Illinois Computer Science Summer Teaching Workshop

Revisiting the D Programming Language

My 2^x Programming Language

Web: mshah.io
YouTube www.youtube.com/c/MikeShah
Social: mikeshah.bsky.social
Courses: courses.mshah.io
Talks: http://tinyurl.com/mike-talks

15 minutes | Audience: For All!
11:00 - 11:15 Central Time Tues,June 3rd, 2025

Abstract (Which you already read :))

Talk Abstract: The D programming language (dlang) is a systems programming language created by Walter Bright and released in 2001. D has continued to evolve during its near 25 year history, and in this talk I will re-introduce the D programming language, as someone who has now decided to do significant amount of teaching with the language. Why I made the decision to choose this language has to do with its ability to scale through the curriculum. The results of my decision have surprised even myself with the success! In this talk, I'll discuss what courses I experimented with, training TA's to use the language, the student outcomes, how students responded, and my thoughts going forward. Audience members will leave this talk with lessons about changing the programming language, and hopefully with courage to make choices that best benefit student outcomes.

Your Tour Guide for Today

Mike Shah

- **Current Role:** Teaching Faculty at **Yale University** (Previously Teaching Faculty at Northeastern University)
  - **Teach/Research**: computer systems, graphics, geometry, game engine development, and software engineering.
- **Available for:**
  - **Contract work** in Gaming/Graphics Domains
    - e.g. tool building, plugins, code review
  - **Technical training** (virtual or onsite) in Modern C++, D, and topics in Performance or Graphics APIs
- **Fun**:
  - Guitar, running/weights, traveling, video games, and cooking are fun to talk to me about!

Web
www.mshah.io
YouTube
https://www.youtube.com/c/MikeShah
Non-Academic Courses
courses.mshah.io
Conference Talks
http://tinyurl.com/mike-talks

Slides

- Note:
  - Slides will be available for this talk on my website: www.mshah.io
  - Just google “Mike Shah Yale” and you’ll find them under “Talks”

Question to Audience

Has anyone heard this wisdom before?

- **“The only reason to switch programming languages, is if it gives you 10x across a dimension [performance/safety/productivity]” - paraphrased**

Has anyone heard this wisdom before?

- **“The only reason to switch programming languages, is if it gives you 10x across a dimension [performance/safety/productivity]” - paraphrased**

- I thought I’d share with this crowd, that this wisdom makes me laugh 😂
  - (Sometimes even roll on the floor laughing 🤣🤣🤣🤣)

Why am I am laughing 😂 (and dispelling this myth)

- We are computer scientists after all -- there's lots of reasons to choose languages, and we don't need a 10x revolution
- If we can get a 2x across one dimension (with most all things being equal), that may be good enough!

Why am I am laughing 😂 (and dispelling bad wisdom)

- We are computer scientists after all -- there's lots of reasons to choose languages, and we don't need a 10x revolution
- If we can get a 2x across one dimension (with most all things being equal), that may be good enough!
- 2x easier to write
- 2x faster code generated
- 2x safer code
- 2x better tooling
- 2x community
- 2x more fun
- 2x more ... etc.

Why am I am laughing 😂 (and dispelling bad wisdom)

- We are computer scientists after all -- there's lots of reasons to choose languages, and we don't need a 10x revolution
- Maybe the numbers look something like this in practice

- 5x easier to write
- 1.1x faster code generated
- 4x safer code
- 0.5x better tooling
- 1x community
- 10x more (perceived) fun
- 2x more ... etc.

Exponential Function Reminder: $O(2^n)$

- It’s actually not that hard to get an exponential improvement (and hit that ‘10x’ mark or more) when you choose the right language for yourself on a project or industry project
- So if you can hit a few of these dimensions (2x easier to write, 2x faster code generated, 2x safer code, 2x better tooling, 2x community, 2x more fun) while most other variables are the same -- why not try something that can give you a competitive advantage?

- Note: There’s a wide range of dimensions, so you’ll have to figure it out for your use case.
  - 2x easier to hire for, 2x more productive, 2x faster evolving, etc.

Who here has heard of **D** Language?

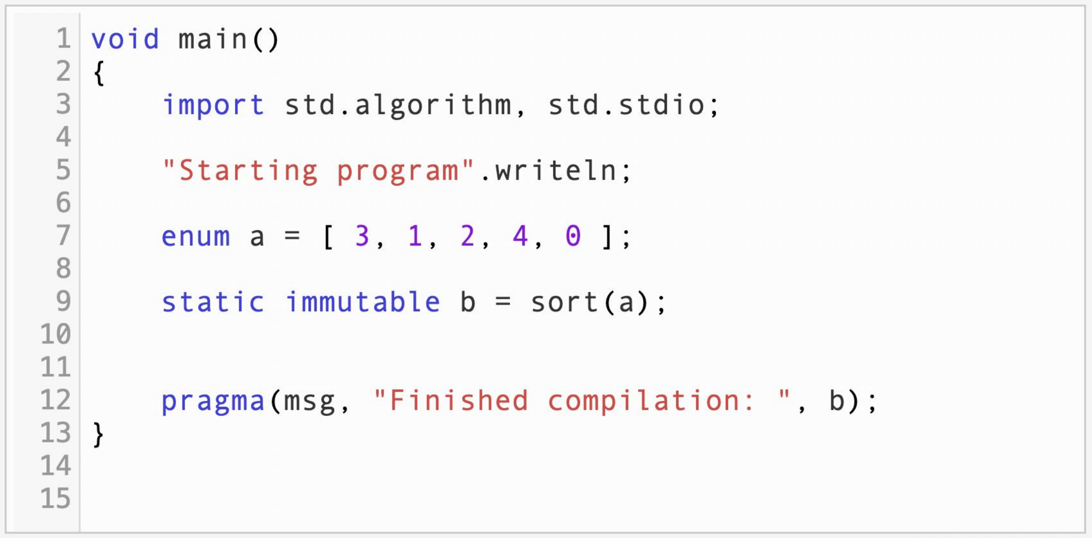
Lesson - A First Impression

La premiere impression

첫인상

Pop Quiz: (l'examen surprise!) (1/3)

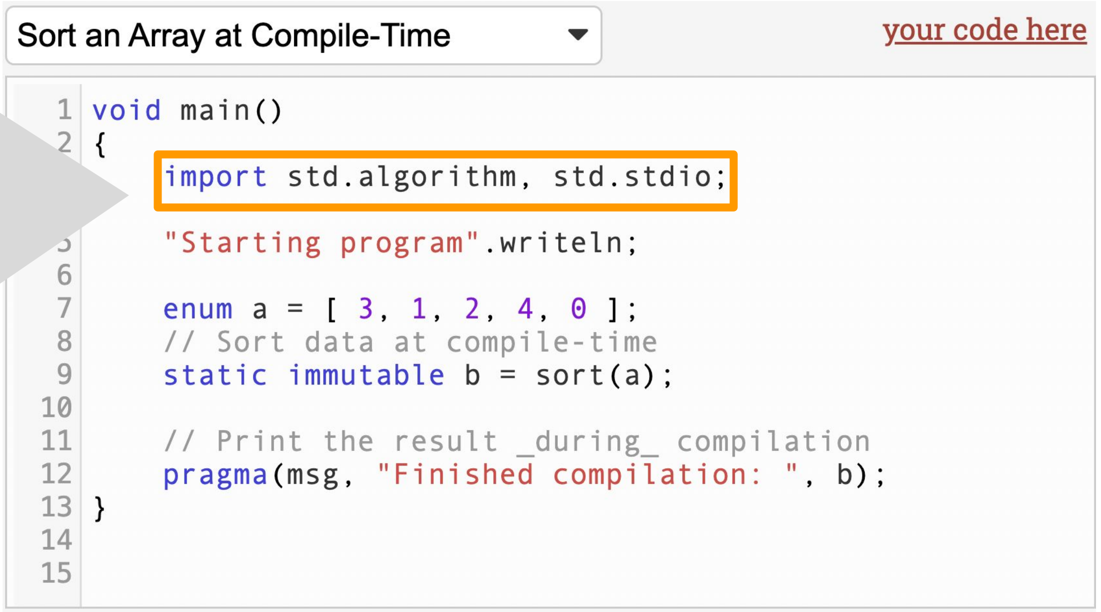
- Let's take a look at an example of D code
  - I'll give everyone a minute to think about or guess what this program does
- So... what does this program do?



```d
void main()
{
    import std.algorithm, std.stdio;

    "Starting program".writeln;

    enum a = [ 3, 1, 2, 4, 0 ];

    static immutable b = sort(a);


    pragma(msg, "Finished compilation: ", b);
}


```

Pop Quiz: (l'examen surprise!) (2/2)

- Line 3:
  - There's a built-in **standard library** (named 'Phobos')
  - There's a **module** system .
- Line 5:
  - Function call using **uniform function call syntax** (UFCS)
- Line 7:
  - enum constant, **evaluated at compile-time**
- Line 9:
  - **immutable static** data stored in b
- Line 12:
  - **pragma** outputs value after compilation (before runtime)

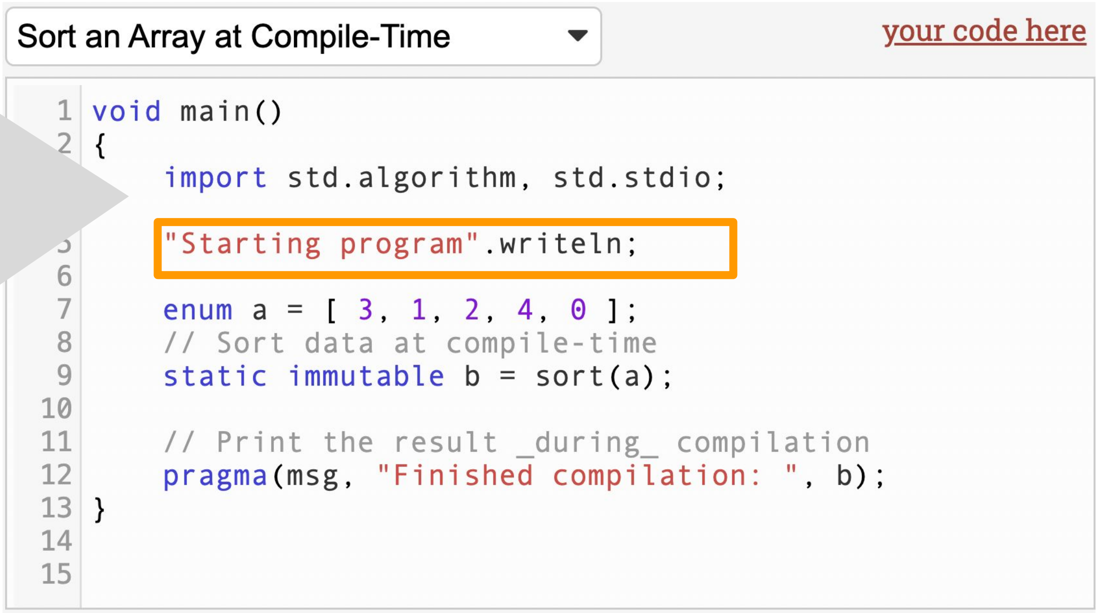


```d
void main()
{
    import std.algorithm, std.stdio;

    "Starting program".writeln;

    enum a = [ 3, 1, 2, 4, 0 ];
    // Sort data at compile-time
    static immutable b = sort(a);

    // Print the result _during_ compilation
    pragma(msg, "Finished compilation: ", b);
}
```

One of the first examples on the www.dlang.org webpage - sorting an array -- at compile-time!

