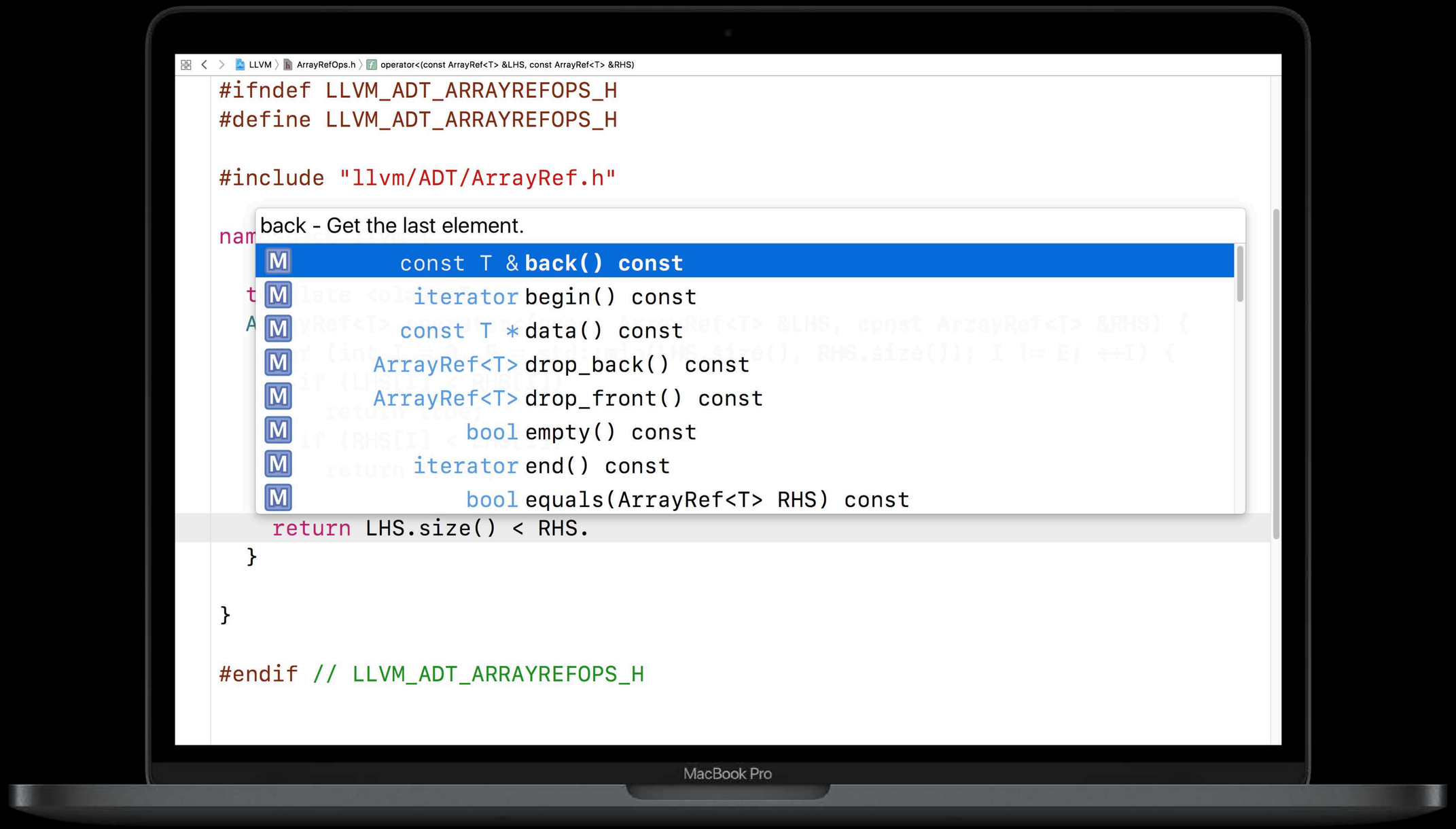

LLVM
ArrayRefOps.h
operator<(const ArrayRef<T> &LHS, const ArrayRef<T> &RHS)
#ifndef LLVM_ADT_ARRAYREFOPS_H
#define LLVM_ADT_ARRAYREFOPS_H
#include "llvm/ADT/ArrayRef.h"
namespace llvm {
template <class T>
ArrayRef<T> operator<(const ArrayRef<T> &LHS, const ArrayRef<T> &RHS) {
for (int I = 0, E = std::min(LHS.size(), RHS.size()); I != E; ++I) {
if (LHS[I] < RHS[I])
return true;
if (RHS[I] < LHS[I])
return false;
}
return LHS.
}
}
#endif // LLVM_ADT_ARRAYREFOPS_H
MacBook Pro

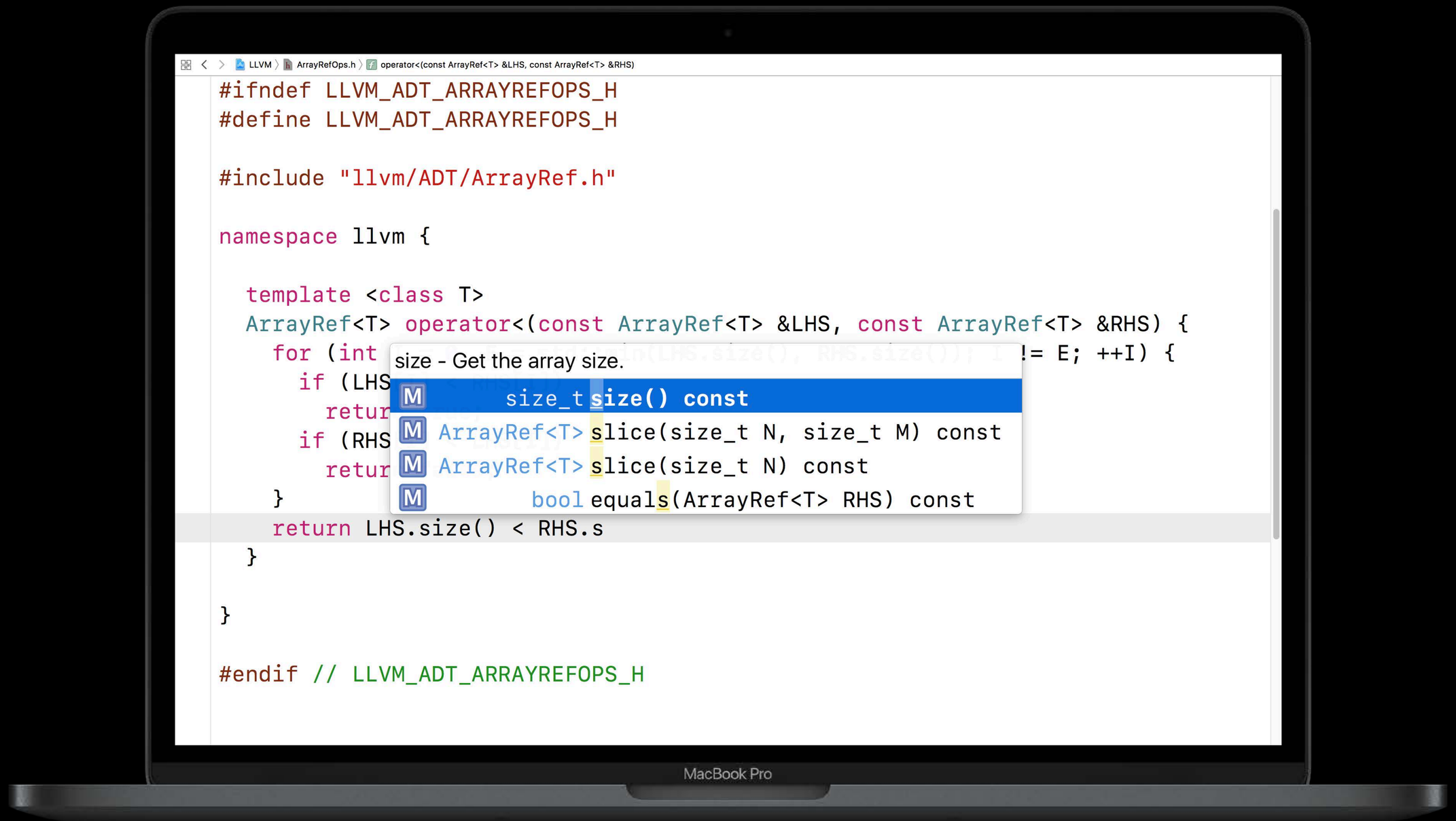

LLVM
ArrayRefOps.h
operator<(const ArrayRef<T> &LHS, const ArrayRef<T> &RHS)
#ifndef LLVM_ADT_ARRAYREFOPS_H
#define LLVM_ADT_ARRAYREFOPS_H
#include "llvm/ADT/ArrayRef.h"
back - Get the last element.
const T &back() const
iterator begin() const
const T *data() const
ArrayRef<T> drop_back() const
ArrayRef<T> drop_front() const
bool empty() const
iterator end() const
bool equals(ArrayRef<T> RHS) const
return LHS.
}
}
#endif // LLVM_ADT_ARRAYREFOPS_H
MacBook Pro

