Great Programming with

ImportC

by Walter Bright
Dlang.org

D is designed to be easy to interface to C

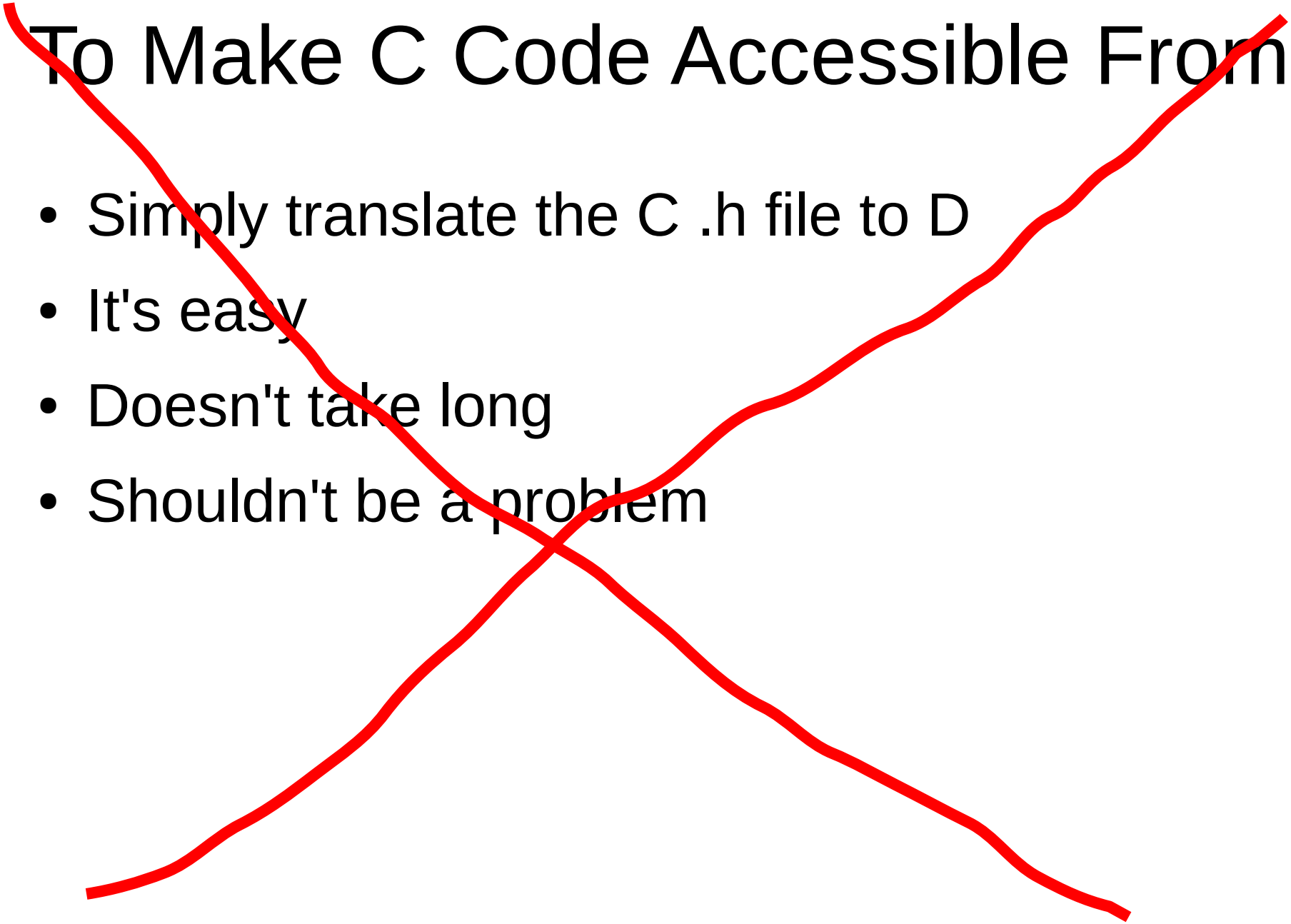
- Zero cost
- Compatible types
- Familiar syntax
- Compatible semantics

To Make C Code Accessible From D

- Simply translate the C .h file to D
- It's easy
- Doesn't take long
- Shouldn't be a problem

To Make C Code Accessible From D

- Simply translate the C .h file to D
- It's easy
- Doesn't take long
- Shouldn't be a problem



How Wrong Can We Be

- Preprocessor macros obfuscate the code
- Mass of #includes can explode to 10,000 lines or more
- Need for competence in both C and D
- No reliable way to verify correctness of translation
- Tedious

Continued...

