Origins of the D Programming Language

Marius Kleppe Larnøy

May 2021

This essay presents the paper Origins of The D Programming Language [1] by Walter Bright, Andrei Alexandrescu and Michael Parker, submitted for HOPL IV. The paper tells the story of Walter Bright's work in the compiler industry from the late 70s up to the inception of the D programming language in 1999. It then covers the development of the language leading up to its 1.00 release in 2007.

The reason I chose this paper is because I was fascinated by the bold decision to attempt to create a successor to two of the most widely used programming languages in history, namely C and C++. Drawing users from two mature languages with codebases decades old is no easy task, and I was interested in learning about what features the D language could offer that would make a reasonably conservative user base of systems developers jump ship.

1 Brief Overview of the HOPL Paper

The Origins of the D Programming Language covers Walter Bright's two decade long journey in the compiler industry, as well as the first seven years of development on the D language.

The paper uses its first few pages to lay down the background for Walter Bright's decision to develop his own language. How he started out as a mechanical engineer at Boeing with game development as a hobby, who wrote his own C compiler to improve performance of his games, to working full time developing and maintaining industry grade compilers for C, C++ and later Java and JavaScript[1].

The main sections of the paper details the first few years of development on the Digital Mars D compiler, Digital Mars being the company Walter founded for the project. How it started out as a one man project and how it grew into a sizable group of people with community driven projects. It tells the story of how Walter Bright as the sole developer interacted with his growing (and vocal) community, and how his vision of the D specification not always matched what his target demographic wanted out of a systems programming language.

The Digital Mars D compiler version 1.00 was released on January 2nd 2007, and with that the paper concludes. "Any tale of *origins* of the D programming language must reasonably end with the release of version 1.00"[1].

2 Brief Overview of D

D is a systems and application programming language. It has support for multiple programming paradigms, such as procedural, object-oriented and metaprogramming. In its current form (D2) D has added support for functional programming, as well as improved metaprogramming as part of the core language[1].

D follows C's "algol-like" syntax closely, with the intention of making it easy for C and C++ programmers to make the transition over to D. To further cater to C-programmers D is interface compatible with C, One one the core design goals of D was that *"a syntactical construction should have the same semantics in D as in C, such as the integer promotion rules, or fail to compile"*[1].

Perhaps the feature that separates D from C and C++ the most is its inclusion of a garbage collector. A controversial addition to a systems programming language aimed at C developers, and heavily debated on the Dlang forums[1]. Until the addition of the `@nogc` function attribute in D2, automatic memory management was a fully embedded part of D.

D is a statically typed language[12], such that every expression has a type and typing errors are resolved at compile time. The language has its own list of basic built in data types, such as numeric types. In addition to the basic types there is also the derived data types, such as pointers, functions and

arrays. D also supports user-defined types such as enums, classes, structs and interfaces. (full overview can be found in the language specification at `dlang.org`).

3 Template mixins: An Overview

Template mixins are an extension to D's templates. According to the paper, mixins were first suggested to solve issues surrounding using macros in C++ to circumvent issues with the C preprocessor in the codebase of the first Unreal game[1]. Semantically, mixins were designed to avoid the problems of the preprocessor, while allowing exposure of metaclass information, i.e passing around a `classref*` which exposes static functions and constructors[1]. When the `mixin` keyword is prefixed to a template instantiation, the body of the template is inserted into that location and takes on the scope in which it is initialized. This is the opposite of what happens when you initialize a template normally, where the body takes on the scope in which it was implemented[1].

According to the specification *"A TemplateMixin takes an arbitrary set of declarations from the body of a TemplateDeclaration and inserts them into the current context"*[13].

Mixing templates comes wrapped in its own scope, with internal symbols aliased to the external scope[1]. This allows for multiple mixins of the same template in the same module, or mixins with several templates with the same internal symbols or possibly conflicting symbol names already in scope.

The example below from the paper highlights the use of two mixins of the same template, with identifiers `v1` and `v2` to disambiguate between them.

```
template addVars(T) {
    T x;
    T y;
    T z;
}
mixin addVars!float v1;
mixin addVars!double v2;

void main() {
    v1.x = 10; v2.x = 20;
}
```

4 Related Work on D

4.1 Related HOPL papers

Out of all the papers submitted to HOPL IV, the paper on D is mainly connected to Bjarne Stroustrup's paper *Thriving in a crowsed and changing world: C++ 2006-2020* [2]. As D was intended as an improvement upon C and C++ with similar syntax to draw users already fluent in those languages, D will inherently be connected to the history and development of C++.

There is one other paper from HOPL IV that has some connection to D, and that is *JavaScript: the first 20 years* [3]. The garbage collector originally employed in the D runtime library was a repurposed garbage collector that Walter Bright had written for the Chromium JavaScript implementation in 2000 [1].

Jumping back in time to the previous HOPL conferences, there are some papers in particular that stand out in regards to relevance for the D language. For HOPL-III, Bjarne Stroustrup wrote a paper on the history of C++ from 1991-2006, *Evolving a language in and for the real world: C++ 1991-2006* [4]. This period of time is particularily connected to the development of D as it was during this time D was initially concieved, and the D language's featureset was strongly influenced by the versions of C++ that emerged during this era.

