
DConf
auto event = 2013.menlo!park();

Prehistory

- A hypothetical successor to C named “D” was talked about on Usenet back in the 80's.
- Nothing much ever came of it

History

- Work began on D in late 1999
- D's first slashdot appearance
  - http://developers.slashdot.org/story/01/08/15/234223/the-d-programming-language
- D1 released Jan 2007
- First D Conference 2007

D Today

- Rapid and dramatic development since D1
- International, worldwide development community
- We're doing the impossible - developing a major new language from grassroots support
- And what amazing grassroots support it is ...

The D Programming Language Conference 2013

by Walter Bright

Funded! This project successfully raised its funding goal on Nov 21.

Support

DConf 2013 - a conference uniting all D programming language enthusiasts

Launched: Oct 22, 2012
Funding ended: Nov 21, 2012

DCONF 2013: THE D CONFERENCE

D is a powerful programming language. And it's up and coming stronger than ever. So we're bringing a bunch of awesome D contributors and users together to talk about D and its uses, potential, and future.

155
backers

$30,855
pledged of $29,999 goal

0
seconds to go

Project by
Walter Bright
Bothell, WA
Contact me

First created · 0 backed

Has not connected Facebook

Log in with Facebook

Website: dlang.org

See full bio

Pledge $10 or more

29 backers

EL POBRE ESTUDIANTE. You're a poor student or D enthusiast with a small wallet but big hopes. Express your belief in the future of D by

Major Sponsors

- Facebook
- Sociomantic
- Remedy Games
- Andrew Edwards

Today

- Copy And Move Semantics
  - Ali Çehreli
- Distributed Caching Compiler for D
  - Robert Schadek
- Inside Regular Expressions
  - Dmitry Olshansky
- Using D Alongside a Game Engine
  - Manu Evans
- Concurrent Garbage Collection
  - Leandro Lucarella

Tomorrow

- GDC
  - Ian Buclaw
- Shared Libraries
  - Martin Nowak
- C# to D
  - Adam Wilson
- Web Development in D
  - Vladimir Panteleev
- A Precise Garbage Collector for D
  - Rainer Schütze
- Higgs, an Experimental JIT Compiler in D
  - Maxime Chevalier-Boisvert
- Falling Down: the birth of Åkerön
  - Andrew Edwards

Friday

- Metaprogramming in the Real World
  - Don Clugston
- Code Analysis for D with AnalyzeD
  - Stefan Rohe
- D-Specific Design Patterns
  - David Simcha
- LDC
  - David Nadlinger
- Effective SIMD for modern architectures
  - Manu Evans
- Writing Testable Code
  - Ben Gertzfield
- Quo Vadis?
  - Andrei Alexandrescu

Vision

- Easy to read & understand code
- Provably correct
- Industrial Quality

Easy To Read & Understand

- code that looks right is right
- minimal boilerplate
- code looks like the problem being solved

Code That Looks Right Is Right

- and code that looks wrong is wrong

Pre-Scope Version of file.read()

```
void[] read(string name)  {
    DWORD numread;
    auto namez = toMBSz(name);
    auto  h=CreateFileA(namez,GENERIC_READ,FILE_SHARE_READ,null, OPEN_EXISTING,
            FILE_ATTRIBUTE_NORMAL | FILE_FLAG_SEQUENTIAL_SCAN,cast(HANDLE)null);
    if (h == INVALID_HANDLE_VALUE) goto err1;
    auto size = GetFileSize(h, null);
    if (size == INVALID_FILE_SIZE)  goto err2;
    auto buf = std.gc.malloc(size);
    if (buf)  std.gc.hasNoPointers(buf.ptr);
    if (ReadFile(h,buf.ptr,size,&numread,null) != 1)  goto err2;
    if (numread != size) goto err2;
    if (!CloseHandle(h)) goto err;
    return buf[0 .. size];

err2:
    CloseHandle(h);
err:
    delete buf;
err1:
    throw new FileException(name, GetLastError());
}
```

Using Scope – No Goto's

```
void[] read(in char[] name, size_t upTo = size_t.max) {
    alias TypeTuple!(GENERIC_READ,
            FILE_SHARE_READ, (SECURITY_ATTRIBUTES*).init, OPEN_EXISTING,
            FILE_ATTRIBUTE_NORMAL | FILE_FLAG_SEQUENTIAL_SCAN,
            HANDLE.init)
        defaults;
    auto h = CreateFileA(toMBSz(name), defaults);

    cenforce(h != INVALID_HANDLE_VALUE, name);
    scope(exit) cenforce(CloseHandle(h), name);
    auto size = GetFileSize(h, null);
    cenforce(size != INVALID_FILE_SIZE, name);
    size = min(upTo, size);
    auto buf = GC.malloc(size, GC.BlkAttr.NO_SCAN)[0 .. size];
    scope(failure) delete buf;

    DWORD numread = void;
    cenforce(ReadFile(h,buf.ptr, size, &numread, null) == 1
            && numread == size, name);
    return buf[0 .. size];
}
```

Minimal Boilerplate

“The IDE is great. With one key, I can add 100 lines of boilerplate!”