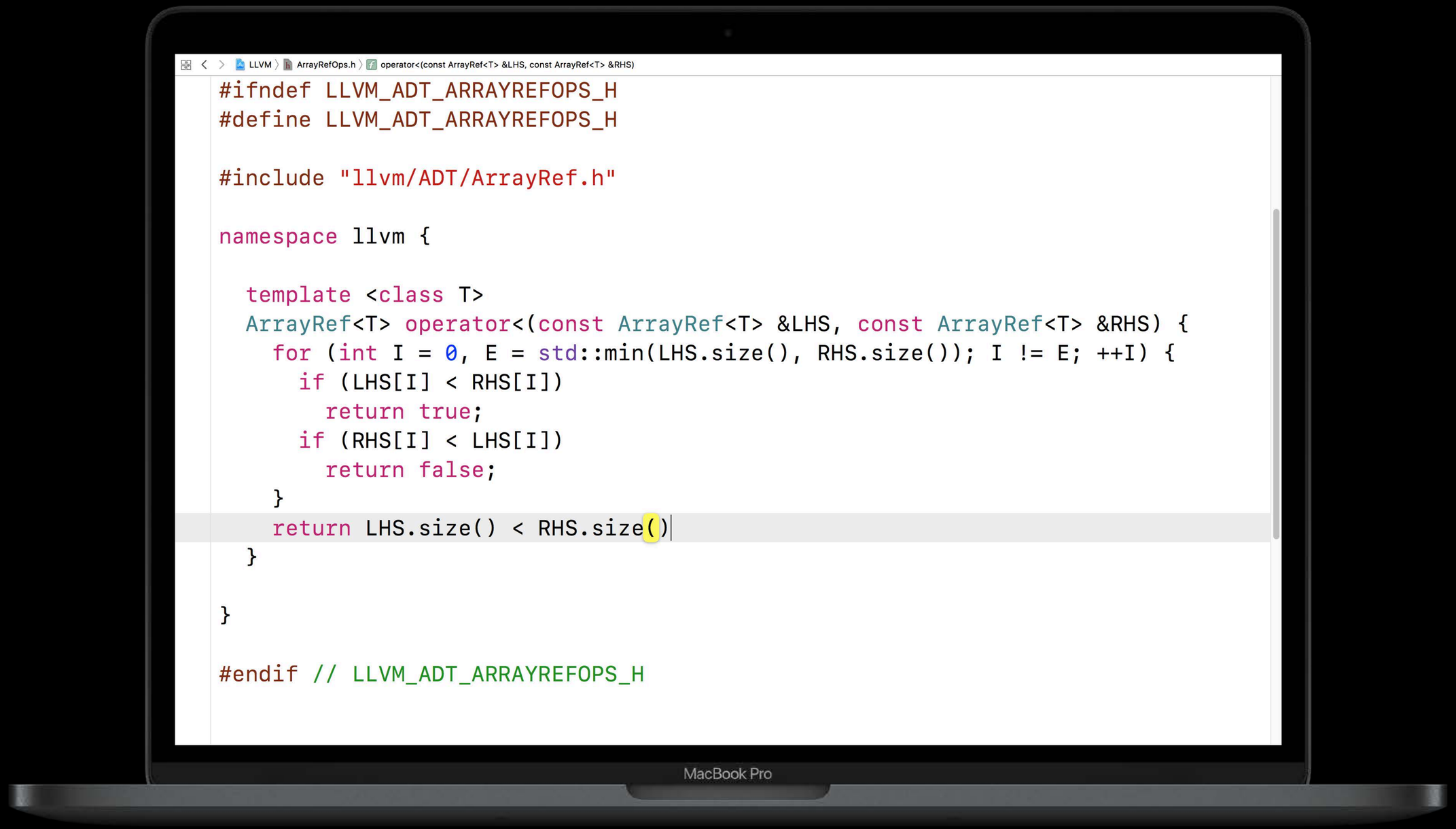

LLVM
ArrayRefOps.h
operator<(const ArrayRef<T> &LHS, const ArrayRef<T> &RHS)
#ifndef LLVM_ADT_ARRAYREFOPS_H
#define LLVM_ADT_ARRAYREFOPS_H
#include "llvm/ADT/ArrayRef.h"
namespace llvm {
template <class T>
ArrayRef<T> operator<(const ArrayRef<T> &LHS, const ArrayRef<T> &RHS) {
.size()); I != E; ++I) {
size - Get the array size.
size_t size() const
ArrayRef<T> slice(size_t N, size_t M) const
ArrayRef<T> slice(size_t N) const
bool equals(ArrayRef<T> RHS) const
return LHS.s
}
}
#endif // LLVM_ADT_ARRAYREFOPS_H
MacBook Pro


LLVM
ArrayRefOps.h
operator<(const ArrayRef<T> &LHS, const ArrayRef<T> &RHS)
#ifndef LLVM_ADT_ARRAYREFOPS_H
#define LLVM_ADT_ARRAYREFOPS_H
#include "llvm/ADT/ArrayRef.h"
namespace llvm {
template <class T>
ArrayRef<T> operator<(const ArrayRef<T> &LHS, const ArrayRef<T> &RHS) {
for (int I = 0, E = std::min(LHS.size(), RHS.size()); I != E; ++I) {
if (LHS[I] < RHS[I])
return true;
if (RHS[I] < LHS[I])
return false;
}
return LHS.size()
}
}
#endif // LLVM_ADT_ARRAYREFOPS_H
MacBook Pro


LLVM
ArrayRefOps.h
operator<(const ArrayRef<T> &LHS, const ArrayRef<T> &RHS)
#ifndef LLVM_ADT_ARRAYREFOPS_H
#define LLVM_ADT_ARRAYREFOPS_H
#include "llvm/ADT/ArrayRef.h"
namespace llvm {
template <class T>
ArrayRef<T> operator<(const ArrayRef<T> &LHS, const ArrayRef<T> &RHS) {
for (int I = 0, E = std::min(LHS.size(), RHS.size()); I != E; ++I) {
if (LHS[I] < RHS[I])
return true;
if (RHS[I] < LHS[I])
return false;
}
return LHS.size() < RHS.
}
}
#endif // LLVM_ADT_ARRAYREFOPS_H
MacBook Pro


LLVM 〉 ArrayRefOps.h 〉 operator<(const ArrayRef<T> &LHS, const ArrayRef<T> &RHS)
#ifndef LLVM_ADT_ARRAYREFOPS_H
#define LLVM_ADT_ARRAYREFOPS_H
#include "llvm/ADT/ArrayRef.h"
nam
back - Get the last element.
const T &back() const
iterator begin() const
const T *data() const
ArrayRef<T> drop_back() const
ArrayRef<T> drop_front() const
bool empty() const
iterator end() const
bool equals(ArrayRef<T> RHS) const
return LHS.size() < RHS.
}
}
#endif // LLVM_ADT_ARRAYREFOPS_H
MacBook Pro


LLVM 〉 ArrayRefOps.h 〉 operator<(const ArrayRef<T> &LHS, const ArrayRef<T> &RHS)
#ifndef LLVM_ADT_ARRAYREFOPS_H
#define LLVM_ADT_ARRAYREFOPS_H
#include "llvm/ADT/ArrayRef.h"
namespace llvm {
template <class T>
ArrayRef<T> operator<(const ArrayRef<T> &LHS, const ArrayRef<T> &RHS) {
for (int
if (LHS
retur
if (RHS
retur
}
return LHS.size() < RHS.s
}
}
#endif // LLVM_ADT_ARRAYREFOPS_H
size - Get the array size.
size_t size() const
ArrayRef<T> slice(size_t N, size_t M) const
ArrayRef<T> slice(size_t N) const
bool equals(ArrayRef<T> RHS) const
MacBook Pro


LLVM
ArrayRefOps.h
operator<(const ArrayRef<T> &LHS, const ArrayRef<T> &RHS)
#ifndef LLVM_ADT_ARRAYREFOPS_H
#define LLVM_ADT_ARRAYREFOPS_H
#include "llvm/ADT/ArrayRef.h"
namespace llvm {
template <class T>
ArrayRef<T> operator<(const ArrayRef<T> &LHS, const ArrayRef<T> &RHS) {
for (int I = 0, E = std::min(LHS.size(), RHS.size()); I != E; ++I) {
if (LHS[I] < RHS[I])
return true;
if (RHS[I] < LHS[I])
return false;
}
return LHS.size() < RHS.size()
}
}
#endif // LLVM_ADT_ARRAYREFOPS_H
MacBook Pro


LLVM
ArrayRefOps.h
operator<(const ArrayRef<T> &LHS, const ArrayRef<T> &RHS)
#ifndef LLVM_ADT_ARRAYREFOPS_H
#define LLVM_ADT_ARRAYREFOPS_H
#include "llvm/ADT/ArrayRef.h"
namespace llvm {
template <class T>
ArrayRef<T> operator<(const ArrayRef<T> &LHS, const ArrayRef<T> &RHS) {
for (int I = 0, E = std::min(LHS.size(), RHS.size()); I != E; ++I) {
if (LHS[I] < RHS[I])
return true;
if (RHS[I] < LHS[I])
return false;
}
return LHS.size() < RHS.size();
}
}
#endif // LLVM_ADT_ARRAYREFOPS_H
MacBook Pro

Features from C++17

Decomposing a Returned Tuple Is Awkward

```
std::tuple<int, double, char> compute();
void run() {
  int a; double b; char c;
  std::tie(a, b, c) = compute();
  ...
}
```

