The Jack of all trades
Automatic memory management makes for safe, simple, and robust code. D also supports scoped resource management (aka the RAII idiom) and scope statements for deterministic transactional code that is easy to write and read. Show example ▾

Built-in linear and associative arrays, slices, and ranges make daily programming simple and pleasant for tasks, both small and large. Show example ▾

🔍 Read Fast

The best paradigm is to not impose something at the expense of others. D offers classic polymorphism, value semantics, functional style, generics, generative programming, contract programming, and more—all harmoniously integrated. Show example ▾

D offers an innovative approach to concurrency, featuring true immutable data, message passing, no sharing by default, and controlled mutable sharing across threads. Read more.

From simple scripts to large projects, D has the breadth to scale with any application’s needs: unit testing, information hiding, refined modularity, fast compilation, precise interfaces. Read more.

⚡ Run Fast

D compiles naturally to efficient native code.

D is designed such that most "obvious" code is fast and safe. On occasion a function might need to escape the confines of type safety for ultimate speed and control. For such rare cases D offers native pointers, type casts, access to any C function without any intervening translation, manual memory management, custom allocators and even inline assembly code. Show example ▾
A jack of all trades is a master of none

but oftentimes better than a master of one
My Language usage

GAME MAKER

2008 2015 2022

https://gamejolt.com/games/inkling-s-nightmare/192544
Game Maker Language (GML)

- All code does something
- Compiles to single .exe

```gml
if (keyboard_pressed(vk_up) and on_ground)
{
    vspeed = -10
    sound_play(snd_jump)
    sprite_index = spr_player_jump
}
```
My Language usage

- BSc. Computer Science at Delft University of Technology

GAME MAKER
Java

- All code does something... No

```java
public class StubFactoryFactoryProxyImpl extends StubFactoryFactoryDynamicBase {
    
    public PresentationManager.StubFactory makeDynamicStubFactory(
        PresentationManager pm, PresentationManager.ClassData classData,
        ClassLoader classLoader )
    {
        return new StubFactoryProxyImpl( classData, classLoader ) ;
    }
}
```

Java

- All code does something... No
- Compiles to .jar, requires setup
C++

- "C++ is my favorite language once I learn it"
- Segfaults instead of Exception traces
- Still had to ship glfw3.dll
- Discovered D on benchmark site
- "They made a sequel to C/C++?"

<table>
<thead>
<tr>
<th>Language</th>
<th>Relative runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>1.0</td>
</tr>
<tr>
<td>C++</td>
<td>1.0</td>
</tr>
<tr>
<td>D</td>
<td>1.1</td>
</tr>
<tr>
<td>Java</td>
<td>1.8</td>
</tr>
</tbody>
</table>
D

- Good rationale
- Automatic boilerplate
- Compiles to .exe

**Rationale**
Questions about the reasons for various design decisions for D often come up. This addresses many of them.

**C to D**
Coming from C? Here are various examples comparing the D way to the C way.

**C Preprocessor vs D**
D doesn't have a preprocessor. This article shows how to do in D what would be a task for the preprocessor in C.

**Builtin Rationale**
D offers several capabilities built in to the core language that are implemented as libraries in other languages. This article answers why.

**C++ to D**
Coming from C++? Here are various examples comparing the D way to the C++ way.

**Code coverage analysis**
D compilers come with a builtin code coverage analyzer. This article explains why and how to use it.
- Good rationale
- Automatic boilerplate
- Compiles to .exe

ubyte[] sdlBytes = cast(ubyte[]) import("SDL2.dll");

https://github.com/p0nce/d-idioms/commit/0661563ccd712aa0d91a2110aa480823a8ba9cc6
My Language usage

Java
C / C++
x86 / MIPS asm
Python
Typescript
Prolog
Coq
Scala
MiniZinc
Julia
LUA
C#

GAME MAKER

D

2008 2015 2018 2022
Using D for everything

- Lots of hobby projects in D
- Why not use specialized languages?
- Complexity in big language

