# D features: Complexity vs. Benefits

Răzvan Nițu
Dconf 2024

## About me

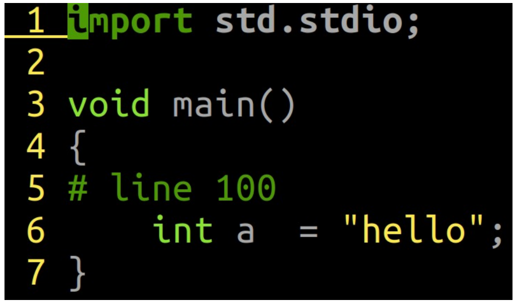
- Contributing since 2016
- PR & Issue Manager since 2021
- PhD graduate 2023
- Associate Teaching Assistant At UPB
- Software Developer at Chenope

## Introduction

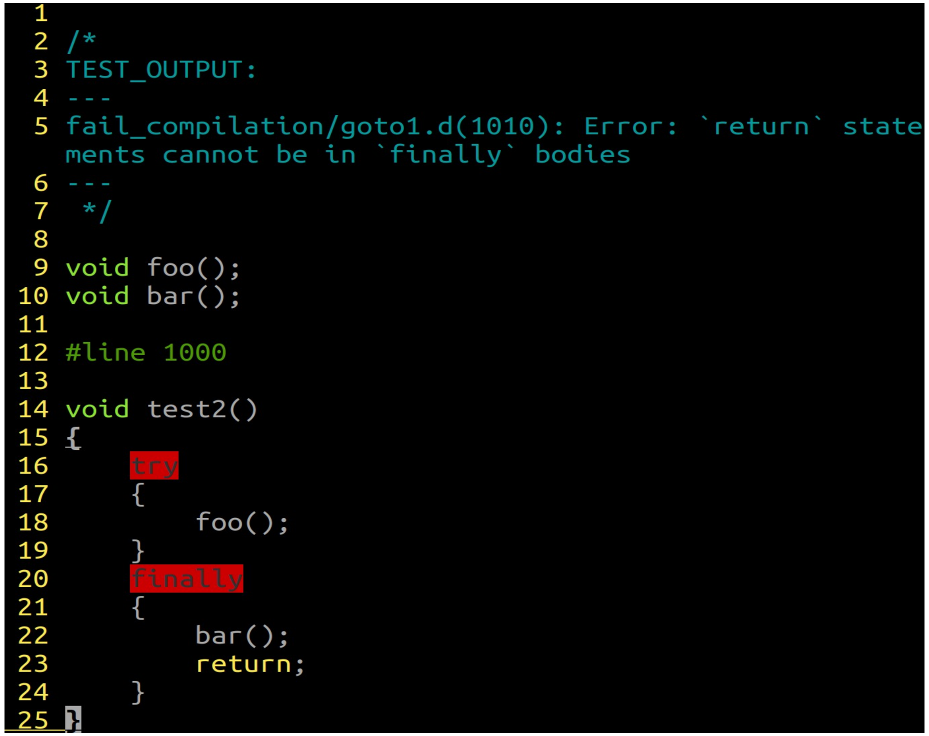
- Every line of code adds complexity
- The complexity is justified if there is benefit (RoI)
- R < I

## #line directive

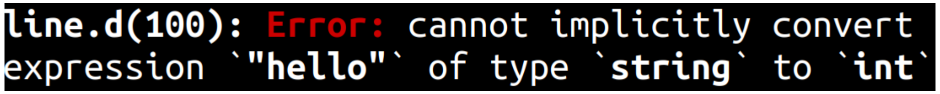
# #line directive



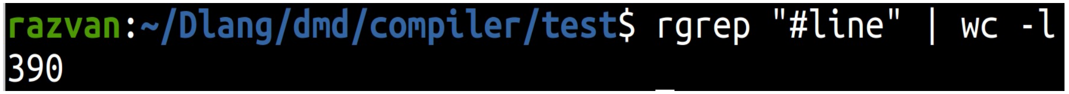
```d
import std.stdio;

void main()
{
# line 100
     int a  = "hello";
}
```



```d

/*
TEST_OUTPUT:
---
fail_compilation/goto1.d(1010): Error: `return` statements cannot be in `finally` bodies
---
 */

void foo();
void bar();

#line 1000

void test2()
{
    try
    {
        foo();
    }
    finally
    {
        bar();
        return;
    }
}
```