Decomposing a Returned Tuple Is Awkward

```
std::tuple<int, double, char> compute();
void run() {
  int a; double b; char c;
  std::tie(a, b, c) = compute();
  ...
}
```

Boilerplate `std::tie`

Decomposing a Returned Tuple Is Awkward

```
std::tuple<int, double, char> compute();
void run() {
  int a; double b; char c;
  std::tie(a, b, c) = compute();
  ...
}
```

Boilerplate `std::tie`

Cannot infer types

Decomposing a Returned Tuple Is Awkward

```
std::tuple<int, double, char> compute();
void run() {
  int a; double b; char c;
  std::tie(a, b, c) = compute();
  ...
}
```

Boilerplate `std::tie`

Cannot infer types

Repeated variable names

Structured Binding

Naturally decompose tuple-like types

```
std::tuple<int, double, char> compute();
void run() {
  int a; double b; char c;
  std::tie(a, b, c) = compute();
  ...
}
```

Structured Binding

Naturally decompose tuple-like types

```
std::tuple<int, double, char> compute();
void run() {
       [a, b, c] = compute();
  ...
}
```

Structured Binding

Naturally decompose tuple-like types

NEW

```
std::tuple<int, double, char> compute();
void run() {
  auto [a, b, c] = compute();
  ...
}
```

Use `auto [...]` to decompose `std::tuple`

Structured Binding

Naturally decompose tuple-like types

NEW

```
std::tuple<int, double, char> compute();
void run() {
  auto [a, b, c] = compute();
  ...
}
```

Use `auto [...]` to decompose `std::tuple`

Supports anything that implements `std::get`

Structured Binding

Naturally decompose tuple-like types

NEW

```
struct Point { double x; double y; double z; };
Point computeMidpoint(Point p1, Point p2);
...
auto [x, y, z] = computeMidpoint(src, dest);
```

Use `auto [...]` to decompose `std::tuple`

Supports anything that implements `std::get`

Supports plain-old data types

Structured Binding

Naturally decompose tuple-like types

NEW

```
struct Point { double x; double y; double z; };
Point computeMidpoint(Point p1, Point p2);
...
auto [x, y, z] = computeMidpoint(src, dest);
```

Use `auto [...]` to decompose `std::tuple`

Supports anything that implements `std::get`

Supports plain-old data types

Structured Binding

Naturally decompose tuple-like types

NEW

```
struct Point { double x; double y; double z; };
Point computeMidpoint(Point p1, Point p2);
...
auto [x, y, z] = computeMidpoint(src, dest);
```

Use `auto [...]` to decompose `std::tuple`

Supports anything that implements `std::get`

Supports plain-old data types

Initializers in `if` Statements

Minimize scope of local variables

Initializers in `if` Statements

Minimize scope of local variables

NEW

```
if (auto slash = path.rfind('/'); slash != std::string::npos)
  return {path.substr(0, slash), path.substr(slash + 1)};
```

Initializers in `if` Statements

Minimize scope of local variables

NEW

```
if (auto slash = path.rfind('/'); slash != std::string::npos)
  return {path.substr(0, slash), path.substr(slash + 1)};
```

Initializer scoped to the `if` statement

Initializers in `if` Statements

Minimize scope of local variables

NEW

```
if (auto slash = path.rfind('/'); slash != std::string::npos)
  return {path.substr(0, slash), path.substr(slash + 1)};
```

Initializer scoped to the `if` statement

Condition can reference new local variables

Initializers in `if` Statements

Minimize scope of local variables

NEW

```
if (auto slash = path.rfind('/'); slash != std::string::npos)
  return {path.substr(0, slash), path.substr(slash + 1)};
```

Initializer scoped to the `if` statement

Condition can reference new local variables

Initializers in `if` Statements

Minimize scope of local variables

NEW

```
if (auto slash = path.rfind('/'); slash != std::string::npos)
  return {path.substr(0, slash), path.substr(slash + 1)};
```

Initializer scoped to the `if` statement

Condition can reference new local variables

Same feature works for `switch` statements

Initializers in `if` Statements

Minimize scope of local variables

```
if (auto slash = path.rfind('/'); slash != std::string::npos)
  return {path.substr(0, slash), path.substr(slash + 1)};
```

Initializers in `if` Statements

Minimize scope of local variables

```
if (auto slash = path.rfind('/'); slash != std::string::npos)
  return {path.substr(0, slash), path.substr(slash + 1)};

// Much later, in unrelated code...
if (slash != std::string::npos)
  launchTheSpaceship();
```

Initializers in `if` Statements

Minimize scope of local variables

```
if (auto slash = path.rfind('/'); slash != std::string::npos)
  return {path.substr(0, slash), path.substr(slash + 1)};

// Much later, in unrelated code...
if (slash != std::string::npos)                    Use of undeclared identifier 'slash'
  launchTheSpaceship();
```

Example: Specializing a Generic Algorithm

Advancing an iterator

```
template <class Iterator> Iterator advance(Iterator it, long n) {
  while (n > 0) { ++it; --n; }
  while (n < 0) { --it; ++n; }
  return it;
}
```

Example: Specializing a Generic Algorithm

Advancing an iterator

```
template <class Iterator> Iterator advance(Iterator it, long n) {
  while (n > 0) { ++it; --n; }
  while (n < 0) { --it; ++n; }
  return it;
}
```

Example: Specializing a Generic Algorithm

Advancing an iterator

```
template <class Iterator> Iterator advance(Iterator it, long n) {
  while (n > 0) { ++it; --n; }
  while (n < 0) { --it; ++n; }
  return it;
}
```

Example: Specializing a Generic Algorithm

Advancing an iterator

```
template <class Iterator> Iterator advance(Iterator it, long n) {
  while (n > 0) { ++it; --n; }
  while (n < 0) { --it; ++n; }
  return it;
}
```

Example: Specializing a Generic Algorithm

Advancing an iterator

```
template <class Iterator> Iterator advance(Iterator it, long n) {
  while (n > 0) { ++it; --n; }
  while (n < 0) { --it; ++n; }
  return it;
}
...
auto fifth_node = advance(list.begin(), 5);
```

Example: Specializing a Generic Algorithm

Advancing an iterator

```
template <class Iterator> Iterator advance(Iterator it, long n) {
  while (n > 0) { ++it; --n; }
  while (n < 0) { --it; ++n; }
  return it;
}
...
auto fifth_node = advance(list.begin(), 5);
```

Example: Specializing a Generic Algorithm

Advancing an iterator

```
template <class Iterator> Iterator advance(Iterator it, long n) {
  while (n > 0) { ++it; --n; }
  while (n < 0) { --it; ++n; }
  return it;
}
...
auto fifth_node = advance(list.begin(), 5);
auto fifth_char = advance(string.begin(), 5);
```

Example: Specializing a Generic Algorithm

Advancing an iterator

```
template <class Iterator> Iterator advance(Iterator it, long n) {
  while (n > 0) { ++it; --n; }
  while (n < 0) { --it; ++n; }
  return it;
}
...
auto fifth_node = advance(list.begin(), 5);
auto fifth_char = advance(string.begin(), 5);
```

Example: Specializing a Generic Algorithm

Advancing an iterator

```
template <class Iterator> Iterator advance(Iterator it, long n) {
  if (is_random_access_iterator_v<Iterator>)
    return it + n;
  while (n > 0) { ++it; --n; }
  while (n < 0) { --it; ++n; }
  return it;
}
...
auto fifth_node = advance(list.begin(), 5);
auto fifth_char = advance(string.begin(), 5);
```

Example: Specializing a Generic Algorithm

Advancing an iterator

```
template <class Iterator> Iterator advance(Iterator it, long n) {
  if (is_random_access_iterator_v<Iterator>)
    return it + n;
  while (n > 0) { ++it; --n; }
  while (n < 0) { --it; ++n; }
  return it;
}
...
auto fifth_node = advance(list.begin(), 5);
auto fifth_char = advance(string.begin(), 5);
```

Example: Specializing a Generic Algorithm

Advancing an iterator

```
template <class Iterator> Iterator advance(Iterator it, long n) {
  if (is_random_access_iterator_v<Iterator>)
    return it + n;
  while (n > 0) { ++it; --n; }
  while (n < 0) { --it; ++n; }
  return it;
}
...
auto fifth_node = advance(list.begin(), 5);
auto fifth_char = advance(string.begin(), 5);
```

Example: Specializing a Generic Algorithm

Advancing an iterator

```
template <class Iterator> Iterator advance(Iterator it, long n) {
  if (is_random_access_iterator_v<Iterator>)
    return it + n;
  while (n > 0) { ++it; --n; }
  while (n < 0) { --it; ++n; }
  return it;
}
...
auto fifth_node = advance(list.begin(), 5);
auto fifth_char = advance(string.begin(), 5);
```

Invalid operands to binary expression...