Pop Quiz: (l’examen surprise!) (2/2)

- Line 3:
  - There’s a built-in **standard library** (named ‘Phobos’)
  - There’s a **module** system .
- Line 5:
  - Function call using **uniform function call syntax** (UFCS)
- Line 7:
  - enum constant, **evaluated at compile-time**
- Line 9:
  - **immutable static** data stored in b
- Line 12:
  - **pragma** outputs value after compilation (before runtime)

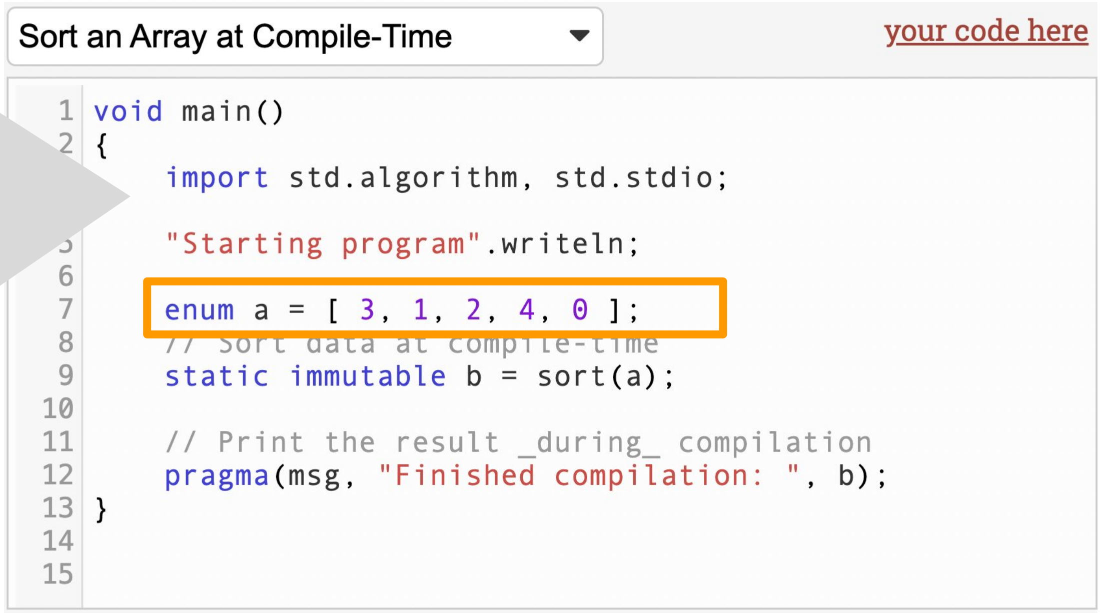


```
 1 void main()
 2 {
        import std.algorithm, std.stdio;

 5      "Starting program".writeln;
 6
 7      enum a = [ 3, 1, 2, 4, 0 ];
 8      // Sort data at compile-time
 9      static immutable b = sort(a);
10
11      // Print the result _during_ compilation
12      pragma(msg, "Finished compilation: ", b);
13 }
14
15
```

One of the first examples on the www.dlang.org webpage - sorting an array -- at compile-time!

Pop Quiz: (l'examen surprise!) (2/2)

- Line 3:
  - There's a built-in **standard library** (named 'Phobos')
  - There's a **module** system .
- Line 5:
  - Function call using **uniform function call syntax** (UFCS)
- Line 7:
  - enum constant, **evaluated at compile-time**
- Line 9:
  - **immutable static** data stored in b
- Line 12:
  - **pragma** outputs value after compilation (before runtime)


Sort an Array at Compile-Time

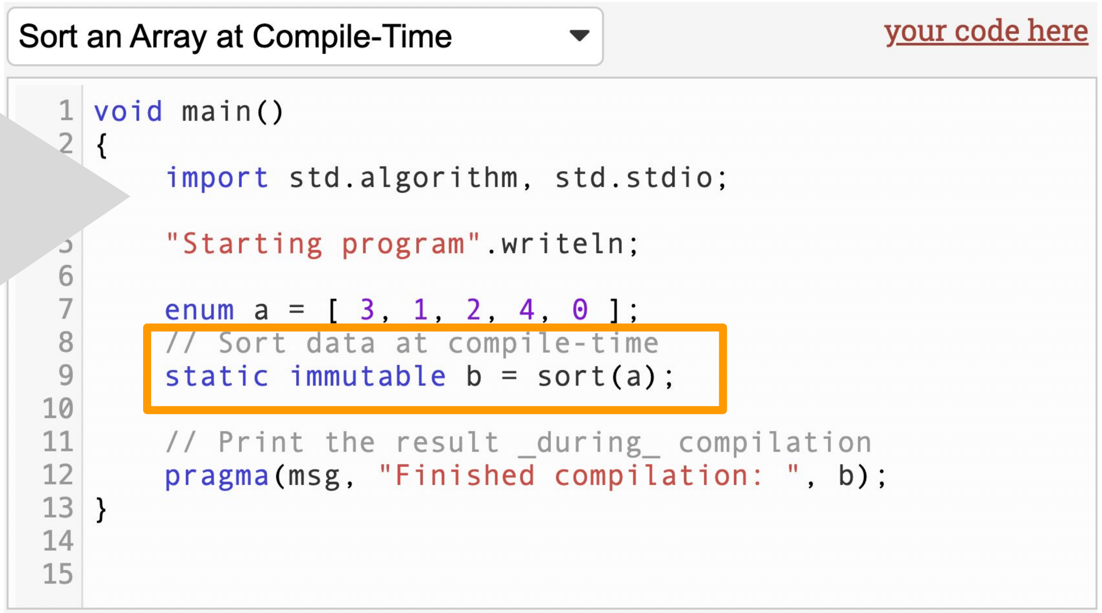
your code here

```
 1  void main()
 2  {
 3      import std.algorithm, std.stdio;
 4
 5      "Starting program".writeln;
 6
 7      enum a = [ 3, 1, 2, 4, 0 ];
 8      // Sort data at compile-time
 9      static immutable b = sort(a);
10
11      // Print the result _during_ compilation
12      pragma(msg, "Finished compilation: ", b);
13  }
14
15
```

One of the first examples on the www.dlang.org webpage - sorting an array -- at compile-time!

Pop Quiz: (l’examen surprise!) (2/2)

- Line 3:
    - There’s a built-in **standard library** (named ‘Phobos’)
    - There’s a **module** system .
- Line 5:
    - Function call using **uniform function call syntax** (UFCS)
- Line 7:
    - enum constant, **evaluated at compile-time**
- Line 9:
    - **immutable static** data stored in binary
- Line 12:
    - **pragma** outputs value after compilation (before runtime)

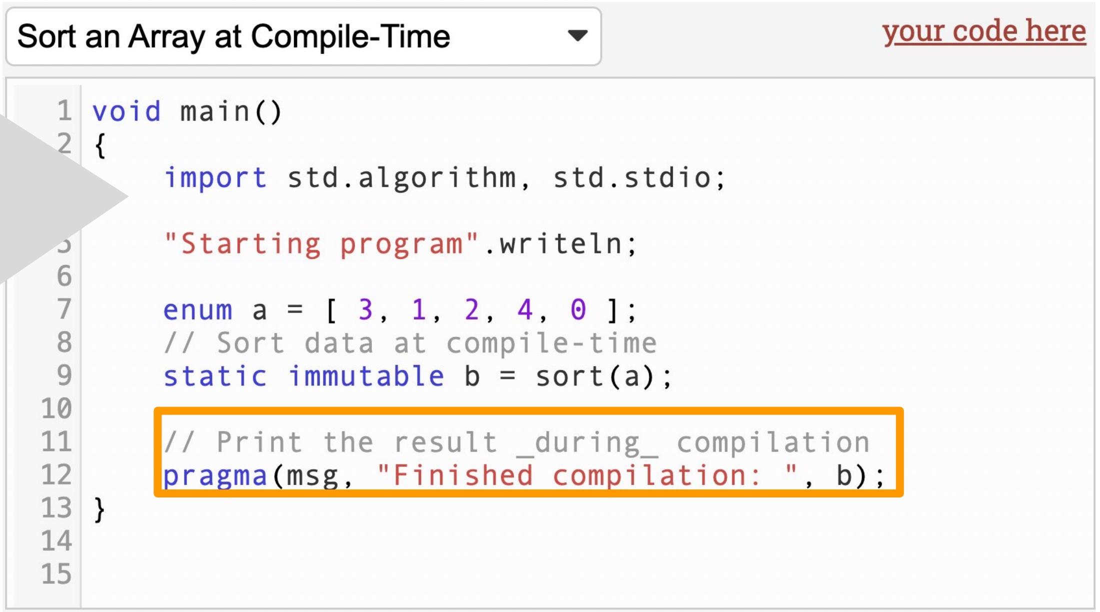


```
 1 void main()
 2 {
       import std.algorithm, std.stdio;

 5     "Starting program".writeln;
 6
 7     enum a = [ 3, 1, 2, 4, 0 ];
 8     // Sort data at compile-time
 9     static immutable b = sort(a);
10
11     // Print the result _during_ compilation
12     pragma(msg, "Finished compilation: ", b);
13 }
14
15
```

One of the first examples on the www.dlang.org webpage - sorting an array -- at compile-time!