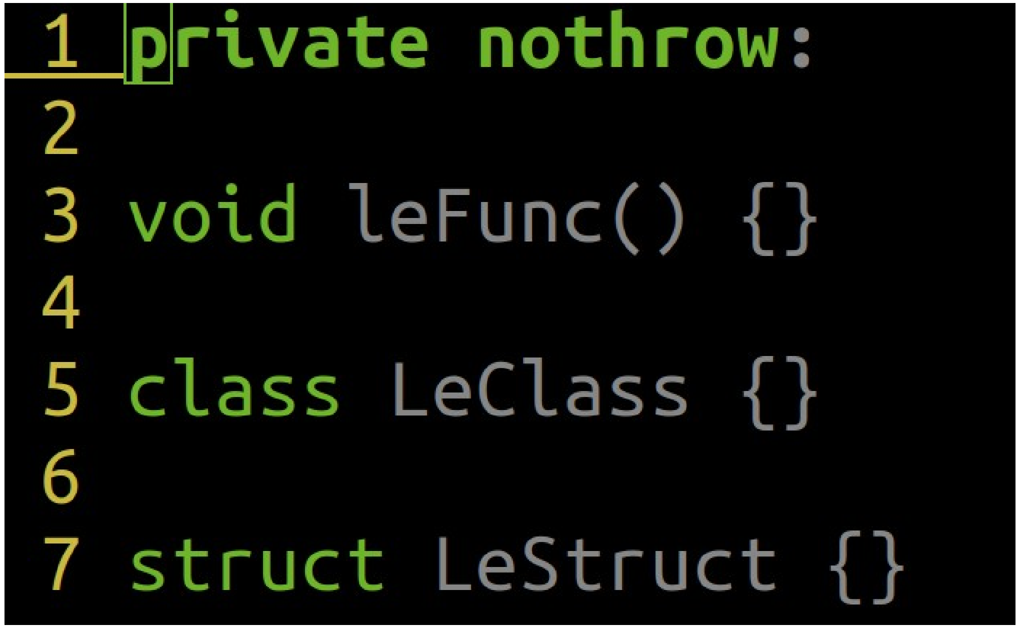
```
line.d(100): Error: cannot implicitly convert
expression `"hello"` of type `string` to `int`
```



```
razvan:~/Dlang/dmd/compiler/test$ rgrep "#line" | wc -l
390
```

## `En-masse` attributes

## `En-masse` attributes

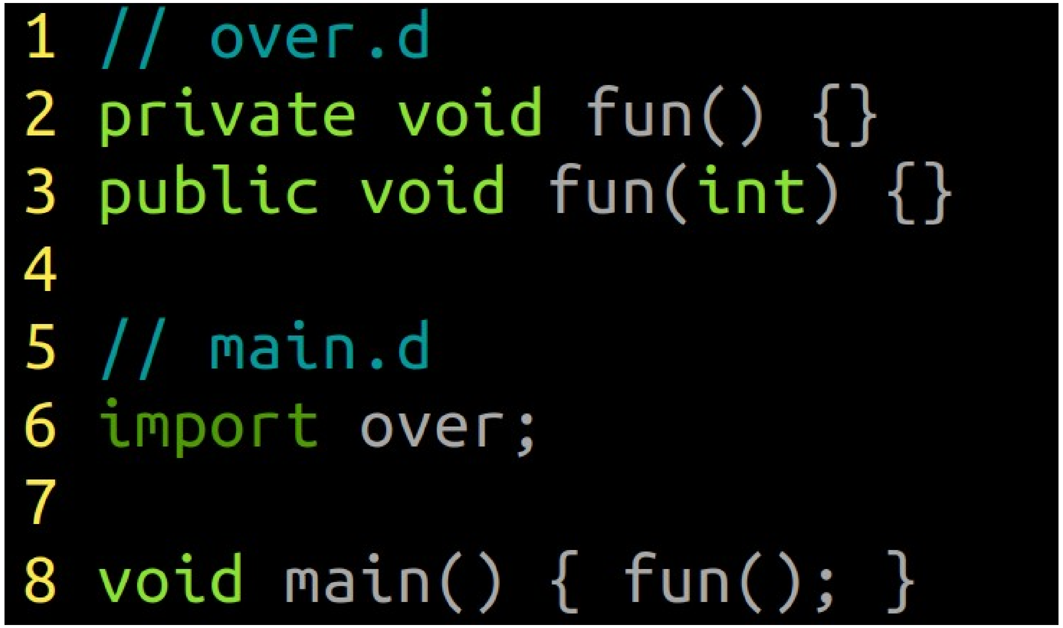


```d
private nothrow:

void leFunc() {}

class LeClass {}

struct LeStruct {}
```