... in instantiation of function template specialization

Example: Specializing a Generic Algorithm

Advancing an iterator

```
  return it + n;



  while (n > 0) { ++it; --n; }
  while (n < 0) { --it; ++n; }
  return it;


template <class Iterator> Iterator advance(Iterator it, long n) {
  return                  (             is_random_access_iterator<Iterator>   )
}
```

Example: Specializing a Generic Algorithm

Advancing an iterator

```
template <class Iterator> Iterator advance_dispatch(Iterator it, long n, true_type) {
  return it + n;
}

template <class Iterator> Iterator advance_dispatch(Iterator it, long n, false_type) {
  while (n > 0) { ++it; --n; }
  while (n < 0) { --it; ++n; }
  return it;
}

template <class Iterator> Iterator advance(Iterator it, long n) {
  return advance_dispatch(first, last, is_random_access_iterator<Iterator>());
}
```

Example: Specializing a Generic Algorithm

Advancing an iterator

```
template <class Iterator> Iterator advance(Iterator it, long n) {
  if (is_random_access_iterator_v<Iterator>)
    return it + n;
  while (n > 0) { ++it; --n; }
  while (n < 0) { --it; ++n; }
  return it;
}
...
auto fifth_node = advance(list.begin(), 5);
auto fifth_char = advance(string.begin(), 5);
```

Invalid operands to binary expression...

... in instantiation of function template specialization

```
template <class Iterator> Iterator advance(Iterator it, long n) {
  if (is_random_access_iterator_v<Iterator>)
    return it + n;
  while (n > 0) { ++it; --n; }
  while (n < 0) { --it; ++n; }
  return it;
}
...
auto fifth_node = advance(list.begin(), 5);
auto fifth_char = advance(string.begin(), 5);
```

Invalid operands to binary expression...

... in instantiation of function template specialization

`constexpr` Evaluation of `if` Statements

An alternative to compile-time dispatch

NEW

```
template <class Iterator> Iterator advance(Iterator it, long n) {
  if constexpr (is_random_access_iterator_v<Iterator>)
    return it + n;
  while (n > 0) { ++it; --n; }
  while (n < 0) { --it; ++n; }
  return it;
}
...
auto fifth_node = advance(list.begin(), 5);
auto fifth_char = advance(string.begin(), 5);
```

Invalid operands to binary expression...

... in instantiation of function template specialization

`constexpr` Evaluation of `if` Statements

An alternative to compile-time dispatch

NEW

```
template <class Iterator> Iterator advance(Iterator it, long n) {
  if constexpr (is_random_access_iterator_v<Iterator>)
    return it + n;
  while (n > 0) { ++it; --n; }
  while (n < 0) { --it; ++n; }
  return it;
}
...
auto fifth_node = advance(list.begin(), 5);
auto fifth_char = advance(string.begin(), 5);
```

`constexpr` Evaluation of `if` Statements

An alternative to compile-time dispatch

NEW

```
template <class Iterator> Iterator advance(Iterator it, long n) {
  if constexpr (is_random_access_iterator_v<Iterator>)
    return it + n;
  while (n > 0) { ++it; --n; }
  while (n < 0) { --it; ++n; }
  return it;
}
...
auto fifth_node = advance(list.begin(), 5);
auto fifth_char = advance(string.begin(), 5);
```

`constexpr` Evaluation of `if` Statements

An alternative to compile-time dispatch

NEW

```
template <class Iterator> Iterator advance(Iterator it, long n) {
  if constexpr (is_random_access_iterator_v<Iterator>)
    return it + n;
  while (n > 0) { ++it; --n; }
  while (n < 0) { --it; ++n; }
  return it;
}
...
auto fifth_node = advance(list.begin(), 5);
auto fifth_char = advance(string.begin(), 5);
```

`constexpr` Evaluation of `if` Statements

An alternative to compile-time dispatch

NEW

```
template <class Iterator> Iterator advance(Iterator it, long n) {
  if constexpr (is_random_access_iterator_v<Iterator>)
    return it + n;
  while (n > 0) { ++it; --n; }
  while (n < 0) { --it; ++n; }
  return it;
}
...
auto fifth_node = advance(list.begin(), 5);
auto fifth_char = advance(string.begin(), 5);
```

Problems with `std::string`-based APIs

Problems with `std::string`-based APIs

```
#include <string>
std::pair<std::string, std::string> split(const std::string &path) {
  if (auto slash = path.rfind('/'); slash != std::string::npos)
    return {path.substr(0, slash), path.substr(slash + 1)};
  return {std::string{}, path};
}
```

Problems with `std::string`-based APIs

```
#include <string>
std::pair<std::string, std::string> split(const std::string &path) {
  if (auto slash = path.rfind('/'); slash != std::string::npos)
    return {path.substr(0, slash), path.substr(slash + 1)};
  return {std::string{}, path};
}
```

Problems with `std::string`-based APIs

```
#include <string>
std::pair<std::string, std::string> split(const std::string &path) {
  if (auto slash = path.rfind('/'); slash != std::string::npos)
    return {path.substr(0, slash), path.substr(slash + 1)};
  return {std::string{}, path};
}
```

Problems with `std::string`-based APIs

```
#include <string>
std::pair<std::string, std::string> split(const std::string &path) {
  if (auto slash = path.rfind('/'); slash != std::string::npos)
    return {path.substr(0, slash), path.substr(slash + 1)};
  return {std::string{}, path};
}
```

Problems with `std::string`-based APIs

```
#include <string>
std::pair<std::string, std::string> split(const std::string &path) {
  if (auto slash = path.rfind('/'); slash != std::string::npos)
    return {path.substr(0, slash), path.substr(slash + 1)};
  return {std::string{}, path};
}
```

Substrings are copies and can be expensive

```cpp
#include <string>
std::pair<std::string, std::string> split(const std::string &path) {
  if (auto slash = path.rfind('/'); slash != std::string::npos)
    return {path.substr(0, slash), path.substr(slash + 1)};
  return {std::string{}, path};
}
```

```cpp
#include <string_view>
std::pair<std::string_view, std::string_view> split(std::string_view path) {
  if (auto slash = path.rfind('/'); slash != std::string_view::npos)
    return {path.substr(0, slash), path.substr(slash + 1)};
  return {std::string_view{}, path};
}
```

std::string_view

Reference a string without ownership

NEW

```
#include <string_view>
std::pair<std::string_view, std::string_view> split(std::string_view path) {
  if (auto slash = path.rfind('/'); slash != std::string_view::npos)
    return {path.substr(0, slash), path.substr(slash + 1)};
  return {std::string_view{}, path};
}
```

`std::string_view`

Reference a string without ownership

NEW

```
#include <string_view>
std::pair<std::string_view, std::string_view> split(std::string_view path) {
  if (auto slash = path.rfind('/'); slash != std::string_view::npos)
    return {path.substr(0, slash), path.substr(slash + 1)};
  return {std::string_view{}, path};
}
```

Encapsulates a `const char *` and a `size_t`

`std::string_view`

Reference a string without ownership

NEW

```
#include <string_view>
std::pair<std::string_view, std::string_view> split(std::string_view path) {
  if (auto slash = path.rfind('/'); slash != std::string_view::npos)
    return {path.substr(0, slash), path.substr(slash + 1)};
  return {std::string_view{}, path};
}
```

Encapsulates a `const char *` and a `size_t`

Rich string API, like `std::string`

`std::string_view`

Reference a string without ownership

NEW

```
#include <string_view>
std::pair<std::string_view, std::string_view> split(std::string_view path) {
  if (auto slash = path.rfind('/'); slash != std::string_view::npos)
    return {path.substr(0, slash), path.substr(slash + 1)};
  return {std::string_view{}, path};
}
```

Encapsulates a `const char *` and a `size_t`

Rich string API, like `std::string`

Never copies the string

Lifetime of a String Reference

A `string_view` has the lifetime of its source

Lifetime of a String Reference

A `string_view` has the lifetime of its source

Raw string literals are always safe

```
std::string_view getImagesPath() { return "resources/images"; }
```

Lifetime of a String Reference

A `string_view` has the lifetime of its source

Raw string literals are always safe

```
std::string_view getImagesPath() { return "resources/images"; }
```

Lifetime of a String Reference

A `string_view` has the lifetime of its source

Raw string literals are always safe

Arguments are safe if not stored

```
std::error_code open_file(std::string_view filename);
bool parse(std::string_view data);
bool parse_header(std::string_view header);
bool parse_body(std::string_view body);
bool parse_token(std::string_view token);
```

Lifetime of a String Reference

A `string_view` has the lifetime of its source

Raw string literals are always safe

Arguments are safe if not stored

Return values are usually safe when derived from arguments

```
std::pair<std::string_view, std::string_view> split(std::string_view path);
...
auto [directory, filename] = split(path);
```

Lifetime of a String Reference

A `string_view` has the lifetime of its source

Raw string literals are always safe

Arguments are safe if not stored

Return values are usually safe when derived from arguments

```
std::pair<std::string_view, std::string_view> split(std::string_view path);
...
auto [directory, filename] = split(path);
```

Lifetime of a String Reference

A `string_view` has the lifetime of its source

Raw string literals are always safe

Arguments are safe if not stored

Return values are usually safe when derived from arguments

```
std::pair<std::string_view, std::string_view> split(std::string_view path);
...
auto [directory, filename] = split(pwd + "/" + path);
auto dot = filename.rfind('.');
```

Lifetime of a String Reference

A `string_view` has the lifetime of its source

Raw string literals are always safe

Arguments are safe if not stored

Return values are usually safe when derived from arguments

```
std::pair<std::string_view, std::string_view> split(std::string_view path);
...
auto [directory, filename] = split(pwd + "/" + path);
auto dot = filename.rfind('.');
```

Lifetime of a String Reference

A `string_view` has the lifetime of its source

Raw string literals are always safe

Arguments are safe if not stored

Return values are usually safe when derived from arguments

```
std::pair<std::string_view, std::string_view> split(std::string_view path);
...
auto [directory, filename] = split(pwd + "/" + path);
auto dot = filename.rfind('.');
```