Pop Quiz: (l’examen surprise!) (2/2)

- Line 3:
  - There’s a built-in **standard library** (named ‘Phobos’)
  - There’s a **module** system .
- Line 5:
  - Function call using **uniform function call syntax** (UFCS)
- Line 7:
  - enum constant, **evaluated at compile-time**
- Line 9:
  - **immutable static** data stored in binary
- Line 12:
  - **pragma** outputs value after compilation (before runtime)

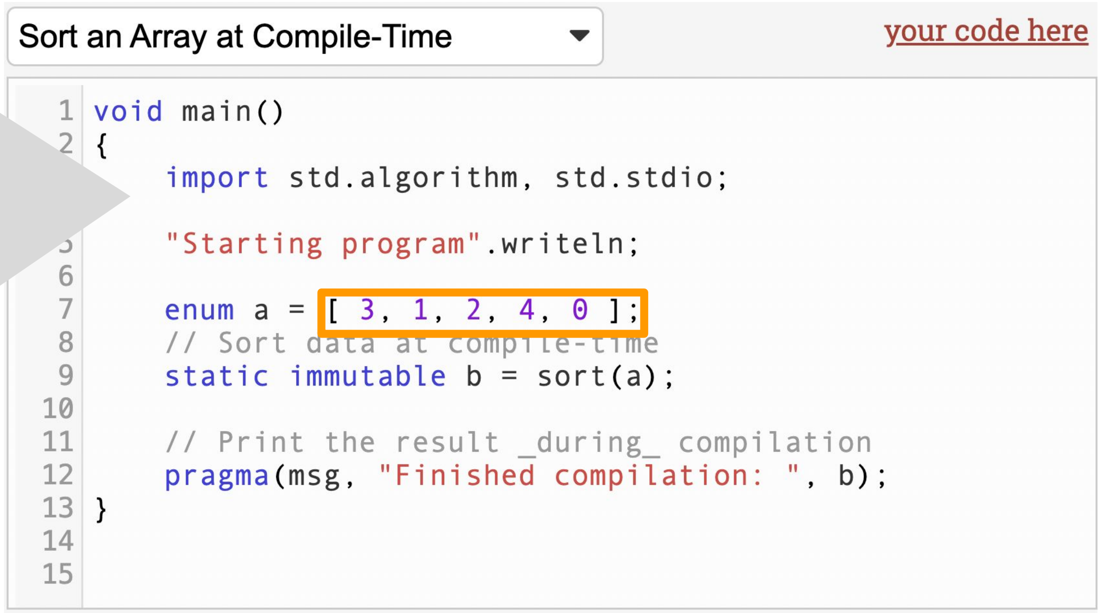


```
 1  void main()
 2  {
 3      import std.algorithm, std.stdio;
 4
 5      "Starting program".writeln;
 6
 7      enum a = [ 3, 1, 2, 4, 0 ];
 8      // Sort data at compile-time
 9      static immutable b = sort(a);
10
11      // Print the result _during_ compilation
12      pragma(msg, "Finished compilation: ", b);
13  }
14
15
```

One of the first examples on the www.dlang.org webpage - sorting an array -- at compile-time!

Pop Quiz: (l'examen surprise!) (2/2)

- Line 7:
  - This is a **fixed-size array**.
  - We can slice into it
    - e.g.
    - a[0 .. 2] returns [3,1,2]
  - Arrays (whether dynamic or static) know their **'length'** and store the **'ptr'** together.



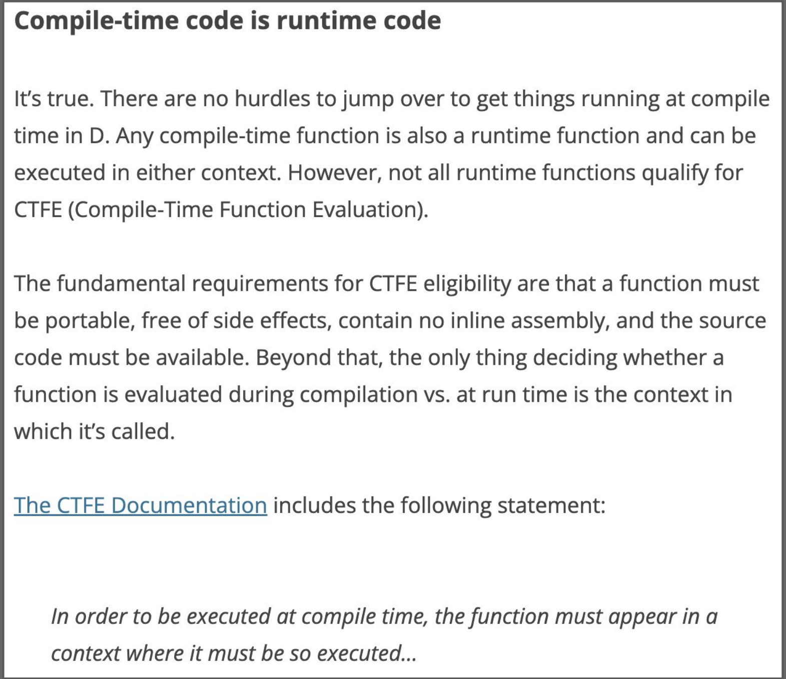
```
 1 void main()
 2 {
        import std.algorithm, std.stdio;

 5      "Starting program".writeln;
 6
 7      enum a = [ 3, 1, 2, 4, 0 ];
 8      // Sort data at compile-time
 9      static immutable b = sort(a);
10
11      // Print the result _during_ compilation
12      pragma(msg, "Finished compilation: ", b);
13 }
14
15
```

One of the first examples on the www.dlang.org webpage - sorting an array -- at compile-time!

Why you might care to look?

- D tries to **execute as much as possible at compile-time**
  - And the code...just looks like regular code!
- Compile-time execution saves the user time at run-time -- big win!

- https://dlang.org/blog/2017/06/05/compile-time-sort-in-d/
- https://tour.dlang.org/tour/en/gems/compile-time-function-evaluation-ctfe



Compile-time code is runtime code

It's true. There are no hurdles to jump over to get things running at compile time in D. Any compile-time function is also a runtime function and can be executed in either context. However, not all runtime functions qualify for CTFE (Compile-Time Function Evaluation).

The fundamental requirements for CTFE eligibility are that a function must be portable, free of side effects, contain no inline assembly, and the source code must be available. Beyond that, the only thing deciding whether a function is evaluated during compilation vs. at run time is the context in which it's called.

The CTFE Documentation includes the following statement:

> *In order to be executed at compile time, the function must appear in a context where it must be so executed...*

your code here

```
d.stdio;
n;
ing_ompilation
mpilation: ", b);
```

compilation

- This program does most of its work (the working) at compile-time!

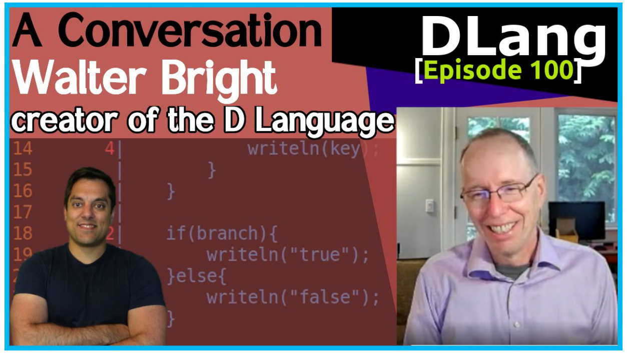
One of the first examples on the www.dlang.org webpage - sorting an array -- at compile-time!

Lesson - D Language History

Note: I will refer to D or DLang to mean any version of D version 2.X.X or later

D Language Creator - Walter Bright [wiki]

- Famously created the Zortech C++ compiler
  - Also a known game designer creating Empire
- Created a C Compiler (Datalight C compiler)
- Between 1999-2006 worked alone on D version 1 (i.e. D1) programming language.
- Around 2006 or 2007 D2 would start being developed with Andrei Alexandrescu and others.
  - Full history here - Origins of the D Programming Language
    - https://dl.acm.org/doi/pdf/10.1145/3386323

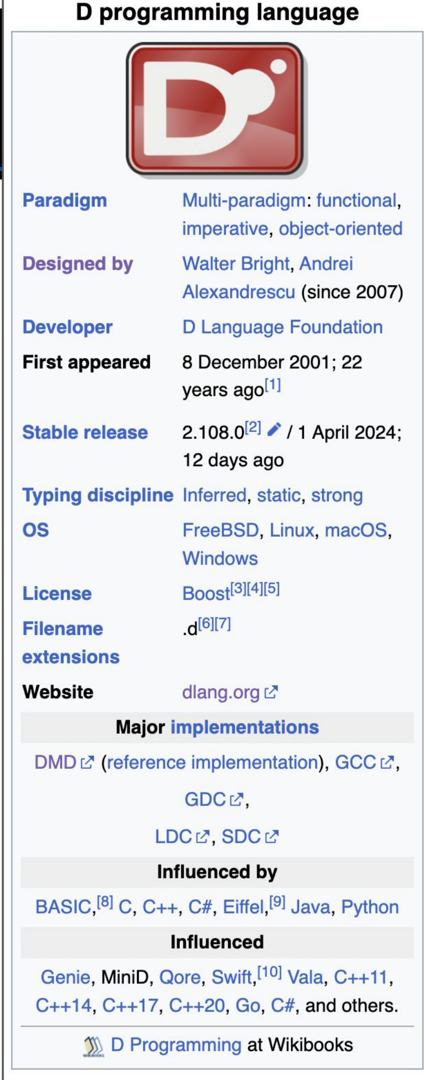


```
writeln(key);
    }
}

if(branch){
    writeln("true");
}else{
    writeln("false");
}
```

My full interview with Walter Bright
https://www.youtube.com/watch?v=O8WEykJraQc

DLang

- DLang has also been evolving and growing since it was first created in 2001 [wiki]
- Revisiting D has helped me improve my code in other other C style languages -- **but now I primarily use the D language.**
- I hope if you find the same joy as I do, whatever language you end up using, you'll end up improving your programming and software engineering skills.
- **Today, I want to sell you on trying D Lang for teaching -- revisiting a language that has evolved for nearly 25 years!**



D programming language

Paradigm	Multi-paradigm: functional, imperative, object-oriented
Designed by	Walter Bright, Andrei Alexandrescu (since 2007)
Developer	D Language Foundation
First appeared	8 December 2001; 22 years ago[1]
Stable release	2.108.0[2] / 1 April 2024; 12 days ago
Typing discipline	Inferred, static, strong
OS	FreeBSD, Linux, macOS, Windows
License	Boost[3][4][5]
Filename extensions	.d[6][7]
Website	dlang.org

Major implementations

DMD (reference implementation), GCC, GDC, LDC, SDC

Influenced by

BASIC,[8] C, C++, C#, Eiffel,[9] Java, Python

Influenced

Genie, MiniD, Qore, Swift,[10] Vala, C++11, C++14, C++17, C++20, Go, C#, and others.