Out of the HOPL-II papers, there are three that can be connected to D in some way. HOPL-II was the first installment that had C++ on its list of papers. In *A history of C++: 1978-1991*[5], Bjarne Stroustrup goes into detail about the first 13 years of C++ existence. This initial era of C++ was when Walter Bright began working on compilers for C and C++, most famously the Zortech C++ compiler [1], and it played a crucial role in his later work on the D compiler. Dennis Richie also submitted a paper on C for the conference, *The development of the C language* [6].

The last papers that have some relevance to the paper on D are the papers on ALGOL 60[9][8] and ALGOL 68[7]. This is more of an "honorable mention" as the connection is transitive. The term "algol-like" syntax spread

from ALGOL to C, to C++, and finally to D, and so beyond that direct impact ALGOL had on the development of D is neglicable.

4.2 Related work and further reading

The full language specification for D, as well as the documentation for the standard runtime library are available at dlang.org[10][11].

Andrei Alexandrescu's book *The D Programming Language*[15] is recommended by the D Language Foundation as "the definitive book on D"[14]. One drawback of the book is that at the time of the release in 2010 the current iteration of D (D2) was not yet feature complete [1], and thus it does not cover the complete specification of modern D.

A more recent book by Ali Çehreli published in 2016 [16] is also recommended by the D Language Foundation. It is a comprehensive introduction to the language aimed at both beginners and more experienced programmers. It also has the benefit of having a online version available which receives updates as the language evolves.

5 Tool Support for D

The tools I decided on using for D programming was VSCode with the code-d extension[17]. code-d requires you to have a D compiler installed beforehand, but offers a nice suite of helpful tools like syntax highlighting, auto-complete and linting. Compiling can be done from the VSCode editor or by running the compiler from a terminal window. I am also aware of that there exists a D-mode for Emacs and a D Plugin for IntelliJ IDEA, but I have not personally tested these out.

There is also a choice between different compilers to use. DMD is the official reference compiler, which features the latest version of D. GDC is a GCC based D compiler which offers lots of optimizations and support for a great variety of architectures. Lastly there is LDC, a LLVM based D compiler, which also offers lots of optimizations. For working on this project I decided to use DMD for the ease of use, but all compilers are viable based on your needs.

6 Personal Experience

At first glance, the motivation behind developing a successor to two languages such as C and C++ can be a bit hard to grasp. The paper does a good job in inviting the reader into the mindset of Walter Bright, and getting to know his motivations behind this decision. Once the paper reaches the beginning of D development in the late 90s, the daunting task seemed more like a natural step in the evolution of C-like languages.

True to its intentions, with some knowledge about C and C++ both reading and writing D code is in my opinion a close to seamless transition. It showcases alot of well-thought-out - sometimes subtle - changes that can take some time getting used to, but overall an enjoyable experience.

Questions to the authors:

- Part of the initial motivation behind D was that C++ was becoming exceedingly large. With D becoming a more mature language with support for multiple paradigms, how to avoid D running into the same issues?
- Where do you see D in 10-20 years?

References

[1] W. Bright, A. Alexandrescu, M. Parker, *Origins of the D Programming Language*, Proc. ACM Program. Lang., Vol. 4, No. HOPL, Article 73.

[2] B. Stroustrup, *Thriving in a crowded and changing world: C++ 2006-2020*, Proc. ACM Program. Lang., Vol. 4, No. HOPL, Article 70.

[3] A. Wirfs-Brock, B. Eich, *JavaScript: the first 20 years*, Proc. ACM Program. Lang., Vol. 4, No. HOPL, Article 77.

[4] B. Stroustrup, *Evolving a language in and for the real world: C++ 1991-2006*, HOPL III: Proceedings of the third ACM SIGPLAN conference on History of programming languages, Pages 4-1–4-59.

[5] B. Stroustrup, *A history of C++: 1979–1991*, HOPL-II: The second ACM SIGPLAN conference on History of programming languages, Pages 271–297.

[6] D. Richie, *The development of the C language*, HOPL-II: The second ACM SIGPLAN conference on History of programming languages, Pages 201-208.

[7] C. H. Lindsey, *A History of ALGOL 68*, HOPL-II: The second ACM SIGPLAN conference on History of programming languages, Pages 97-132.

[8] A. J. Perlis, *The American side of the development of Algol*, ACM SIGPLAN Not. 13, 8, pp 3-14.

[9] P. Naur, *The European side of the last phase of the development of ALGOL 60*, ACM SIGPLAN Not. 13, 8, pp 15-44.

[10] D Lang Foundation, *The D Language specification*, available at: `dlang.org`.

[11] D Lang Foundation, *Phobos Runtime Library*, available at: `dlang.org`.

[12] D Lang Foundation, *Types*, available at: `dlang.org`

[13] D Lang Foundation, *Template Mixins*, available at: `dlang.org`

[14] D Lang Foundation, *Books*, available at: `dlang-wiki`

[15] A. Alexandrescu, *The D Programming Language*, Addison-Wesley Professional, 2010. ISBN-13: 978-0321635365.

[16] A. Çehreli, *Programming in D*, Ali Çehreli, 2016. ISBN-13: 978-0692599433. Available at: `ddili.org`

[17] Webfreak, *D Programming Language(code-d)*, `github`