Lifetime of a String Reference

A `string_view` has the lifetime of its source

Raw string literals are always safe

Arguments are safe if not stored

Return values are usually safe when derived from arguments

```
std::pair<std::string_view, std::string_view> split(std::string_view path);
...
auto [directory, filename] = split(pwd + "/" + path);
auto dot = filename.rfind('.');
```

Address Sanitizer: Use of deallocated memory

Lifetime of a String Reference

A `string_view` has the lifetime of its source

Raw string literals are always safe

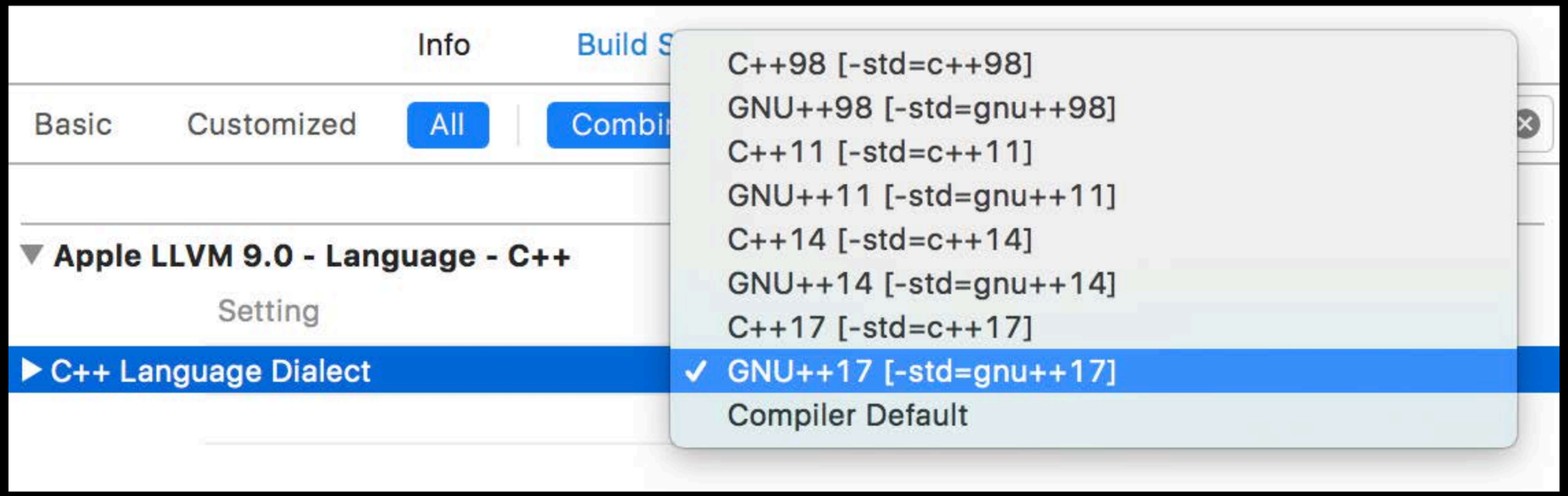
Arguments are safe if not stored

Return values are usually safe when derived from arguments

Finding Bugs Using Xcode Runtime Tools	WWDC 2017
Understanding Undefined Behavior	WWDC 2017

Build Setting: C++ Language Dialect

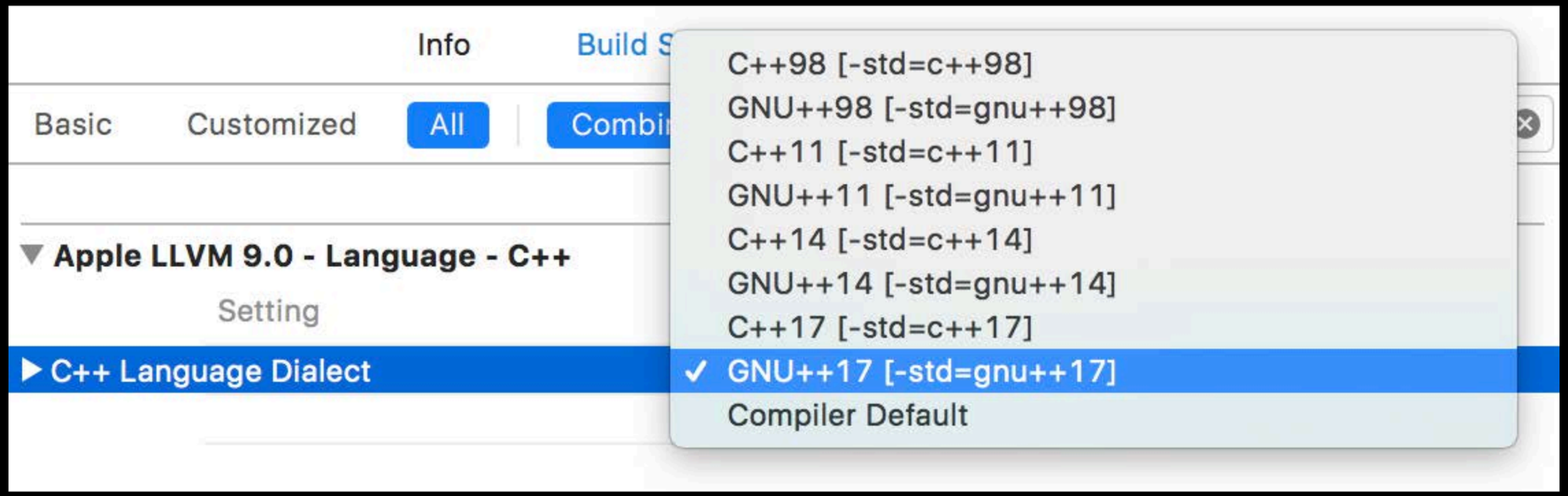
Use C++17 or GNU++17 to try out the new features



Build Setting: C++ Language Dialect

Use C++17 or GNU++17 to try out the new features

C++17 *C++17 without language extensions*

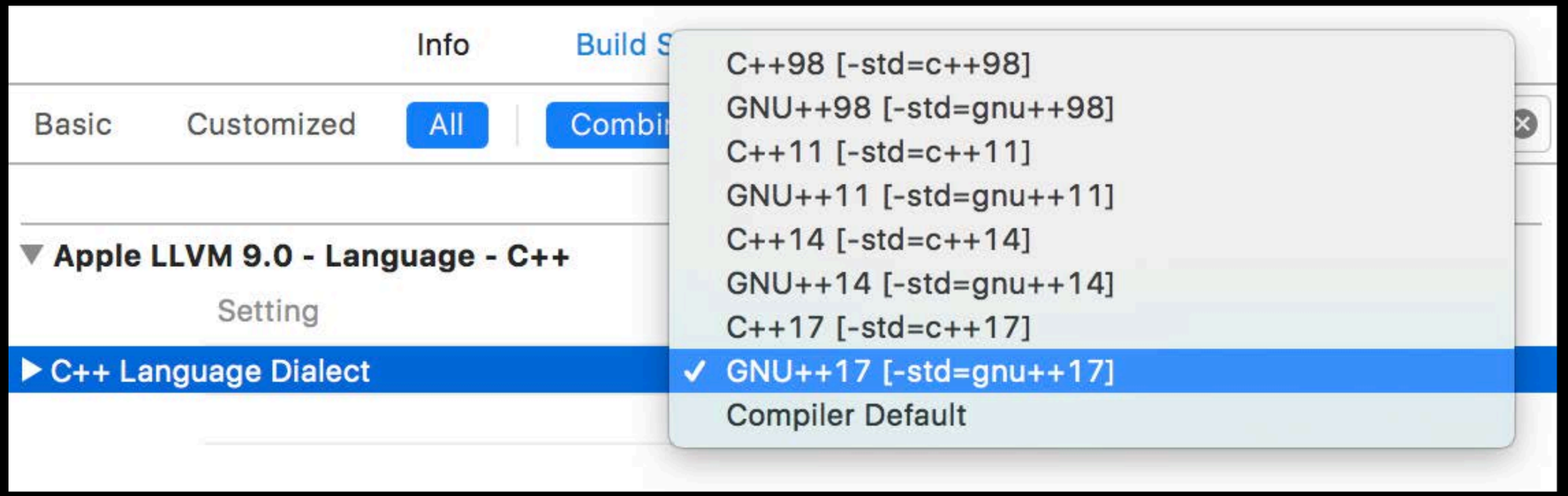


Build Setting: C++ Language Dialect

Use C++17 or GNU++17 to try out the new features

C++17 *C++17 without language extensions*

GNU++17 *C++17 with language extensions*



Link-Time Optimization

What Is Link-Time Optimization (LTO)?

What Is Link-Time Optimization (LTO)?

Upgrade runtime performance by optimizing across source files

What Is Link-Time Optimization (LTO)?

Upgrade runtime performance by optimizing across source files

Incremental LTO has low overhead and fast incremental builds

What Is Link-Time Optimization (LTO)?