‘You may not like to wipe your ass, but you just gotta do it’ – Atila Neves, Dconf 2023

## Public+Private Overloads

## Public+Private overloads

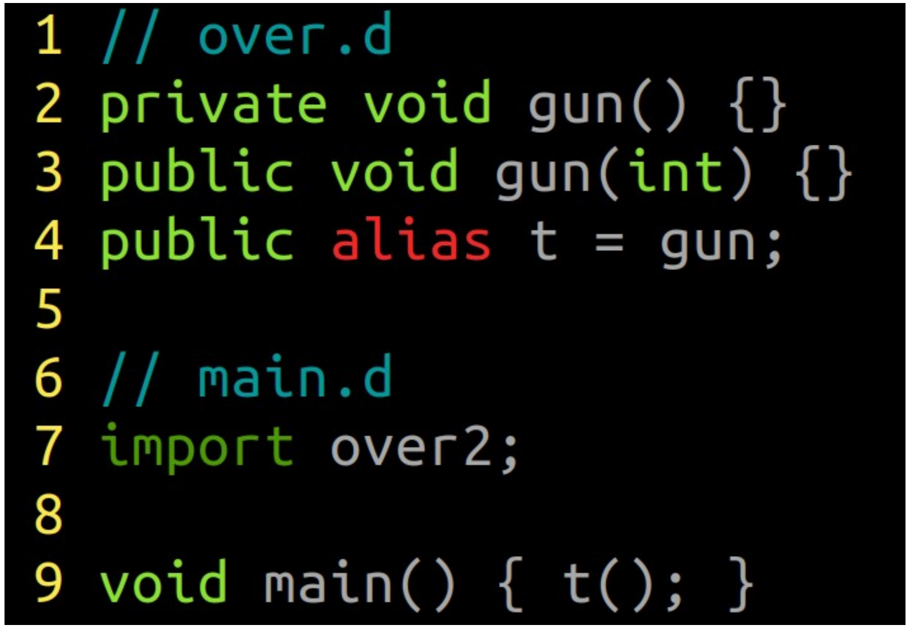


```d
// over.d
private void fun() {}
public void fun(int) {}

// main.d
import over;

void main() { fun(); }
```

## Public+Private Overloads

- Compiler does:
  - Search for symbol
  - If symbol is an overload set, get most visible overload
  - Match the function call
  - Check access specifier

## Public+Private Overloads



```
// over.d
private void gun() {}
public void gun(int) {}
public alias t = gun;

// main.d
import over2;

void main() { t(); }
```

## Public+Private Overloads

- Just don’t allow overloads with different access specifiers
  - No need for extra checks to fix the initial issue
  - Public aliases to private symbols still work
  - Breaks code