Looks Like the Problem Being Solved

```
import std.stdio;
import std.array;
import std.algorithm;

void main() {
    stdin.byLine(KeepTerminator.yes)
    map!(a => a.idup).
    array.
    sort.
    copy(
       stdout.lockingTextWriter());
}
```

Provably Correct

- Provable memory safety
- Provable purity and immutability
- Contract programming

  – No 'faith based' programming

Provable Memory Safety

- Memory safety means “no memory corruption”
- enabled with @safe attribute
  - works by disallowing things like pointer arithmetic
- safety is transitive
  - turtles all the way down

Provable Purity and Immutability

- solid foundation for functional programming
- FP has excellent track record of producing robust, reliable programs
- major aid to understanding code

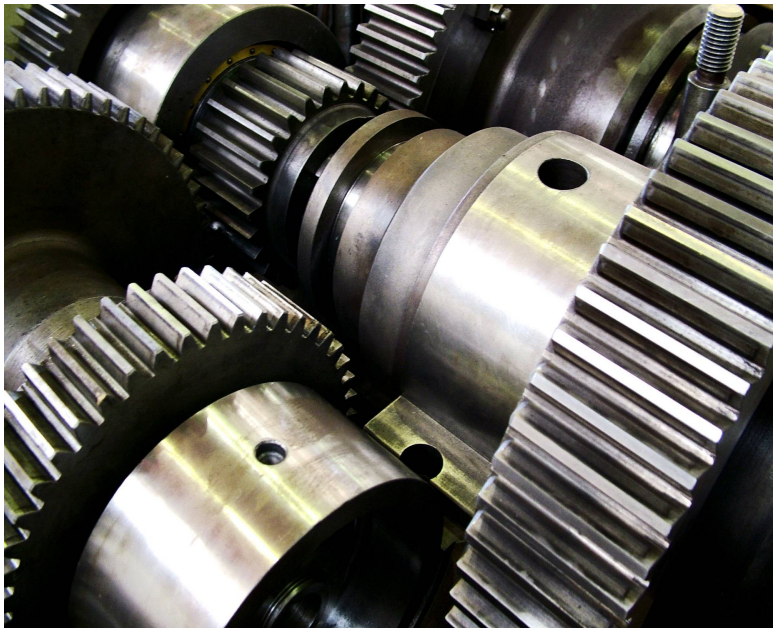
Toi Mine

Contract Programming

- some things can only be checked at runtime
- contracts are used to validate that assertions about data are true
  - (contracts are NOT for validating user input)
- can also be used by advanced optimizer to generate better code

Industrial Quality

- No-compromise performance
- Scales to enormous programs
- Management tools

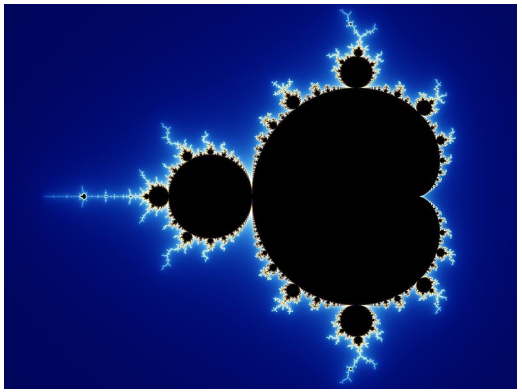


No Compromise Performance

- Semantics that map directly onto hardware
  - basic types are typical native machine types
  - even SIMD types
- Uses modern optimizing back ends
  - LLVM, GCC, Digital Mars
- Semantics amenable to powerful optimization

Scales to Enormous Programs

- Separate compilation
- Strong encapsulation semantics
  - no global namespace
  - anti-hijacking
  - voldemort types
- High speed compilation



Management Tools

- Documentation generation
- Unit test
- Coverage analysis

Conclusion

- D has fantastic support from the community
- Very strong technical content for this conference
- D will be the premier language for high performance high productivity computing