Upgrade runtime performance by optimizing across source files

Incremental LTO has low overhead and fast incremental builds

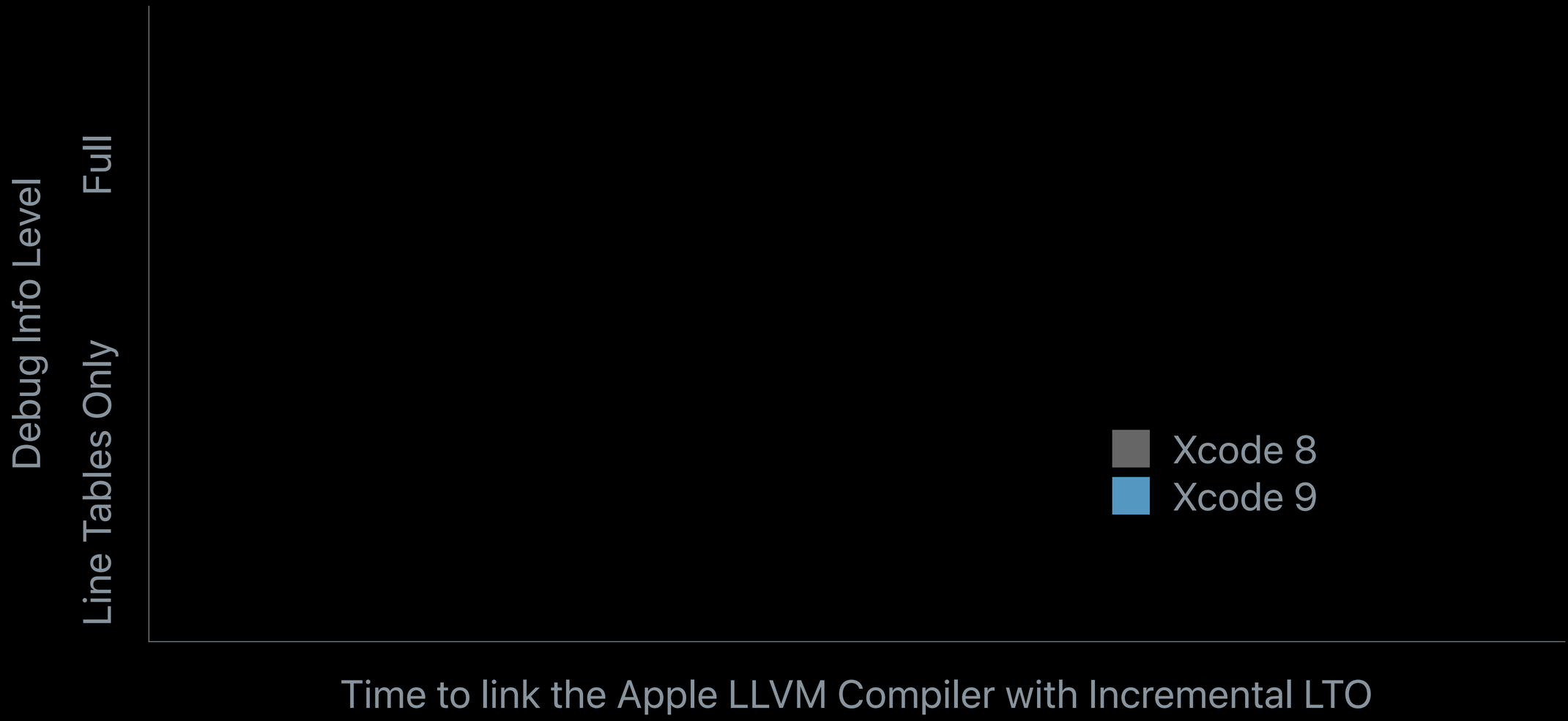
What Is Link-Time Optimization (LTO)?

Upgrade runtime performance by optimizing across source files

Incremental LTO has low overhead and fast incremental builds

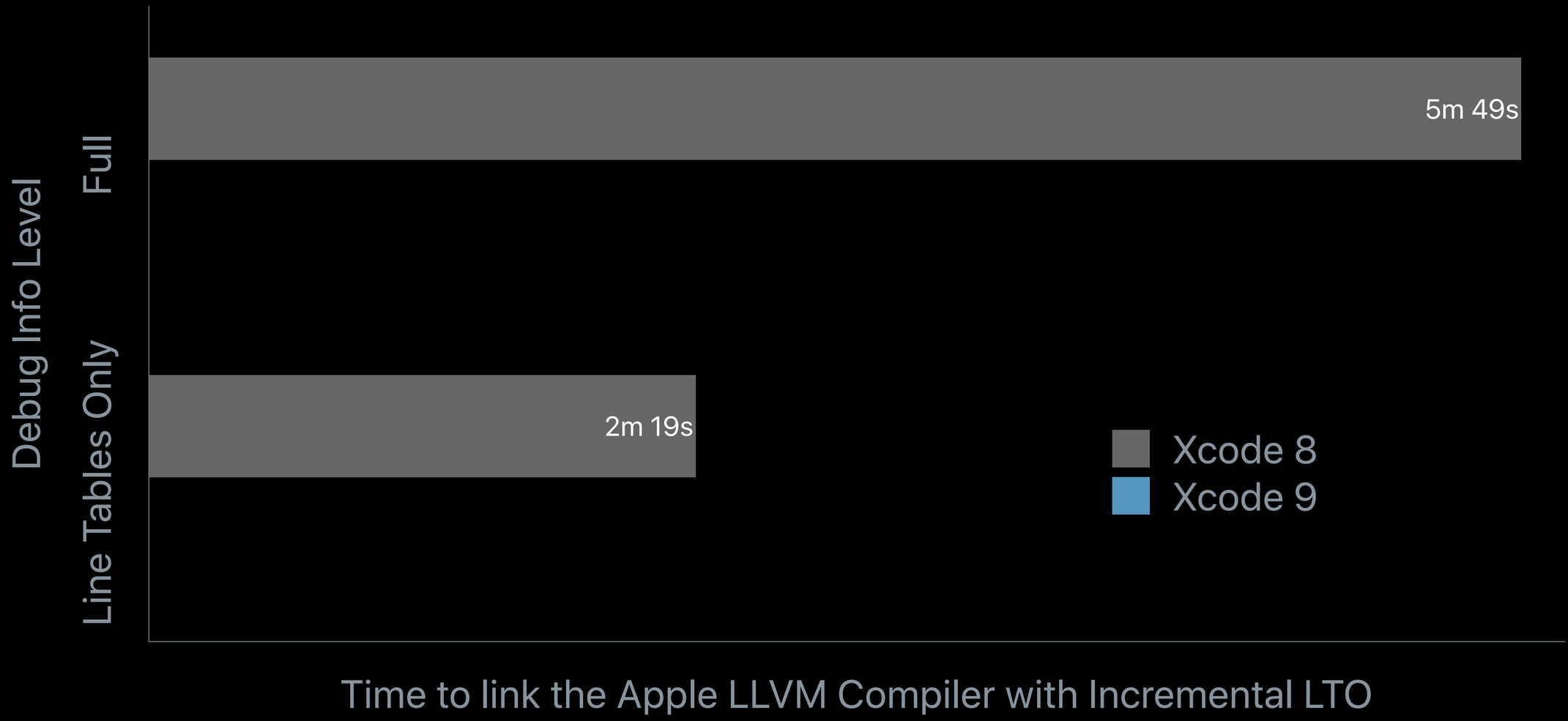
Time for Clean Link of a Large C++ Project

Smaller is better



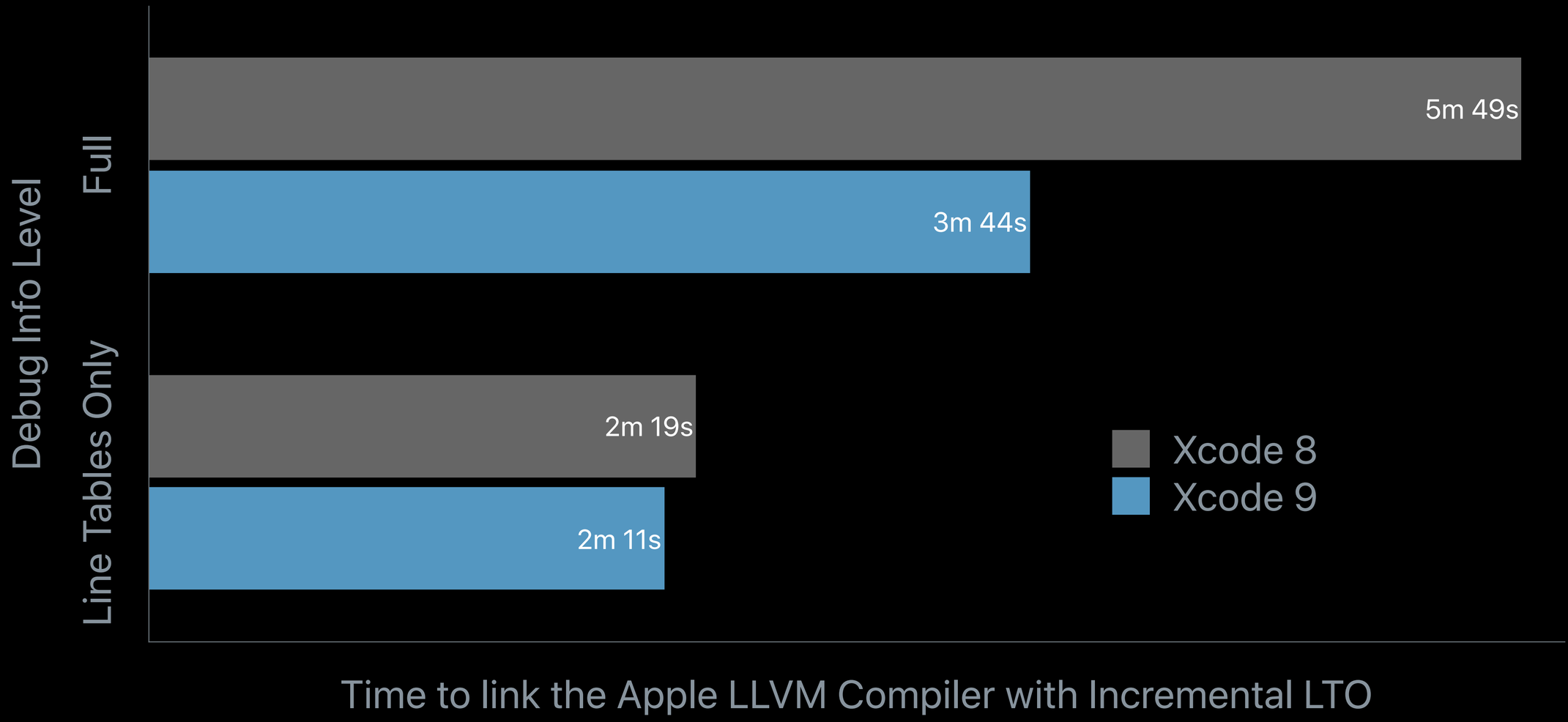
Time for Clean Link of a Large C++ Project