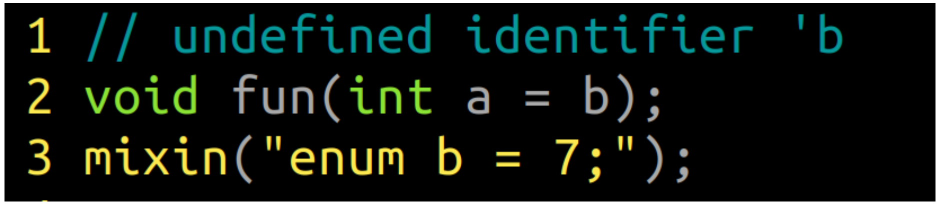
## Mixins

## Mixins

- std.bitmanip example

## Mixins

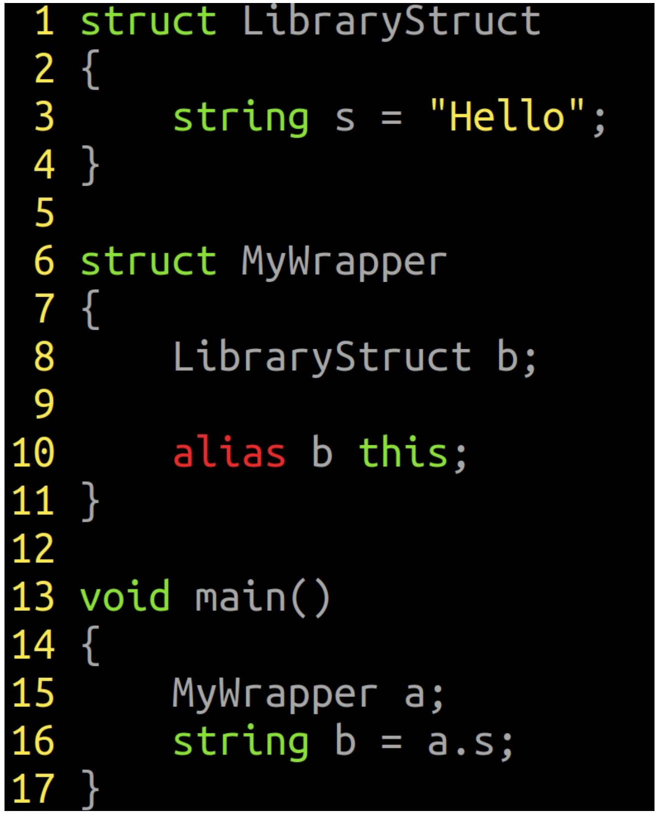
- Powerful, but a nightmare to maintain
- Add complexity to the codebase
- Issues with forward declarations (~234 bugzilla)



```
// undefined identifier 'b
void fun(int a = b);
mixin("enum b = 7;");
```