D Programming at Wikibooks

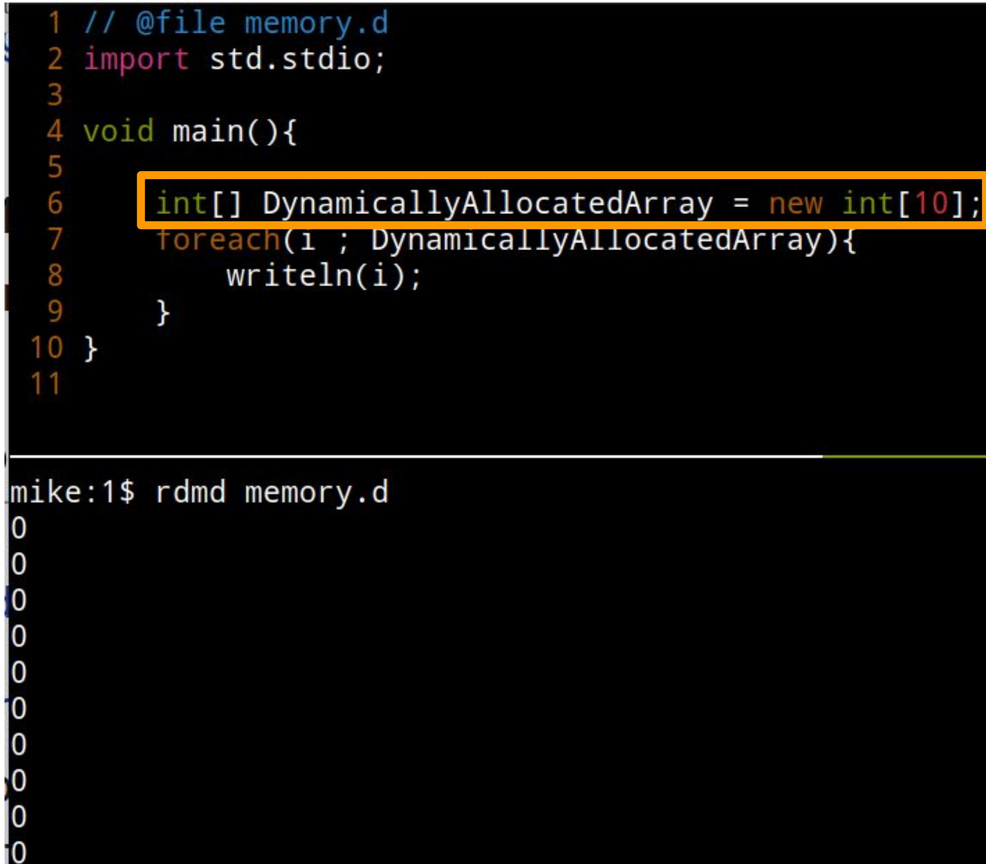
Sales Pitch 1 - Safety

The right defaults for safety

- Here are a few examples of the 'right' defaults in D I really like:
    - Variables are default initialized
        - (Can use '=void' if you do not want to spend time initializing)
    - Arrays store 'ptr' and 'length' which means...(see point below)
    - Arrays bounds can be checked by default
        - (You can turn this off to save performance)
    - Memory safety by default (with garbage collector)
        - But you can use **any** memory allocation strategy as you like
    - const is transitive -- as well as immutable (i.e. a stronger const)
    - Thread local data enabling more concurrent code
        - (You can of course create '__gshared' on variables to create global shared memory)
    - Casts -- almost always explicit done -- fewer surprises!
    - 'structs' are value types, and 'classes' are reference types
        - We'll cover more on this later.
    - Other safety features
        - Annotations (e.g. @safe, @trusted, @system) means a path for safe code!
    - ABI compatible with C

Memory - D is a systems language (1/3)

- D has a garbage collector (gc) that is on by default (it can be turned off)
  - This means that we don't have to explicitly delete memory that we have allocated.
  - In the example on the right, we dynamically allocate an array of 10 integers
  - Then I use a 'foreach' loop to display them all.
  - The garbage collector will periodically run, and remove any memory that cannot be reached for us.

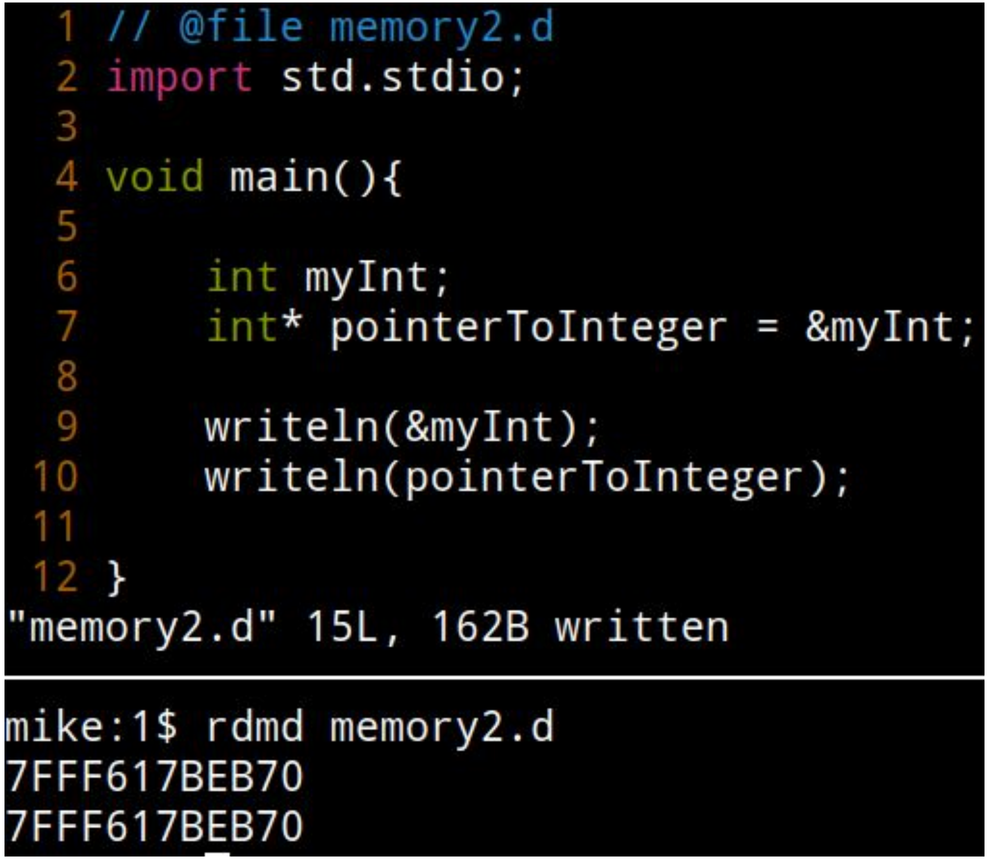


```d
// @file memory.d
import std.stdio;

void main(){

    int[] DynamicallyAllocatedArray = new int[10];
    foreach(i ; DynamicallyAllocatedArray){
        writeln(i);
    }
}

```

```
mike:1$ rdmd memory.d
0
0
0
0
0
0
0
0
0
0
```

Memory - D is a systems language (2/3)

- D does allow us to use pointers as shown on line 7
- We can use the ‘&’ operator to get the **address of** a variable.
    - Observe the address printed out below.

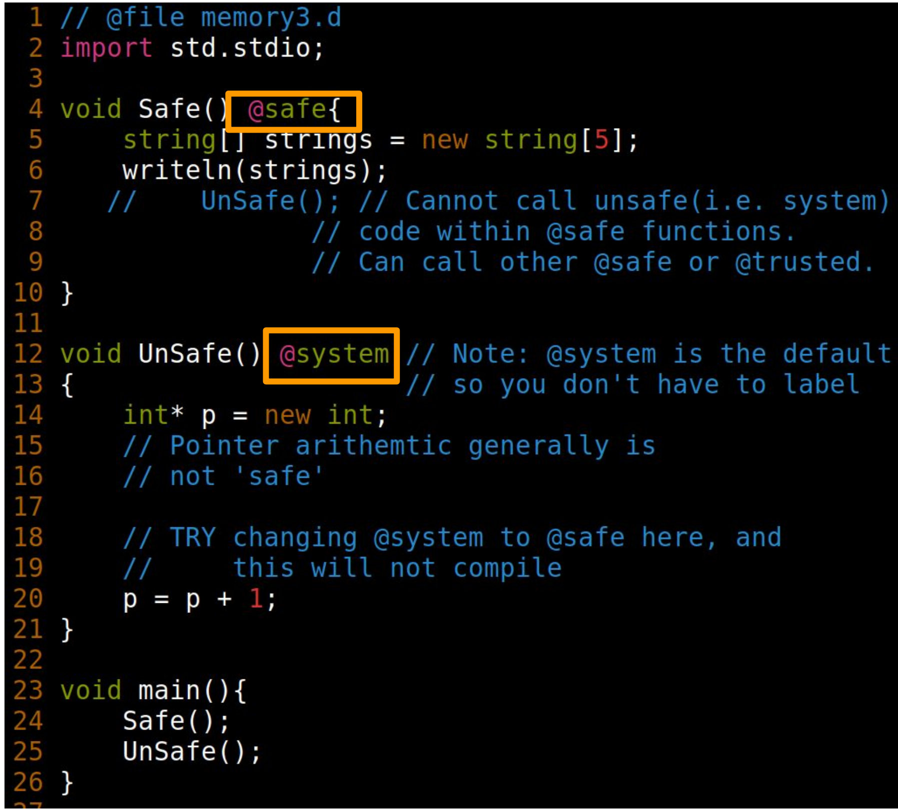


```
 1 // @file memory2.d
 2 import std.stdio;
 3
 4 void main(){
 5
 6     int myInt;
 7     int* pointerToInteger = &myInt;
 8
 9     writeln(&myInt);
10     writeln(pointerToInteger);
11
12 }
"memory2.d" 15L, 162B written
```

```
mike:1$ rdmd memory2.d
7FFF617BEB70
7FFF617BEB70
```

Memory - D is a systems language (3/3)

- D pays extra attention to memory safety.
    - You can add an @safe attribute after a function, and this will ensure that memory safety bugs are avoided.
    - @system is the ‘default’ however -- so observe on line 9 we can manipulate memory.
        - While this is the default,
        - **try changing** @system to @safe on line 9, you’ll see the compiler give you an error that this is not verified to be safe code.



```
 1 // @file memory3.d
 2 import std.stdio;
 3 
 4 void Safe() @safe{
 5     string[] strings = new string[5];
 6     writeln(strings);
 7    //     UnSafe(); // Cannot call unsafe(i.e. system)
 8                 // code within @safe functions.
 9                 // Can call other @safe or @trusted.
10 }
11 
12 void UnSafe() @system // Note: @system is the default
13 {                     // so you don't have to label
14     int* p = new int;
15     // Pointer arithemtic generally is
16     // not 'safe'
17 
18     // TRY changing @system to @safe here, and
19     //      this will not compile
20     p = p + 1;
21 }
22 
23 void main(){
24     Safe();
25     UnSafe();
26 }
```