Smaller is better



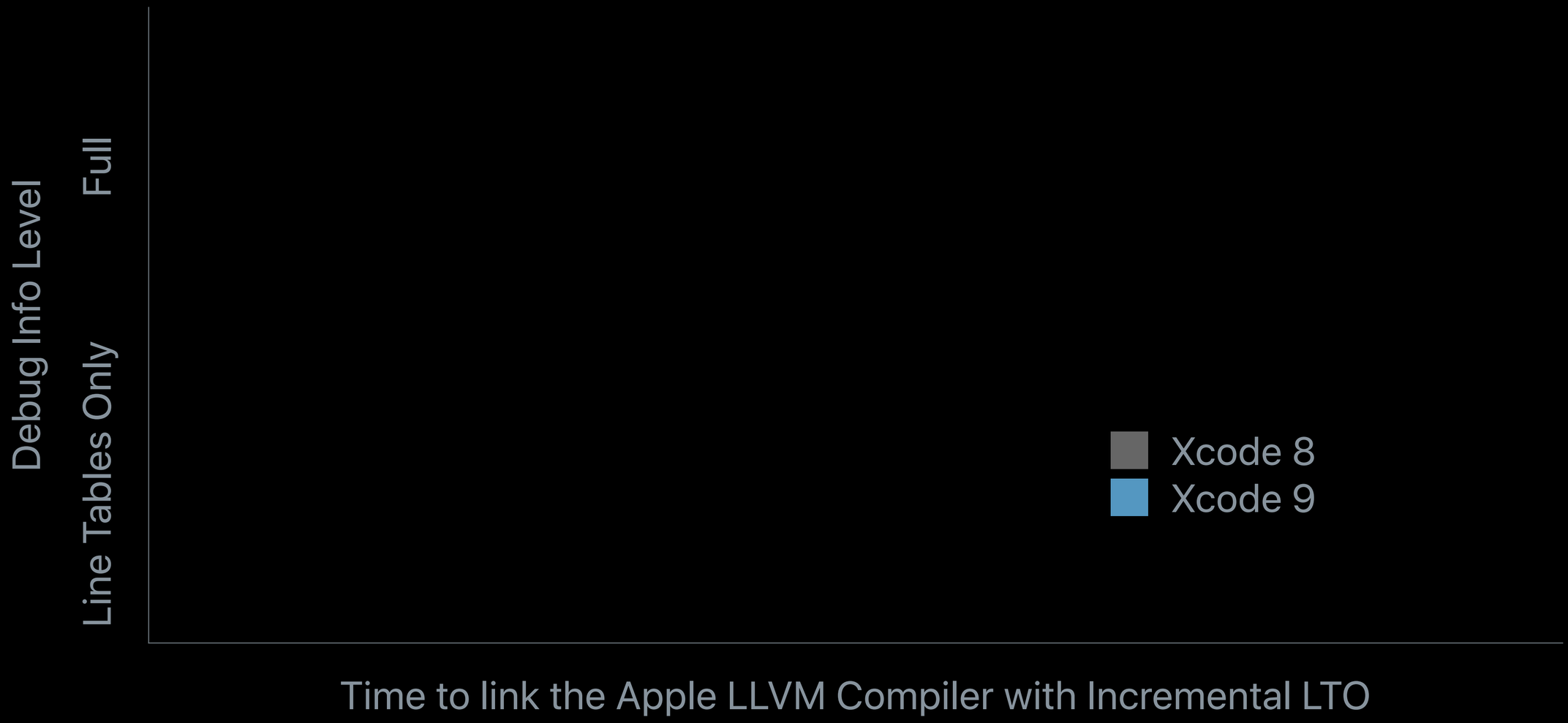
Time for Clean Link of a Large C++ Project

Smaller is better



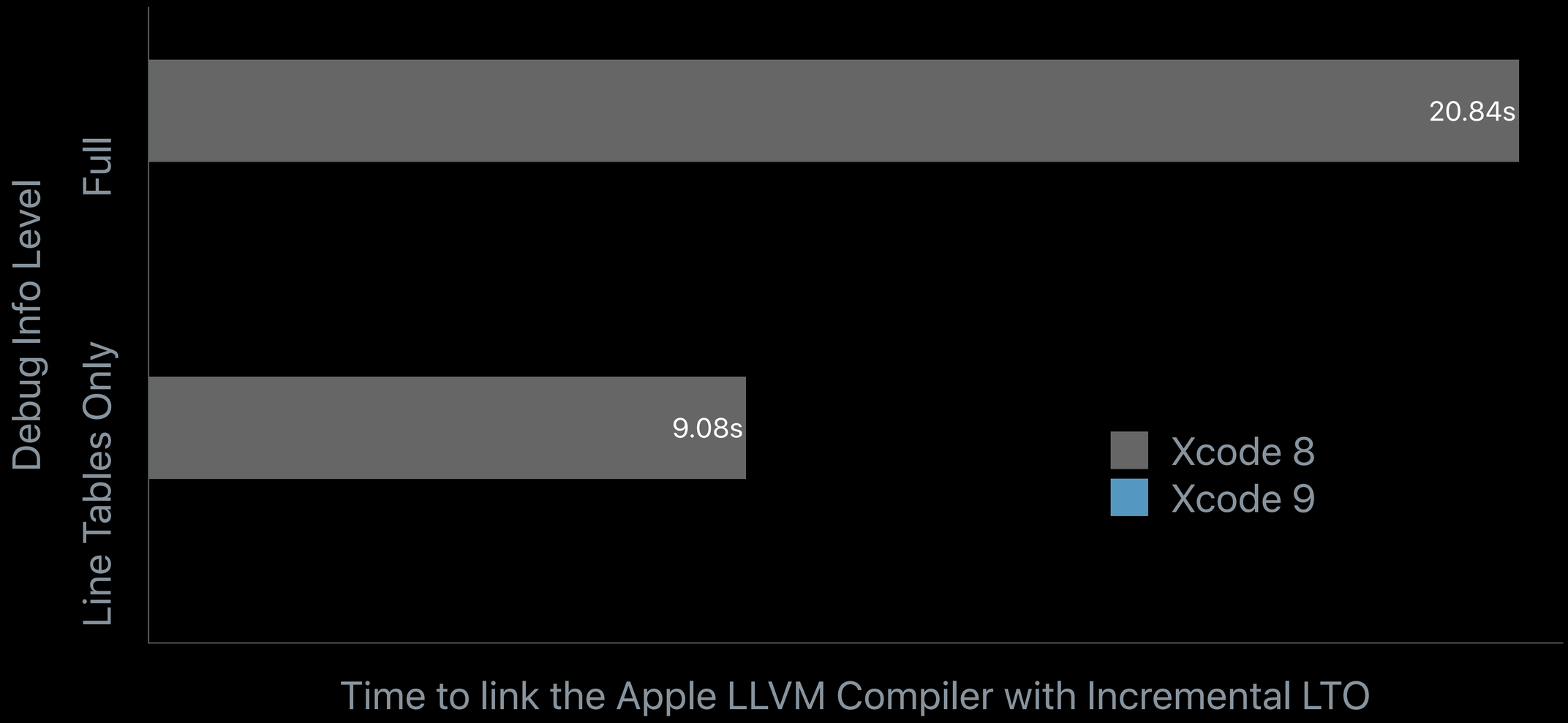
Time for Incremental Link of a Large C++ Project