- Cannot just write one's own system .h files
- Portability problems
  - A C long can be a D int or a D long
  - char can be signed or unsigned
  - Bitfields
  - alignment
- .h files change over time

Solutions

- We provide a subset of the system C .h files already carefully translated to D
- The Diemos project is a crowdsourced repository where people share the work the translate .h files
  - Of course that doesn't work for proprietary C code

Experience shows
it just isn't good enough and is a significant barrier

Enter The .h To .d Translator

- Such a great idea, we wrote three of them:
  - htod by Walter Bright (that's me)
  - DStep by Jacob Carlborg
  - dpp by Atila Neves

Much improves the situation, but:

- There's always a “but”
- Clumsy to set up with the build system
- Creates extra files
- Has difficulties translating things like bit fields
- Friction

What would be the simplest, most obvious, most perfect way for D code to simply get all the declarations from, say, stdio.h?

```
import stdio;
```

Once you've seen that, you can't unsee it. That is what the user experience has to be.

it must *just work*

What's the obvious way to make that just work?

Incorporate a freakin' real live actual honest-to-god C compiler into the D compiler!

This idea has come up before, but of course it is an eeediotic idea only a naïve madman would propose.

Buuuuuuuuttt, maybe we shouldn't be so hasty. C is a simple language. I've written a C compiler. I could probably whip one out in a weekend, right?

And thus

ImportC

was conceived!

(cue dramatic music)

First Gigantic Problem

- The preprocessor
  - Preprocessor metaprogramming
  - D has no analog for text macros
  - Preprocessing has lots of switches for it
  - Just no way to reliably deal with all the nutburger use of C macros out there in the wild
- This problem has stymied us for years
- But there is a solution staring us in the face

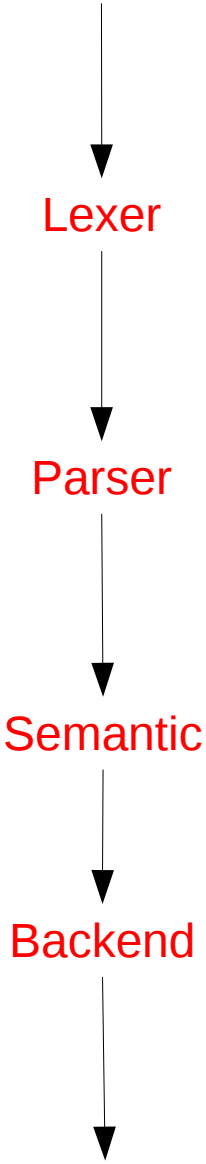
Abandon the Preprocessor!

- Have ImportC only work on preprocessed files
- It all fits in one file
- C preprocessors already exist as standalone programs
  - Don't have to worry about getting it right
- Only the macros are of interest to D
  - Dpp has shown they can be handled in an ad-hoc manner
  - But for now we'll just not be concerned about it