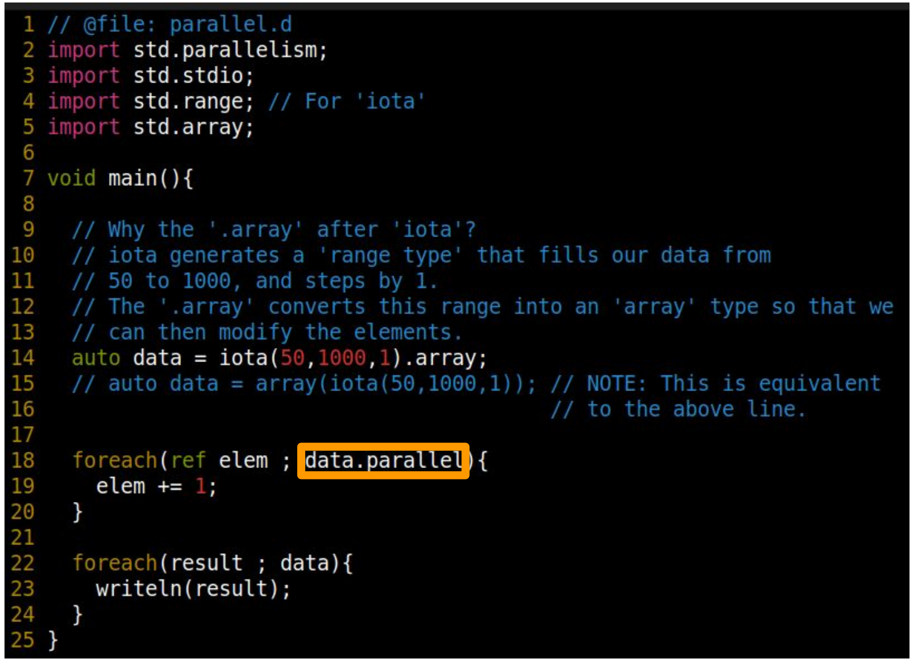
DLang: Many other small nuances improved

- Covered earlier, but D fixes many defaults that C++ inherited from C
- Initialization of values
  - (But use '=void' if you don't want to initialize for performance reasons)
- And several other small quirks --
  - `int* x,y; // in D produces two pointers to integers`
  - `int* x,y; // in C produces x as type int* and y as type int.`
  - More
    - https://dlang.org/blog/the-d-and-c-series/
    - https://dlang.org/articles/cpptod.html

Sales Pitch 2 - Performance

Where does Performance come from in DLang? (1/2)

- It is a compiled language
  - (i.e. machine code is executed as opposed to interpreting code)
  - The compilers (DMD, LDC2, GDC) have years of optimization built into them
- The language allows you to control system resources
  - i.e. You can turn on and off garbage collection for example.
- Parallelization can often be trivially enabled

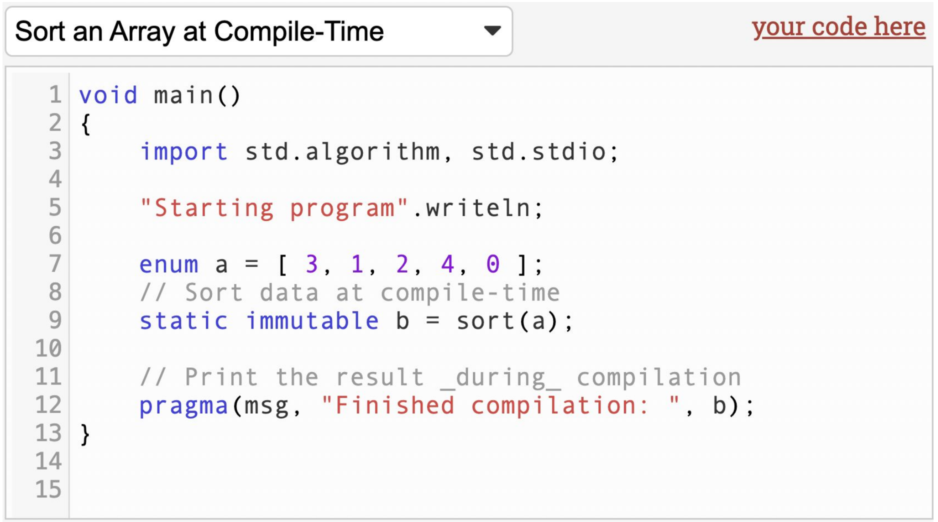


```d
// @file: parallel.d
import std.parallelism;
import std.stdio;
import std.range; // For 'iota'
import std.array;

void main(){

  // Why the '.array' after 'iota'?
  // iota generates a 'range type' that fills our data from
  // 50 to 1000, and steps by 1.
  // The '.array' converts this range into an 'array' type so that we
  // can then modify the elements.
  auto data = iota(50,1000,1).array;
  // auto data = array(iota(50,1000,1)); // NOTE: This is equivalent
                                         // to the above line.

  foreach(ref elem ; data.parallel){
    elem += 1;
  }

  foreach(result ; data){
    writeln(result);
  }
}
```

std.parallelism library allows you to simply make a 'parallel' call on a range to enable data-parallelism

Where does Performance come from in DLang? (2/2)

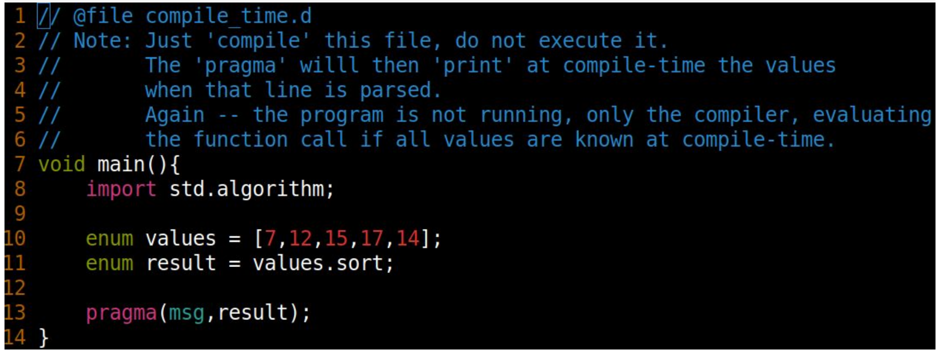
- We saw this example previously -- but it's important!
- D does lots of compile-time function evaluation (CTFE)
  - Run code at compile-time, so you don't need to evaluate at run-time
  - While it may cost us as 'developers' time to compute at compile-time, the end-user pays '0' time, as the value is already known
- The **meta-programming** and **mixins** are one of **D's superpowers** for enabling performance.



Sort an Array at Compile-Time

your code here

```d
void main()
{
    import std.algorithm, std.stdio;

    "Starting program".writeln;

    enum a = [ 3, 1, 2, 4, 0 ];
    // Sort data at compile-time
    static immutable b = sort(a);

    // Print the result _during_ compilation
    pragma(msg, "Finished compilation: ", b);
}


```



```d
// @file compile_time.d
// Note: Just 'compile' this file, do not execute it.
//       The 'pragma' willl then 'print' at compile-time the values
//       when that line is parsed.
//       Again -- the program is not running, only the compiler, evaluating
//       the function call if all values are known at compile-time.
void main(){
    import std.algorithm;

    enum values = [7,12,15,17,14];
    enum result = values.sort;

    pragma(msg,result);
}
```

Sales Pitch 3 - Productive

Three Things for Productivity

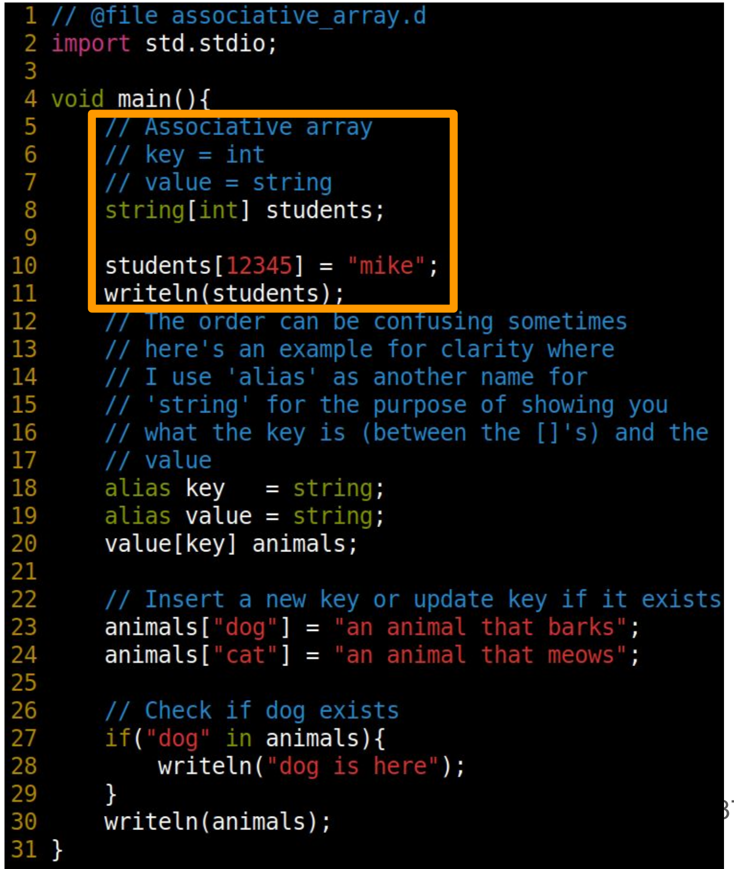
1. Built-in types
    a. I'll call them dynamic arrays (really just a pointer and a size)
    b. Associative arrays
2. 'rdmd' for otherwise building fast
    a. Regular 'dmd' compiler is otherwise fast.
3. Uniform Function Call Syntax
    a. Read-left to right
    b. Combined with 'Ranges' this becomes particularly more powerful
4. Ability to interface trivially with C code
    a. Briefly discuss 'importC'
    b. Briefly discuss 'betterC' which is a mode that disables the D runtime

Lesson - Associative Arrays

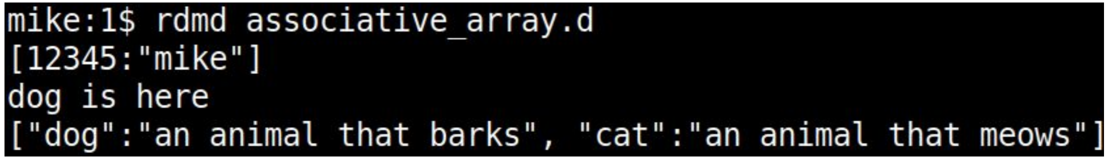
i.e. Dictionaries

Associative Arrays (and sneak peak at alias)

- Associative Arrays
  - a.k.a dictionaries, hashmaps, hash tables
  - array
  - Note: In c++ this is a std::map or more specifically std::unordered_map
- **D is as simple as Python in regard to 'dynamic arrays, dictionaries'**
- https://tour.dlang.org/tour/en/basics/arrays



```d
// @file associative_array.d
import std.stdio;

void main(){
    // Associative array
    // key = int
    // value = string
    string[int] students;

    students[12345] = "mike";
    writeln(students);
    // The order can be confusing sometimes
    // here's an example for clarity where
    // I use 'alias' as another name for
    // 'string' for the purpose of showing you
    // what the key is (between the []'s) and the
    // value
    alias key   = string;
    alias value = string;
    value[key] animals;

    // Insert a new key or update key if it exists
    animals["dog"] = "an animal that barks";
    animals["cat"] = "an animal that meows";

    // Check if dog exists
    if("dog" in animals){
        writeln("dog is here");
    }
    writeln(animals);
}
```



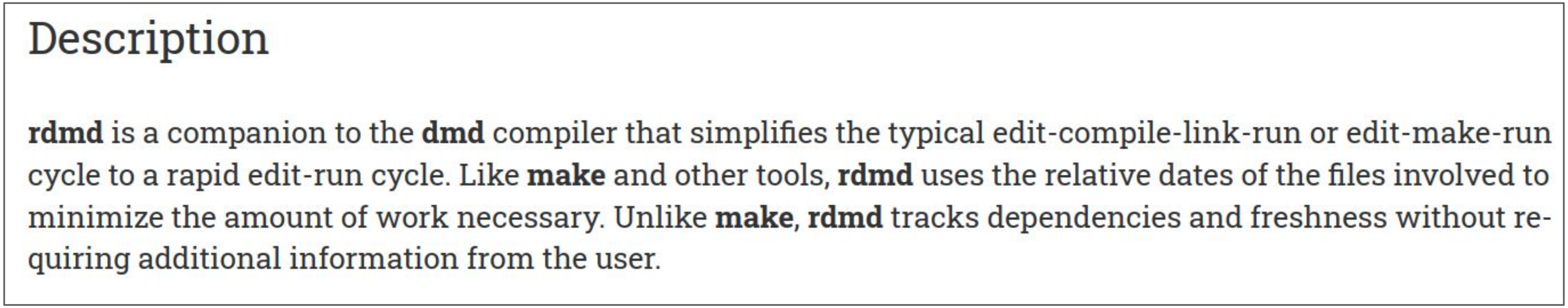
```
mike:1$ rdmd associative_array.d
[12345:"mike"]
dog is here
["dog":"an animal that barks", "cat":"an animal that meows"]
```

I want to pause for a moment and show you a little bit more -- action!