Smaller is better



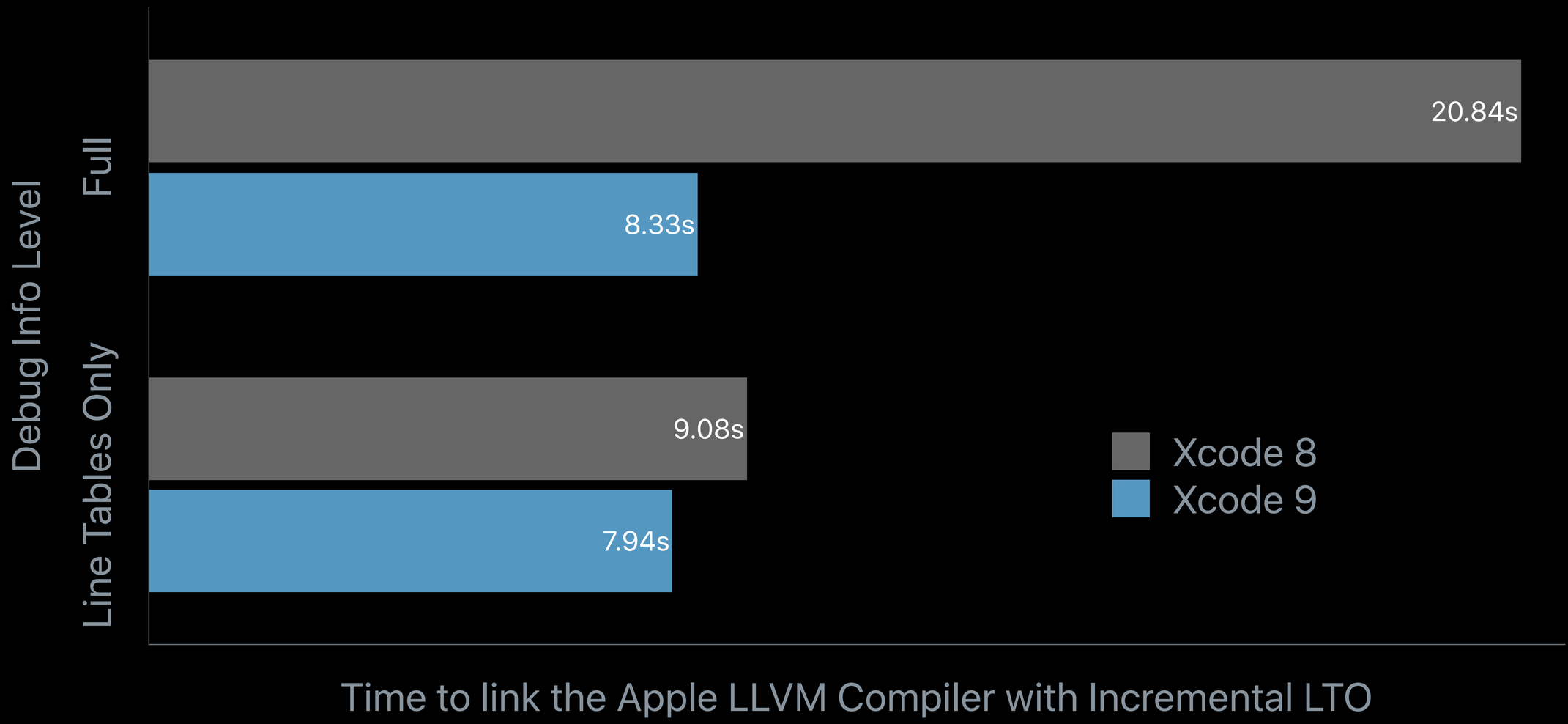
Time for Incremental Link of a Large C++ Project

Smaller is better



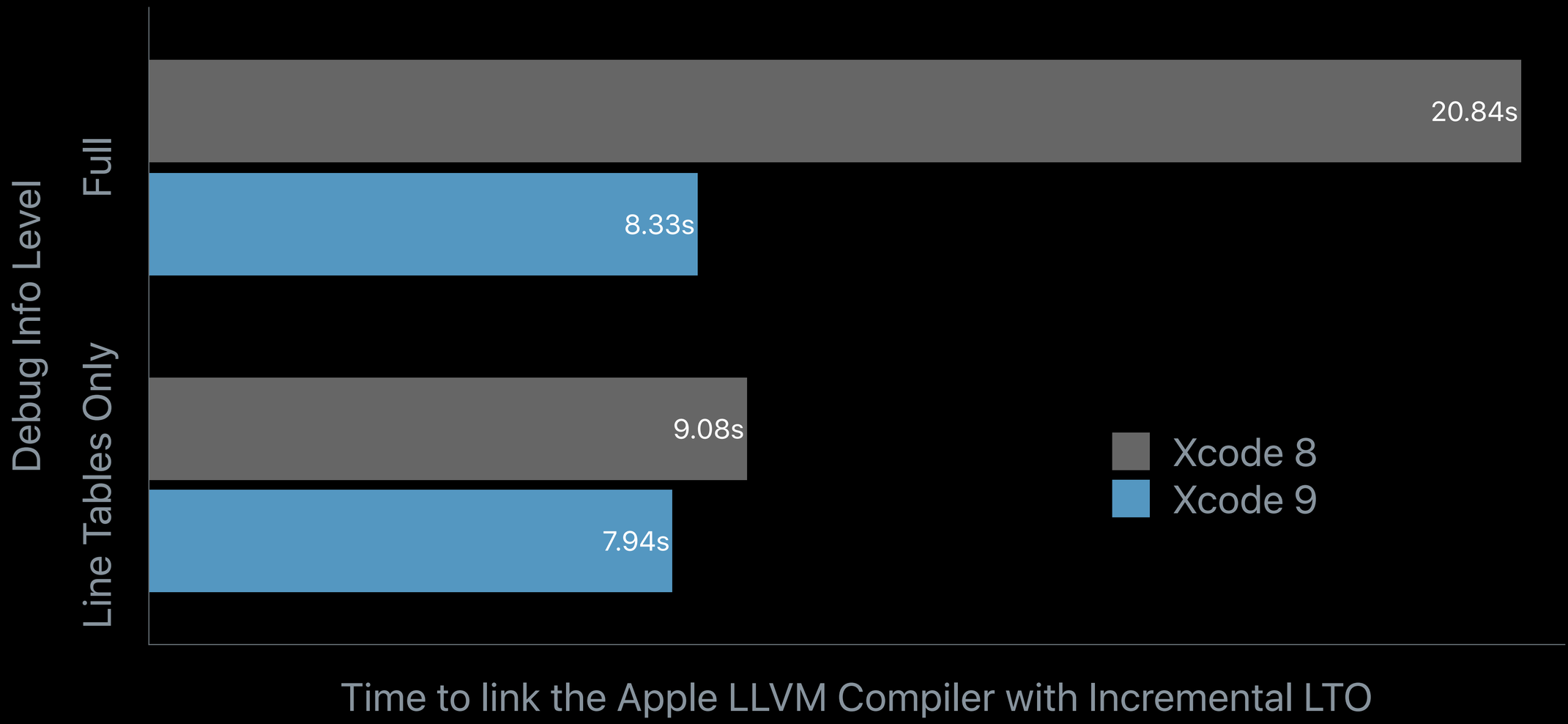
Time for Incremental Link of a Large C++ Project

Smaller is better



Time for Incremental Link of a Large C++ Project

Smaller is better



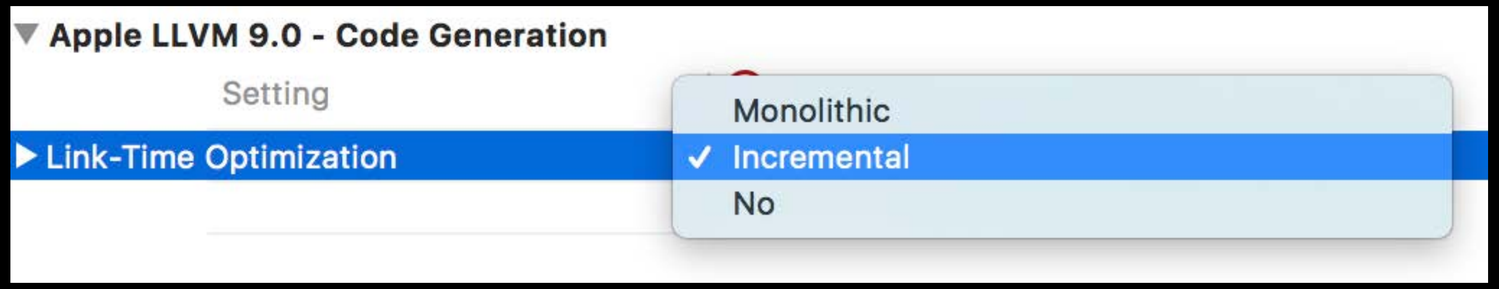
Enable Incremental LTO

Recommendation

Upgrade runtime performance

Low overhead and fast incremental builds

Now recommended even with full debug info



Summary

Use `@available` to check for API availability

Run the static analyzer while you build

Use Xcode to refactor your code

Try out C++17

Enable Incremental LTO

More Information

https://developer.apple.com/wwdc17/411

Related Sessions

What's New in Swift		WWDC 2017
Modernizing Grand Central Dispatch Usage		WWDC 2017
Finding Bugs Using Xcode Runtime Tools		WWDC 2017
Understanding Undefined Behavior		WWDC 2017
App Startup Time: Past, Present, and Future	Hall 2	Friday 10:00AM

Labs

LLVM Compiler, Objective-C, and C++ Lab	Technology Lab E	Friday 9:00AM–11:00AM
Xcode Open Hours	Technology Lab K	Friday 1:50PM–4:00PM


WWDC17