OpenGL app
ELF linker
MIPS assembler
Scripts
Computer algebra system
C to D translator
Software synthesizer
Ultimate tic-tac-toe game
Codingame challenges
Chess game
Complexity in the spec

9 – The Complete Syntax of Lua

```plaintext
chase := block
  Block := ifBlock
  ifBlock := condition ? trueBlock : falseBlock
  condition := boolean
  trueBlock := Block
  falseBlock := Block

print := functionCall
  functionCall := Identifier parenExpList ? ;
  Identifier := literal | Identifier parenExpList

while := whileBlock
  whileBlock := condition whileBlock
  condition := boolean

do := doBlock
  doBlock := Block
```

Complexity for the user

- Aggregate two integers?

C: `struct`  Java: `class`  Lua: `table`

D:

```d
struct Pair { int x; int y; }
class Pair { int x; int y; }
alias Pair = Tuple!(int, int);
alias Pair = int[2];
```
Complexity in design

- String interpolation
- Feature in C#, JavaScript, Python...
- Just add it to D?

```d
void main()
{
    string name = "Dennis";
    writeln("hello ", name, "!");
    writeln("hello $name!");
}
```

Complexity in design

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIP:</td>
<td>1027</td>
</tr>
<tr>
<td>Review Count</td>
<td>2</td>
</tr>
<tr>
<td>Author:</td>
<td>Walter Bright</td>
</tr>
<tr>
<td>Implementation</td>
<td></td>
</tr>
<tr>
<td>Status:</td>
<td>Rejected</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIP:</td>
<td>1036</td>
</tr>
<tr>
<td>Review Count</td>
<td>2</td>
</tr>
<tr>
<td>Author:</td>
<td>Adam D. Ruppe, Steven Schweighoffer</td>
</tr>
<tr>
<td>Implementation</td>
<td></td>
</tr>
<tr>
<td>Status:</td>
<td>Withdrawn</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIP:</td>
<td>xxxx</td>
</tr>
<tr>
<td>Review Count</td>
<td>0</td>
</tr>
<tr>
<td>Author:</td>
<td>Andrei Alexandrescu, John Colvin <a href="mailto:john.loughran.colvin@gmail.com">john.loughran.colvin@gmail.com</a></td>
</tr>
<tr>
<td>Implementation</td>
<td></td>
</tr>
<tr>
<td>Status:</td>
<td></td>
</tr>
</tbody>
</table>

String interpolation design challenges

- `writefln!i"$x"`
- `@nogc`
- `mixin(i"void $f() {}" )`;
- `printf(i"$x")`;
- simple, easy to use
Better to mix languages?

High-level

Low-level

Scripting

Application

19
High level Python

- Embedded code written in C
- GUI in Python

https://www.st.ewi.tudelft.nl/koen/cs4140/
len = snprintf(buffer, 512, "{" "\"timeStamp\" : %u, "");

- Embedded code written in C
- GUI in Python
- JSON sent over a socket
- Hard to maintain
LUA scripting

- Super Mario 64 (1996)
- "How did they code it?"
- Emulator with Lua scripting
Increasing complexity

- From small script to 5000 LOC

Suddenly!

- Runtime nil errors
Type systems

- "Static typing" in Lua

```lua
local function checkTypes()
    for typeName, typeObj in pairs(_typeTable) do
        assert(type(typeObj._name) == "string", "Type "..typeName.." is not a string but a "..type(typeObj._name)"
        if (typeObj._prim == false and typeObj._dict ~= nil) then
            for k, v in pairs(typeObj._dict) do
                local tt = _typeTable[v.dataType]
                assert(tt ~= nil, "In struct type "..typeName.." key ["..k.."] has unknown type "..v.dataType)
                assert((v.dataType ~= "pointer" and v.dataType ~= "array") or (v.parameter ~= nil),
                   "In dictionary of "..typeName.." in "..v.dataType.." entry "..k.." has no parameter." )
            end
        end
        local t0 = type(typeObj._readFunc)
        assert(t0 == "function" or t0 == "userdata", "Read function of "..type(typeObj._name.." is of type "..t0)
        t0 = type(typeObj._writeFunc)
        assert(t0 == "function" or t0 == "userdata", "Write function of "..type(typeObj._name.." is of type "..t0)
    end
end
```
Type systems