That is -- I want to show you just how fast you can get started in the D language as we go through our introduction.

rdmd is a tool that will make you think you're working in Python -- with the power of a Systems language!

rdmd

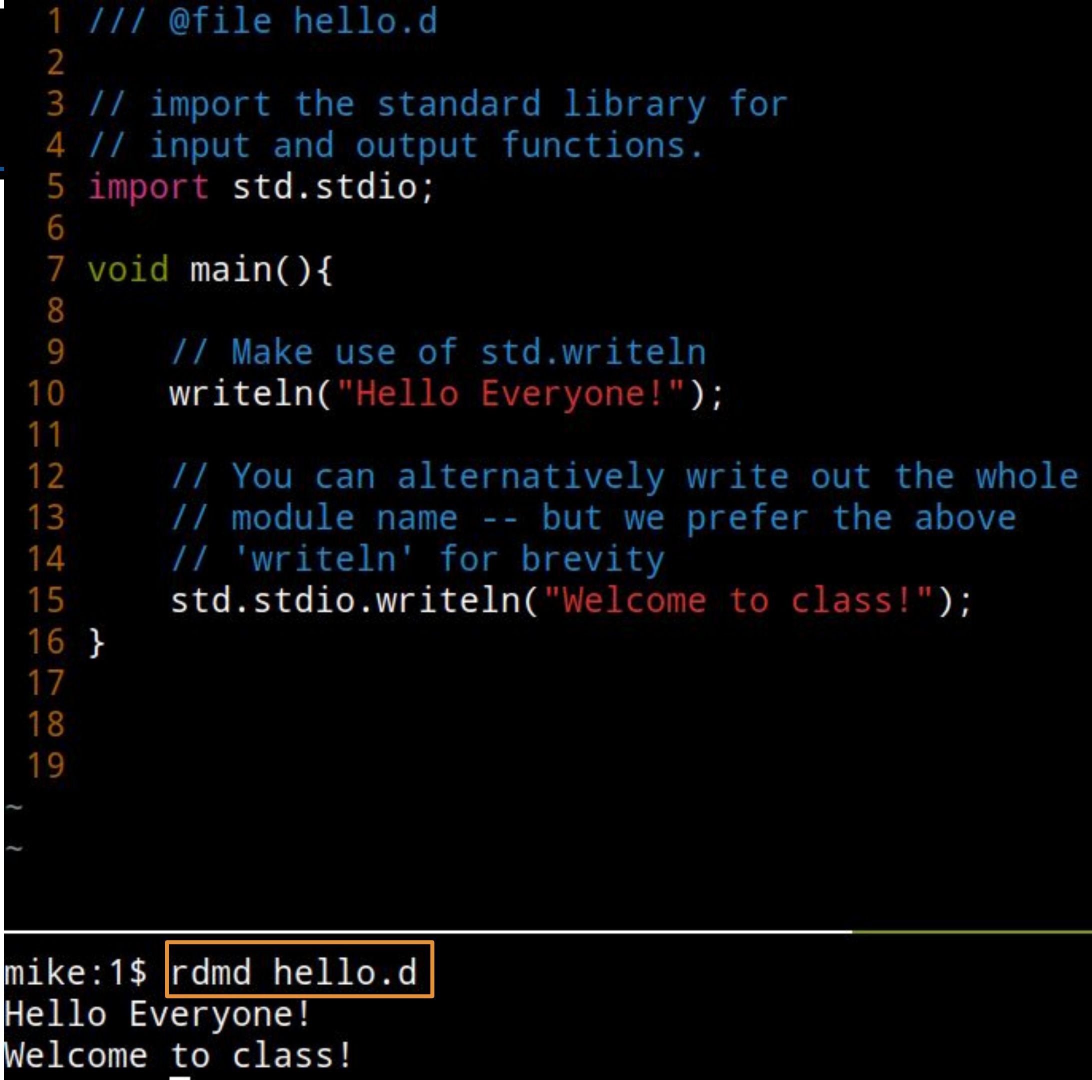


Description

rdmd is a companion to the **dmd** compiler that simplifies the typical edit-compile-link-run or edit-make-run cycle to a rapid edit-run cycle. Like **make** and other tools, **rdmd** uses the relative dates of the files involved to minimize the amount of work necessary. Unlike **make**, **rdmd** tracks dependencies and freshness without requiring additional information from the user.

rdmd introduction

- Now I’m going to re-run the hello.d program again
  - This time with a ‘shortcut’, the rdmd
  - This allows me to speed up my edit-compile-run cycle
    - rdmd is a smart tool to help us iterate more quickly when writing D code
- Note: You can also use:
  - dmd -run hello.d
  - ldc2 -run hello.d

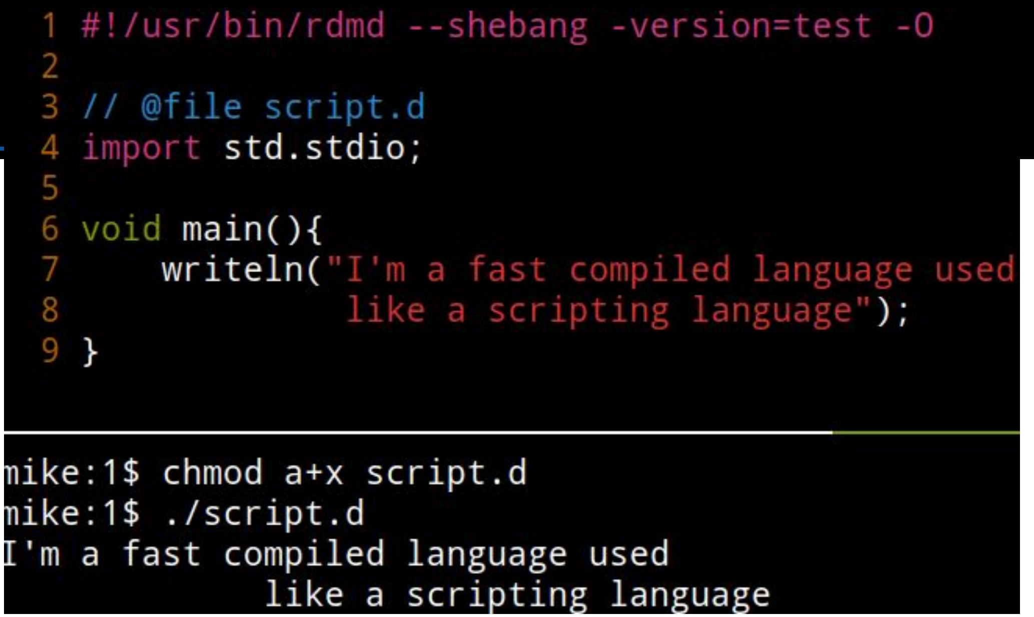


```d
/// @file hello.d

// import the standard library for
// input and output functions.
import std.stdio;

void main(){

    // Make use of std.writeln
    writeln("Hello Everyone!");

    // You can alternatively write out the whole
    // module name -- but we prefer the above
    // 'writeln' for brevity
    std.stdio.writeln("Welcome to class!");
}



~
~
```

```
mike:1$ rdmd hello.d
Hello Everyone!
Welcome to class!
```

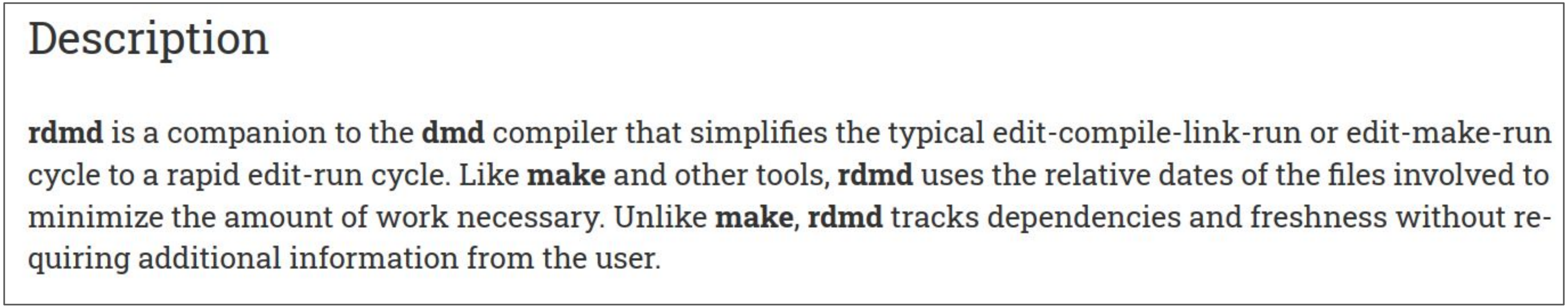
rdmd scripts

- You can check out more here: https://dlang.org/rdmd.html
  - Having the rdmd tool allows us to essentially use the D compiler like a scripting language
    - See example to the right



```d
#!/usr/bin/rdmd --shebang -version=test -O

// @file script.d
import std.stdio;

void main(){
    writeln("I'm a fast compiled language used
             like a scripting language");
}
```

```
mike:1$ chmod a+x script.d
mike:1$ ./script.d
I'm a fast compiled language used
             like a scripting language
```