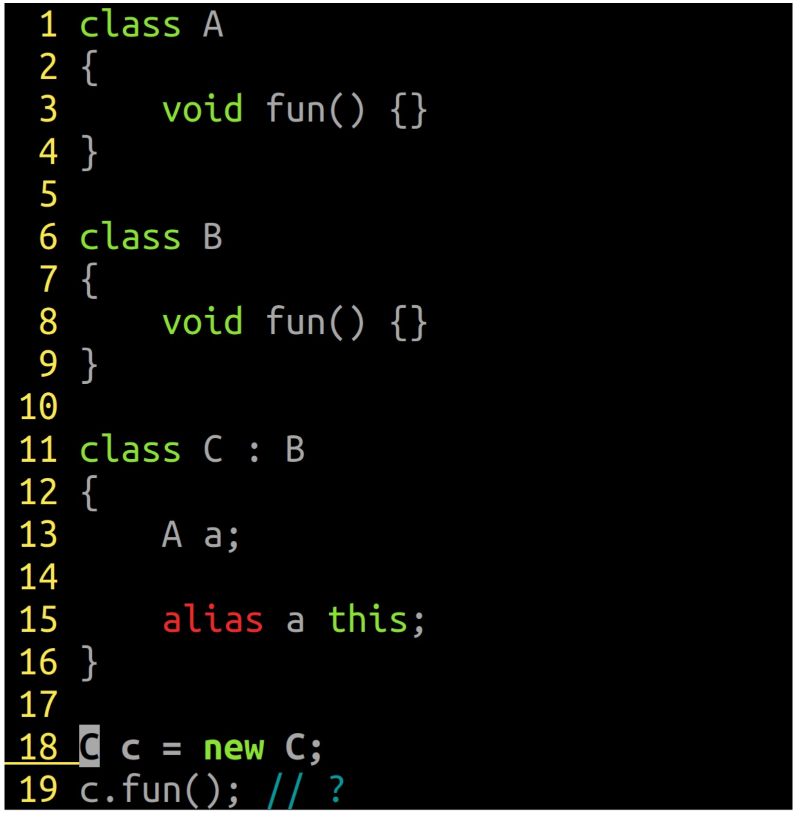
## alias this

## alias this



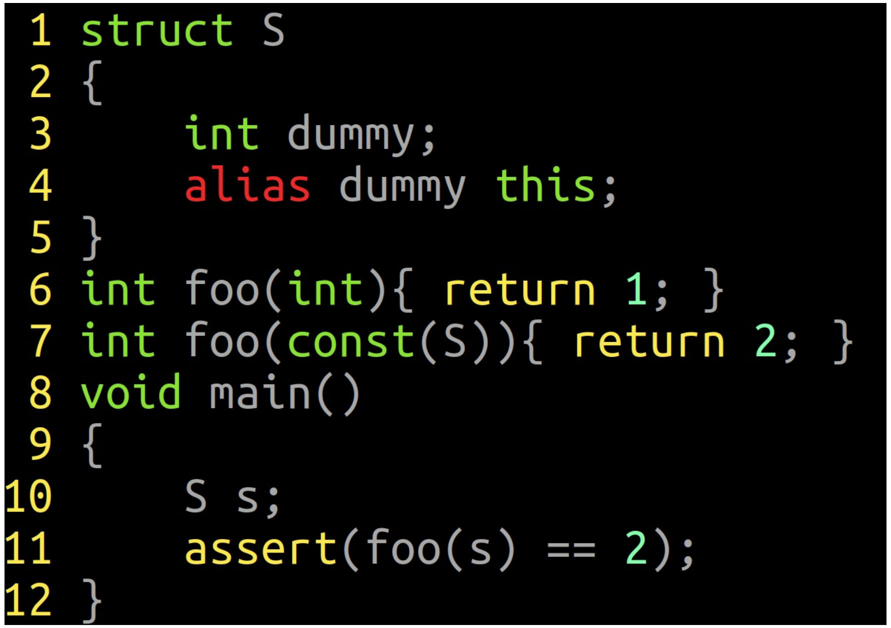
```d
struct LibraryStruct
{
	string s = "Hello";
}

struct MyWrapper
{
	LibraryStruct b;

	alias b this;
}

void main()
{
	MyWrapper a;
	string b = a.s;
}
```

## alias this - Classes

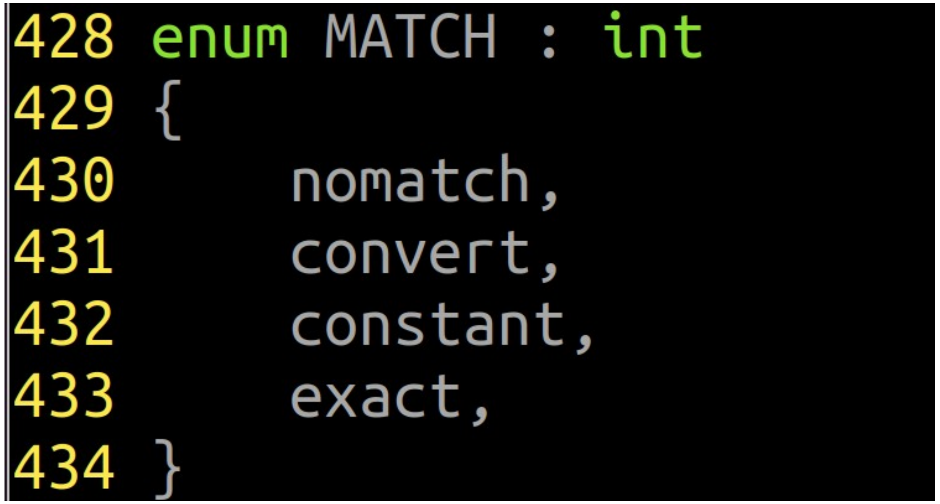


```d
class A
{
    void fun() {}
}

class B
{
    void fun() {}
}

class C : B
{
    A a;

    alias a this;
}

C c = new C;
c.fun(); // ?
```

# alias this – Overload Resolution



```d
struct S
{
    int dummy;
    alias dummy this;
}
int foo(int){ return 1; }
int foo(const(S)){ return 2; }
void main()
{
    S s;
    assert(foo(s) == 2);
}
```