- "Static typing" in Lua
- "Dynamic typing" in D

```d
T max(T) (T x, T y)
{
    return x > y ? x : y;
}

const x = max(10, 20);
```
Seamless data

- One slice / dynamic array type

```d
void lowLevel()
{
    import core.stdc.stdio : snprintf;
    char[8] buf;
    const n = snprintf(buf.ptr, buf.length, "%d", 99);
    highLevel(buf[0 .. n]);
}

void highLevel(char[] a)
{
    import std.stdio : writeln;
    a ~= " bottles";
    writeln(a);
}
```
Seamless data

- One slice / dynamic array type
- Good D-to-D FFI

I think the disadvantages of D being like this are obvious. An advantage of it being like this, is that if you one day decide that you'd prefer a D application have C++-style performance, you don't have to laboriously rewrite the application into a completely different language. The D-to-D FFI, as it were, is really good, so you can make transitions like that as needed, even to just the parts of the application that need them.
D code

in a Nintendo 64 emulator
Assembly hacking

- Collision viewer
- Lua API too limited / slow
- Inject MIPS assembly
- Bugs

LUI AT, 0x8039
LW T0, 0xEE9C (AT)
SLL T1, S0, 2  // 48*S0 = (4*S0-S0)*16
SUB T1, T1, S0
SLL T1, T1, 4
ADDU S5, T0, T1  // S5 = tri ptr = *38EE9C + 48*i

https://gist.github.com/dkorpel/dc7c435bf937fe886b67bfb51b7ec43a
Assembly hacking

- Write my own language?

- LDC has `-march=mips -mcpu=mips3`

- And `-output-s` flag

- How to resolve labels?
Linkers

- Relocation table
- Don't need assembler
- ELF binary format

"A linker is a very stupid, pedestrian, straightforward program. (...) The tedium in writing a linker is usually all about decoding and generating the usually ridiculously overcomplicated file formats" - Walter Bright

SHT_MIPS_ABIFLAGS
SHT_MIPS_REGINFO

Linkers

- Relocation table
- Don't need assembler
- ELF binary format

https://code.woboq.org/llvm/llvm/include/llvm/BinaryFormat/ELFRelocs/Mips.def.html
Result

- Very restricted feature set

- Still, benefits of using D over C:

syntax, reusable code, `import("data.bin")`

LLVM ERROR: Not supported instr:
`<MCInst 0 <MCOperand Reg:30> <MCOperand Reg:25>>`
D code

in the browser
D in the browser

- JavaScript
- Java Applet, Flash player
- Summer 2018: LDC adds WebAssembly
- betterC / Bare metal
- Ultimate tic-tac-toe game

https://github.com/dkorpel/tictac
Exact math calculator

- Simplify formulas with variables
- Mathematica not open source
- Write my own

SymPy is a Python library for symbolic mathematics. It aims to become a full-featured computer algebra system (CAS) while keeping the code as simple as possible in order to be comprehensible and easily extensible. SymPy is written entirely in Python.

Why SymPy

SymPy is...

- **Free**: Licensed under BSD, SymPy is free both as in speech and as in beer.
- **Python-based**: SymPy is written entirely in Python and uses Python for its language.
- **Lightweight**: SymPy only depends on mpmath, a pure Python library for arbitrary floating point arithmetic, making it easy to use.
- **A library**: Beyond use as an interactive tool, SymPy can be embedded in other applications and extended with custom functions.
Generic number types

- D has operator overloading
- Drop-in replacement for *float*

```d
t[2][2] matrixInverse2x2(T)(T[2][2] c)
{
    const T det = c[0][0] * c[1][1] - c[0][1] * c[1][0];
    const T invDet = 1 / det;
    T[2][2] res;
    res[0][0] = c[1][1] * invDet;
    res[0][1] = -c[0][1] * invDet;
    res[1][0] = -c[1][0] * invDet;
    res[1][1] = c[0][0] * invDet;
    return res;
}
```
Generic number types