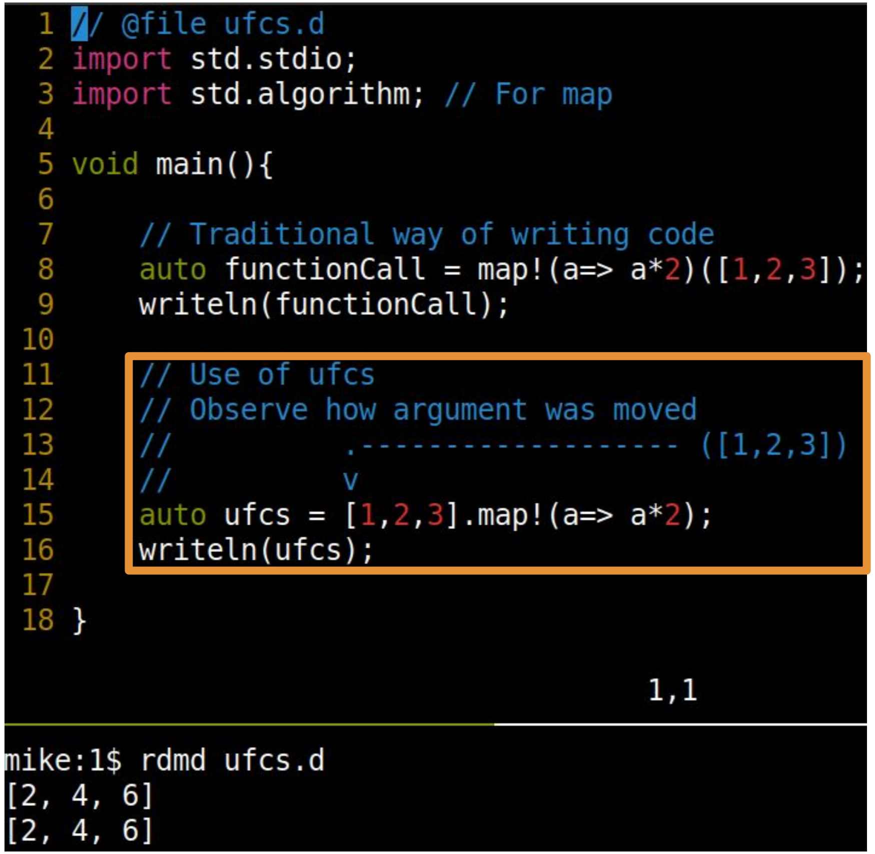
Description

rdmd is a companion to the **dmd** compiler that simplifies the typical edit-compile-link-run or edit-make-run cycle to a rapid edit-run cycle. Like **make** and other tools, **rdmd** uses the relative dates of the files involved to minimize the amount of work necessary. Unlike **make**, **rdmd** tracks dependencies and freshness without re-quiring additional information from the user.

Lesson - Uniform Function Call Syntax (UFCS)

Uniform Function Call Syntax and Chaining (1/2)

- Allows you to call free functions with the '.' syntax
  - e.g.
    - func(param)) is called as
    - param.func.
  - d tour - uniform-function-call-syntax-ufcs
- Article by Walter Bright
  - [archived link]

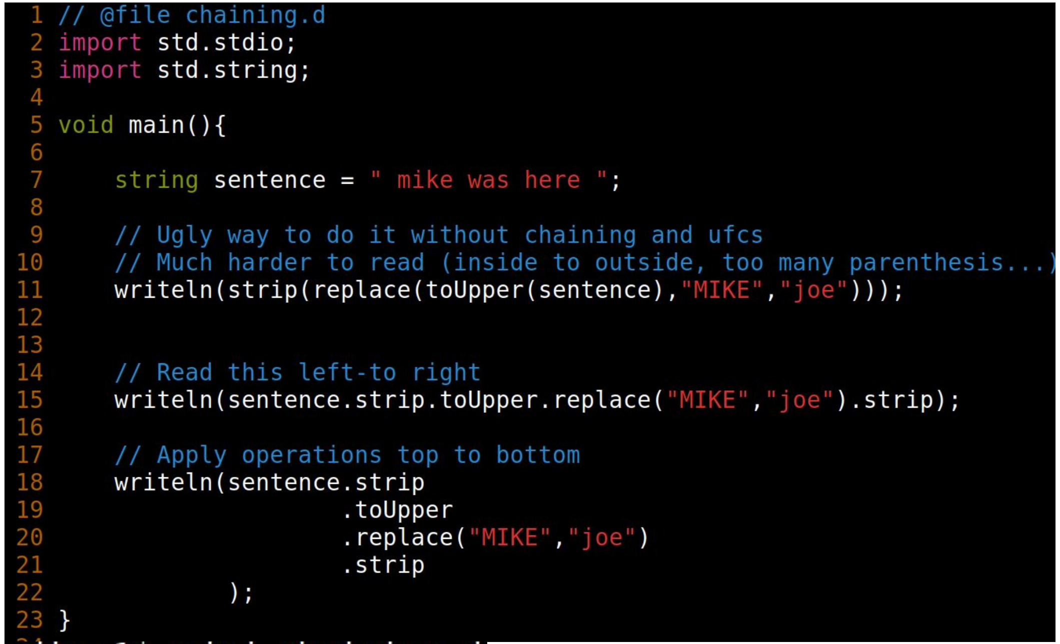


```d
// @file ufcs.d
import std.stdio;
import std.algorithm; // For map

void main(){

    // Traditional way of writing code
    auto functionCall = map!(a=> a*2)([1,2,3]);
    writeln(functionCall);

    // Use of ufcs
    // Observe how argument was moved
    //           .-------------------- ([1,2,3])
    //           v
    auto ufcs = [1,2,3].map!(a=> a*2);
    writeln(ufcs);

}
```

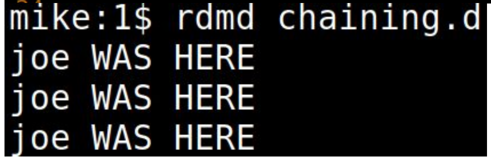
```
                                              1,1
mike:1$ rdmd ufcs.d
[2, 4, 6]
[2, 4, 6]
```

Uniform Function Call Syntax and Chaining (2/2)

- UFCS allows you to more conveniently chain together function calls
  - Here's an example of chaining together several calls
- Note: It can be useful to space out the calls.



```d
// @file chaining.d
import std.stdio;
import std.string;

void main(){

    string sentence = " mike was here ";

    // Ugly way to do it without chaining and ufcs
    // Much harder to read (inside to outside, too many parenthesis...)
    writeln(strip(replace(toUpper(sentence),"MIKE","joe")));


    // Read this left-to right
    writeln(sentence.strip.toUpper.replace("MIKE","joe").strip);

    // Apply operations top to bottom
    writeln(sentence.strip
                    .toUpper
                    .replace("MIKE","joe")
                    .strip
            );
}
```



```
mike:1$ rdmd chaining.d
joe WAS HERE
joe WAS HERE
joe WAS HERE
```

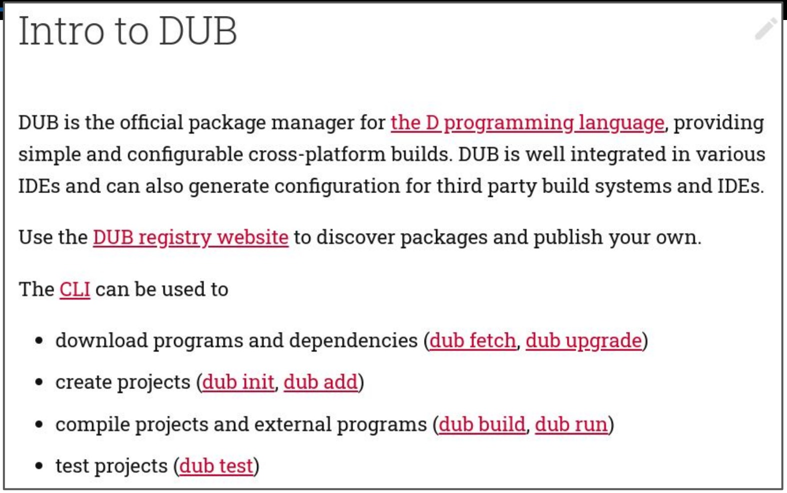
Teaching D

Exponential Gains in Teaching

- Now the trick with this talk, is that where I’ve really seen the most exponential improvement is in my teaching
- I seem to be better able to prep students with a language that’s a bit more clean
    - 2x better teaching with D
    - That’s a pretty good (and exponential) result (and remember, and exponential function)

Exponential Gains in Teaching - How? (1/2)

- **‘dub’** is the built-in package manager and build system
- Having a package manager / build system is just necessary
  - (I do show students how to compile on the command-line however!)
- (Yes, I learned about **Greenspun’s rule** recently)



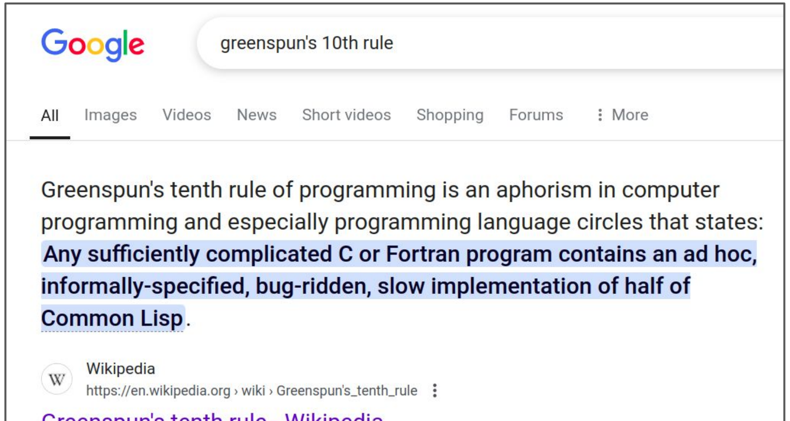
Intro to DUB

DUB is the official package manager for the D programming language, providing simple and configurable cross-platform builds. DUB is well integrated in various IDEs and can also generate configuration for third party build systems and IDEs.

Use the DUB registry website to discover packages and publish your own.