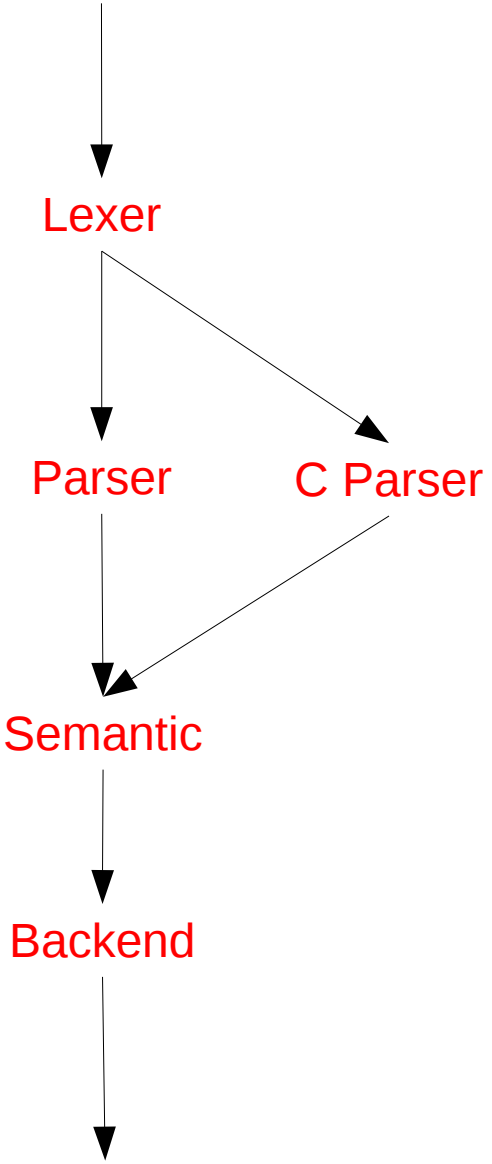
Next Problems: need a C

- Lexer
- Parser
- Semantic analysis
- Code generation

Layout of the D compiler



With ImportC



Have to tweak the lexer and semantic, but it can work

Ground Rules

- C11 is baseline
- No implicit function declarations
- No warnings
- No printf checking
- Not fixing C
- Minimal C extensions
- No zoo of compiler switches
  - It should “just work”

Lexing differences

- Different keywords
  - Signed, unsigned, register, inline, typedef, _Static_assert, restrict, volatile, _Alignas, _Alignof, _Atomic, _Bool, _Complex, _Imaginary, _Noreturn, _Thread_local
- Numeric literals
- #pragma

C parsing is just not that complicated

- Same old recursive descent
- Needs arbitrary lookahead
- Cannot use symbol table to disambiguate parse
  - C is not designed to separate parsing from semantic
  - But we're going to do it!

Ambiguous Syntax

(A)(B)

Cast or function call? Cannot determine that without knowing if A is a function or a type.

Turn the construct into a special AST node, which is then rewritten by the semantic pass into a cast expression or a function call AST.

One Big Simplification

- C doesn't have modules. It's just one big completely self-contained file.
  - After preprocessing, of course!

Some constructs do not exist in D

- - >
- _Generic
- (type-name) { initializer-list }
- Add new AST nodes for them, and add semantic routines to rewrite them into D AST nodes

Bit Fields

- Cannot determine them at parse time
  - So make an AST node for them
  - Code generator already works with bit fields
- Underdocumented
- But – worked out very nicely
  - Considering adding it to D

Old-Style Function Declarations

```
int foo(a, b)
int a;
double d;
{
    ...
}
```

Very Different Static Initializers

- Add special AST node for them
  - Again, cannot determine their shape in the parse pass
- Translate them into D static initializers in semantic pass

#pragma pack

- Non-standard
- Under-documented
- Kludgey
- Had to implement it anyway, as too much existing code used it

```
int
#pragma pack(8)
    x
#pragma pack()
    ;
```

Tag Name Space

```
struct S { … };
int S;
```

Using two parallel tables consumes too much time and space, so used separate hash table for the tag names.

Advantages (Enhancements?)

```
int x = square(2);
int square(int i) {
    return i * i;
}
```

forward references and
compile time function execution

Problems

- Don't have a C test suite
  - Don't really like the old C one I have
- Const is transitive in D semantics
  - Too hard to change type system for C semantics
  - T *const p; is const pointer to mutable in C, const pointer to const in ImportC
  - Surprisingly, this hasn't caused trouble with D interfacing to C, it seems the D semantics are how people naturally use const

Additional Uses

- Use it as a fast C compiler
- D compiler “dogfoods” C libraries
- Convenient when needing a bit of C code to interface
- Convenient when mixing and matching C and D modules

Conclusion

- Transformative in ease of interfacing to C
- No need to translate .h files anymore
- Immunized from changes in .h files
- Available now as beta

References

- https://dlang.org/spec/importc.html