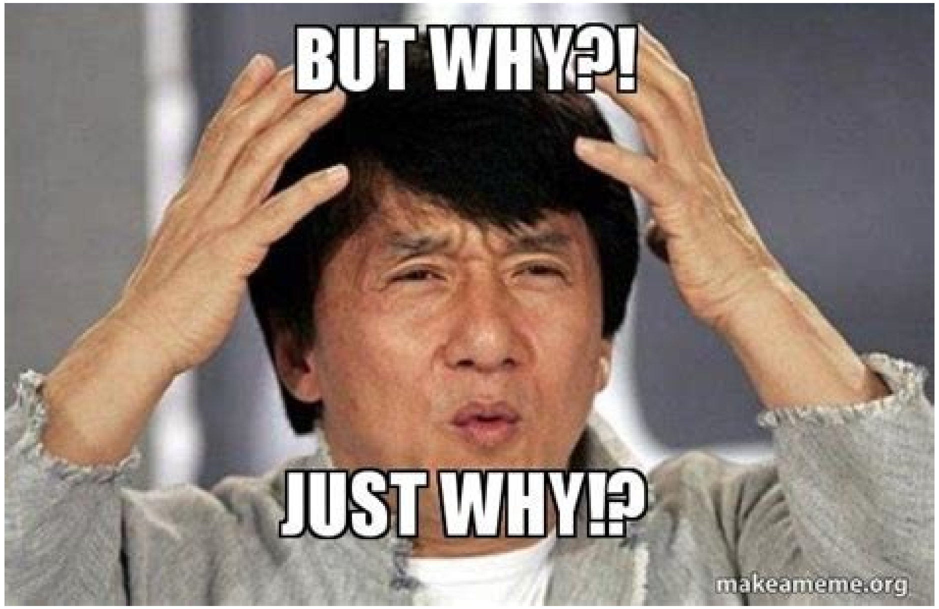


```d
enum MATCH : int
{
    nomatch,
    convert,
    constant,
    exact,
}
```

## Attributes

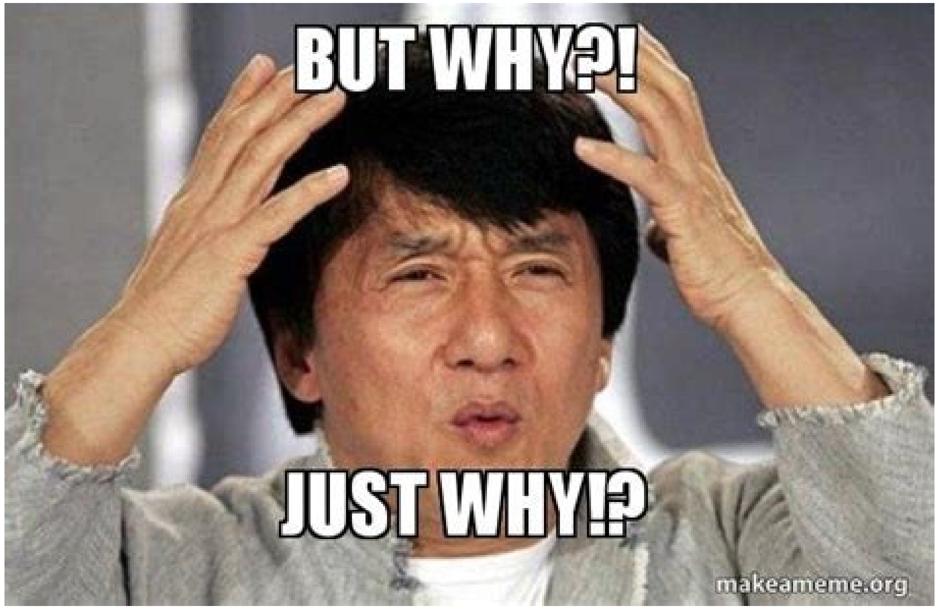
## Attributes - pure

- strongly/weakly pure
- No compiler optimizations based on pure
- One of the reasons why __metadata/__mutable DIP was killed
- Major pain when templating runtime hooks



## Attributes - nothrow

- Works with exceptions that are caught



## Attributes - @nogc, @safe

- @safe => just go to Rust
- @nogc => kind of ok

## Attributes

- Is this really the compiler’s job? (linter)
- A lot of complexity that leads to other complexity
- We have them, we need to support them (phobos v3)

## Editions

## Editions

- Pleaso, no!
- Issuing deprecations is actually a good mechanism

## Conclusions

## Conclusions

- Complexity breeds complexity
- We need to make sure that what is added actually brings tangible benefits
- We need to be able to reduce complexity