The CLI can be used to

- download programs and dependencies (dub fetch, dub upgrade)
- create projects (dub init, dub add)
- compile projects and external programs (dub build, dub run)
- test projects (dub test)

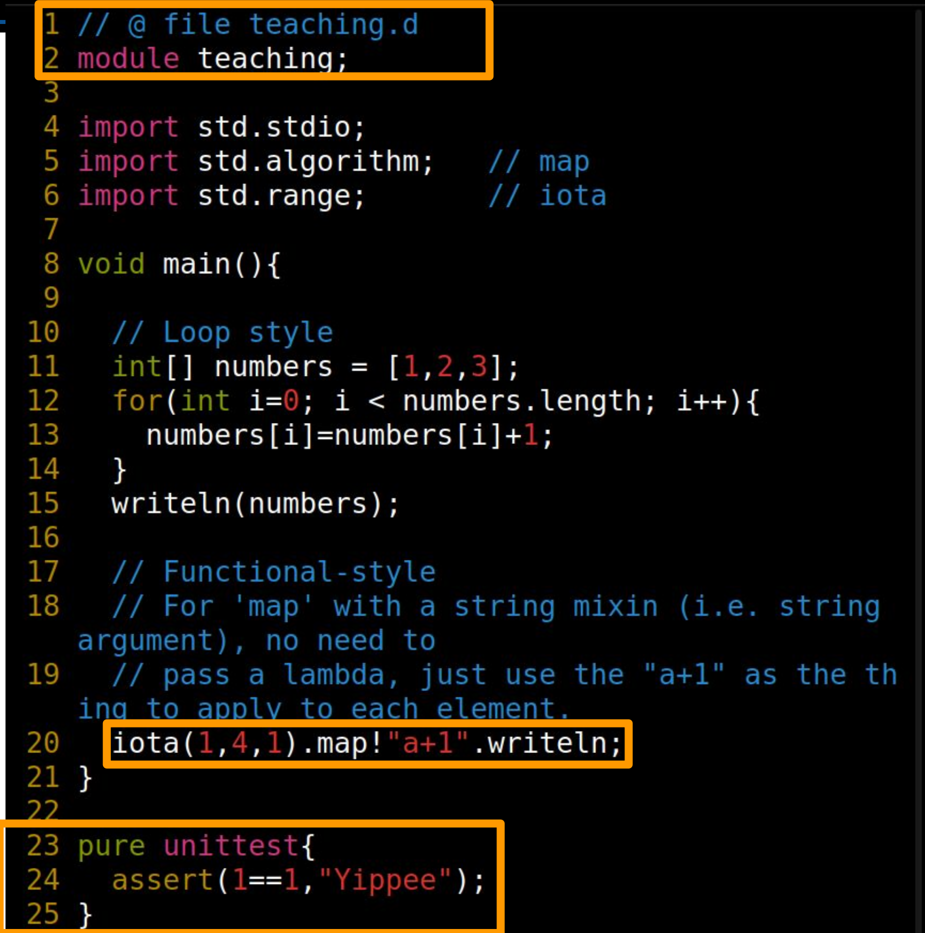


Greenspun's tenth rule of programming is an aphorism in computer programming and especially programming language circles that states: Any sufficiently complicated C or Fortran program contains an ad hoc, informally-specified, bug-ridden, slow implementation of half of Common Lisp.

Wikipedia
https://en.wikipedia.org › wiki › Greenspun's_tenth_rule

Exponential Gains in Teaching - How? (2/2)

- **Modules instead of header files** is a big when for both iteration, and management.
- **Multiple paradigms**
  - I get to talk about things like concurrency, OOP -- specifically message passing, functional programming, generic programming, etc.
- **unit testing** built-in
  - Should show unit tests for doing test-driven development
    - (note: Tests can be annotated with 'pure')
- And much more!



```d
// @ file teaching.d
module teaching;

import std.stdio;
import std.algorithm;   // map
import std.range;       // iota

void main(){

  // Loop style
  int[] numbers = [1,2,3];
  for(int i=0; i < numbers.length; i++){
    numbers[i]=numbers[i]+1;
  }
  writeln(numbers);

  // Functional-style
  // For 'map' with a string mixin (i.e. string argument), no need to
  // pass a lambda, just use the "a+1" as the thing to apply to each element.
  iota(1,4,1).map!"a+1".writeln;
}

pure unittest{
  assert(1==1,"Yippee");
}
```

Courses where I changed languages to use D

- Spring 2023
  - **Software Engineering,** C++ --> D
- Fall 2024
  - **Building Game Engines**, C++ --> D
- Spring 2025
  - **Real-Time Computer Graphics,** C++ --> D
- Fall 2025
  - (Tentatively)
  - **Computer Systems,** C --> A mix of C and D
    - (D has the ‘importC’ compiler which you can use as a C compiler)
- Note:
  - Most courses that have a final project I allow students to choose their language
  - Almost all choose D (otherwise some choose C++)

How Generally Students Respond?

- Generally most students are open to learning a new language
  - Some are disappointed to not be learning C++ in my graphics/games courses initially
  - Most by the end of the semester report being happy.
- Almost all students who previously used C++ previously reported enjoying using and collaborating on group projects in D more than C++.
- **Even better** -- you can hear directly the students perspective
- D Conf 2023:
  - YouTube: https://www.youtube.com/live/wXTlafzlJVY?si=Xpy6g5h4wtIUrt2E&t=7711
  - Link to Conference Talk Description: https://dconf.org/2023/index.html

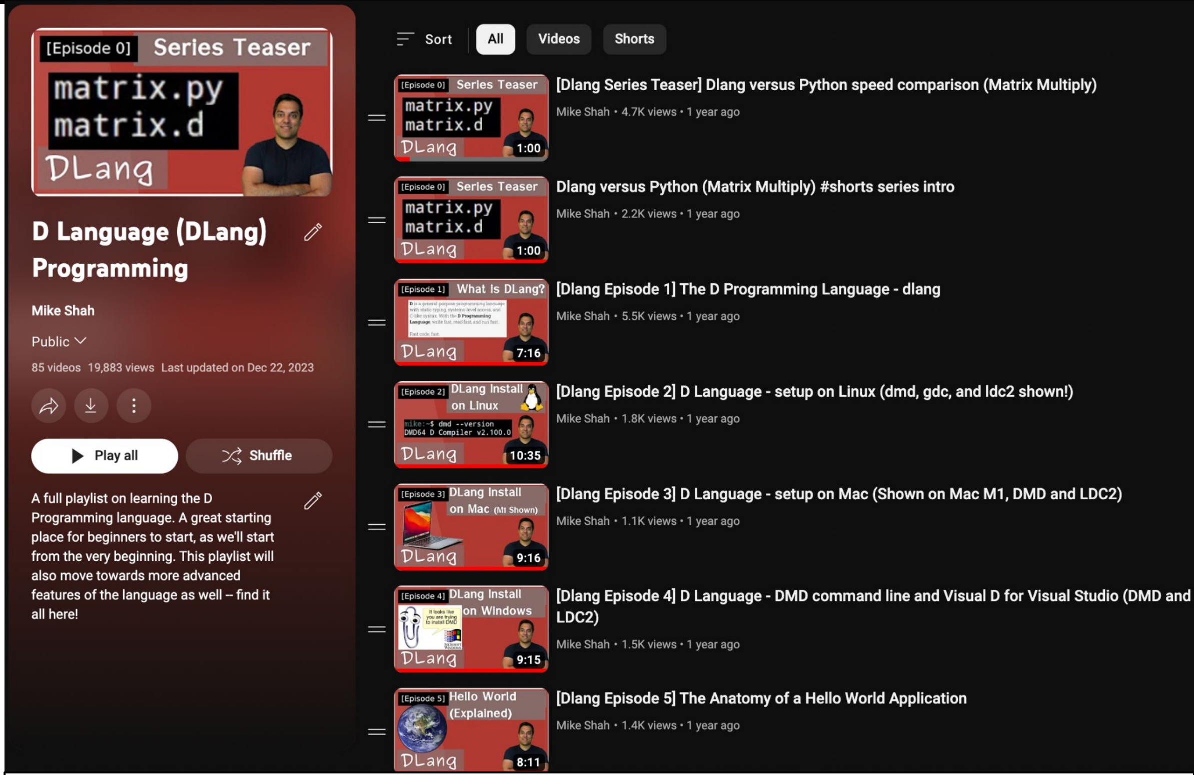
DCONF '23
August 29 - September 1
A Semester at University: Teaching Software Engineering in DLang
Social: @MichaelShah
Web: mshah.io
Courses: courses.mshah.io
YouTube: www.youtube.com/c/MikeShah
Presentor: Mike Shah, Ph.D.
11:00-11:30, Wed, Aug. 30, 2023
Audience: Everyone!

How Teaching Assistants Responded

- The first term a language gets changed, there is an ‘extra degree’ of difficulty
  - i.e. Teaching Assistants may not use D.
- In the case of D however, from C++, the transition is quite manageable.
  - Cases where support is needed, is to help with ecosystem and tooling, to help prepare teaching assistants

YouTube

- If you're going to do something new -- with teaching, you have to support it.
- I am actively adding more lessons about the D programming language
    - **129 lessons and counting**
    - **(Installation, tooling, debuggers, language features)**
    - https://www.youtube.com/c/MikeShah



D Language (DLang) Programming

Mike Shah

Public

85 videos 19,883 views Last updated on Dec 22, 2023

A full playlist on learning the D Programming language. A great starting place for beginners to start, as we'll start from the very beginning. This playlist will also move towards more advanced features of the language as well – find it all here!

[Dlang Series Teaser] Dlang versus Python speed comparison (Matrix Multiply)
Mike Shah • 4.7K views • 1 year ago

Dlang versus Python (Matrix Multiply) #shorts series intro
Mike Shah • 2.2K views • 1 year ago

[Dlang Episode 1] The D Programming Language - dlang
Mike Shah • 5.5K views • 1 year ago

[Dlang Episode 2] D Language - setup on Linux (dmd, gdc, and ldc2 shown!)
Mike Shah • 1.8K views • 1 year ago

[Dlang Episode 3] D Language - setup on Mac (Shown on Mac M1, DMD and LDC2)
Mike Shah • 1.1K views • 1 year ago

[Dlang Episode 4] D Language - DMD command line and Visual D for Visual Studio (DMD and LDC2)
Mike Shah • 1.5K views • 1 year ago

[Dlang Episode 5] The Anatomy of a Hello World Application
Mike Shah • 1.4K views • 1 year ago

https://www.youtube.com/playlist?list=PLvv0ScY6vfd9Fso-3cB4CGnSlW0E4btJV

Illinois Computer Science Summer Teaching Workshop

Thank you!

Revisiting the **D Programming Language**

My 2^x **Programming Language**

Web:	mshah.io
YouTube	www.youtube.com/c/MikeShah
Social:	mikeshah.bsky.social
Courses:	courses.mshah.io
Talks:	http://tinyurl.com/mike-talks

15 minutes | Audience: For All!
11:00 - 11:15 Tues,June 3rd, 2025

Tuesday, June 3, 2025

Morning Session: Unique Courses

Time	Presenter	Talk title
09:45 AM		Second Day’s Opening Remarks
10:00 AM	**Invited Talk: Shinkha Singh**	Teaching at a small liberal-arts college: Takeaways and Challenges
10:30 AM	LeeAnn Grant	Bridging the Digital Divide: Supporting Computer Science Education in Rural Schools
10:45 AM	Rush Sanghrajka	Learning to Ride with AI: Teaching Critical Engagement with LLMs in a Data Science Course
11:00 AM	Mike Shah	Revisiting the D Programming language for Teaching
11:15 AM	Casey W. O’Brien	Seeing Double: Securing Critical Infrastructure with Digital Twins
11:30 AM	**Invited Talk: Mariana Silva**	Turning Data Into Impact: My Experiences with Course Redesign