- D has operator overloading
- Drop-in replacement for `float`
- Value type + tree data structure

```d
void main() {
    float x = 3;
    float y = x;
    y++;
    assert (x == 3);
}
```

\[(x+3)^2\]
Generic number types

- D has operator overloading
- Drop-in replacement for float
- Value type + tree data structure
- Inspired by BigInt and "Invariant strings"

```d
struct MathNum {
  int tag;
  immutable(MathNum)[] args;
}
```

https://digitalmars.com/articles/b01.html
WebAssembly

- I use function attributes a lot
WebAssembly

- I use function attributes a lot
- But this one uses GC
- Remove \texttt{-betterC} flag?
- druntime not ready

https://forum.dlang.org/post/xkedsnvgejvupjzpkur@forum.dlanq.org
WebAssembly

- Solution: custom druntime
- GitHub: adamdruppe/webassembly
- Pass custom object.d to compiler

 ldc2 -i -mtriple=wasm32-unknown-unknown-wasm -L-allow-undefined \
  wasm/object.d app.d \ 
  -of=app.wasm

Result

- Almost a programming language
- Leaks memory
- Limited implicit conversions

```d
float[] array = [10, 20];
MathNum[] arrayB = [MathNum(10), MathNum(20)];
```
D code on the graphics card
(I didn't run D code on the graphics card)
Graphics cards

- GPUs have no defined ISA
- APIs: DirectX, OpenGL, Vulkan

x86

Wasm

MIPS

SPIR-V
Using OpenGL

- Inform GPU about struct layout

```d
struct Vertex
{
    float[3] position;
    ubyte[4] color;
}
```

```glsl
#version 330
layout(location = 0) in vec3 position;
layout(location = 1) in vec4 color;

out vec3 fragColor;

uniform mat4 matrix;

void main()
{
    gl_Position = matrix * vec4(position, 1.0);
    fragColor = color;
}
```
Using OpenGL

- Inform GPU about struct layout
- Very brittle

```c
void setupVao()
{
    GLuint vao;
    glGenVertexArrays(1, &vao);
    glBindVertexArray(vao);

    glEnableVertexAttribArray(0);
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 16, 0);
    glEnableVertexAttribArray(1);
    glVertexAttribPointer(1, 4, GL_UNSIGNED_BYTE, GL_TRUE, 16, 12);
}

struct Vertex
{
    float[3] position;
    ubyte[4] color;
}
```
Oops!
Dennis Korpel
Metaprogramming

- Avoid magic numbers

```d
struct Vertex {
    float[3] position;
    ubyte[4] color;
}

glVertexAttribPointer (/*location*/ 0,
    /*components*/ Vertex.position.length,
    /*base type*/ GL_FLOAT,
    /*normalized*/ GL_FALSE,
    /*stride*/ Vertex.sizeof,
    cast(void*) Vertex.position.offsetof);
```
Metaprogramming

- Avoid magic numbers
- Generate the right calls

```
import std.meta, std.traits;

void defineVaoAttributes(V)()
{
    static foreach(i; 0 .. V.tupleof.length)
    {
        alias Arr = typeof(V.tupleof[i]);
        alias Elem = typeof(Arr.init[0]);
        glVertexAttribPointer(i, Arr.length, toGl!Elem, isIntegral!Elem,
                            V.sizeof, cast(void*) V.tupleof[i].offsetof);
    }
}

enum toGl(T) = [GL_UNSIGNED_BYTE, GL_FLOAT][staticIndexOf!(T, ubyte, float)];
```
Result

- Expand to support all types
- Automatically generate GLSL code
Result

- Expand to support all types
- Automatically generate GLSL code
- Still some friction
Wrapping up

- ❤️ to other programming languages
- Just one guy's perspective
- D is complex
- Reusable, robust, flexible code that runs everywhere
The Jack of all trades