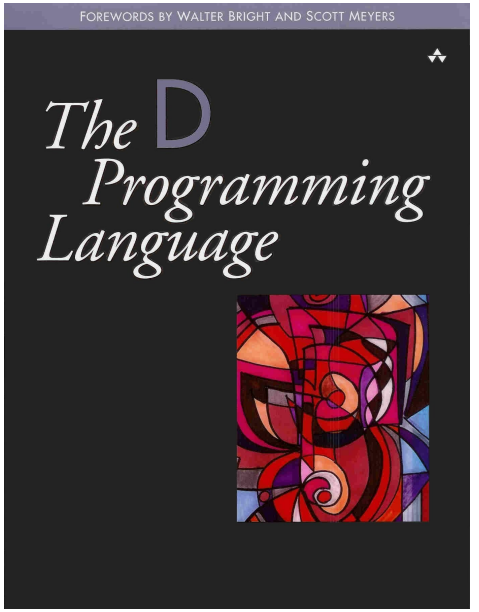
The D Programming Language

by Walter Bright
Digital Mars

http://www.digitalmars.com/d/

Signed by Andrei Alexandrescu,
3 copies,
for the most interesting comments
and questions.



"You want to go forward, what do you do? You put it in D."

http://voices.washingtonpost.com/44/2010/08/obamas-latest-joke-republicans.html

What is D?

- Systems and applications programming language
  - Native code generation
  - Static typing
  - Fast turnaround
- Born of decades of experience with industry projects
- Multi-paradigm

Multiparadigm

- C style
- Imperative
- Scripting
- Heavy Metal
- Object-oriented
- RAII
- Functional
- Generic
- Generative
- Concurrent
- Compile Time

C Style: Sieve in C

```
#include        <stdio.h>
#include        <stdlib.h>

const int true = 1;
const int false = 0;
const int size = 8190;
const int sizepl = 8191;

int main() {
    int i, prime, k, count, iter;

    printf("10 iterations\n");
    char *flags = (char *)malloc(sizepl);
```

```
    for (iter = 1; iter <= 10; iter++) {
        count = 0;
        for (i = 0; i <= size; i++)
            flags[i] = true;
        for (i = 0; i <= size; i++) {
            if (flags[i]) {
                prime = i + i + 3;
                k = i + prime;
                while (k <= size)  {
                    flags[k] = false;
                    k += prime;
                }
                count += 1;
            }
        }
    }
    free(flags);
    printf("\n%d primes", count);
    return 0;
}
```

Corresponding Sieve in D

```
import std.c.stdio;
import std.c.stdlib;

const int size = 8190;
const int sizepl = 8191;

int main() {
    int i, prime, k, count, iter;

    printf("10 iterations\n");
    char *flags = cast(char*) malloc(sizepl);
```

```
    for (iter = 1; iter <= 10; iter++) {
        count = 0;
        for (i = 0; i <= size; i++)
            flags[i] = true;
        for (i = 0; i <= size; i++) {
            if (flags[i]) {
                prime = i + i + 3;
                k = i + prime;
                while (k <= size)  {
                    flags[k] = false;
                    k += prime;
                }
                count += 1;
            }
        }
    }
    free(flags);
    printf("\n%d primes", count);
    return 0;
}
```

The Only Changes

- #include => import
- true and false are predefined
- (char *) => cast(char *)

ABI Compatible With C

- Can call any C function or library
- No translation layer
- Yes, that really is C's malloc/free/printf being called
- Writing C-style code in D will get same performance as in C
  - The same code is generated
- C and D code can be mixed and matched

Imperative Programming

```
import std.stdio;

void main() {
  int count;

  writeln("10 iterations");
  auto flags = new bool[8191];
```

```
 foreach (iter; 0 .. 10) {
     count = 0;
     flags[ ] = true;
     foreach (i, flag; flags) {
     if (flag) {
        auto prime = i + i + 3;
        for (auto k = i + prime;  k < flags.length;
             k += prime) {
             flags[k] = false;
         }
         ++count;
      }
    }
  }
  writefln("\n%s primes", count);
}
```

Notes

- Type inference
- Foreach ranges
- Foreach over arrays
- Automatic memory management
- Automatic error management
- Array operations
- Typesafe printing

Script

```
#!/usr/bin/rdmd

import std.stdio;

void main() {
    writeln("hello world!");
}
```

```
$ chmod u+x hello.d
$ ./hello.d
hello world!
$
```

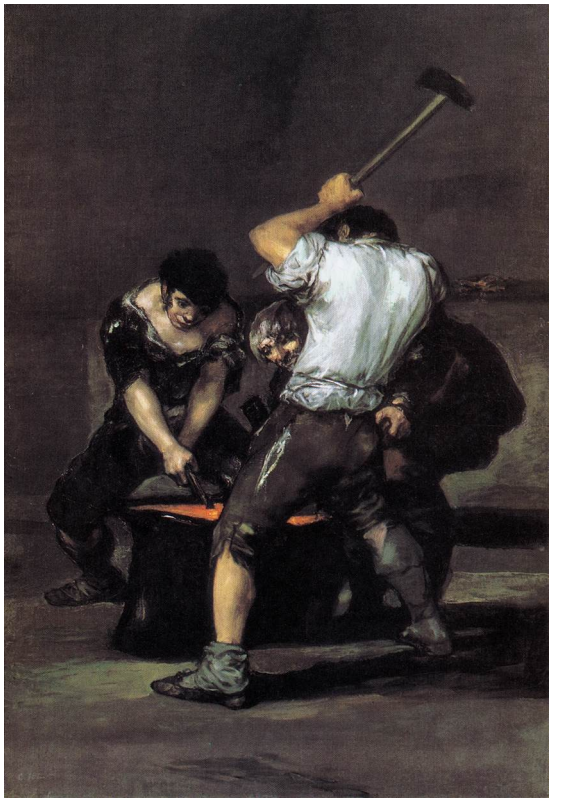
from The D Programming Language chapter 1

Notes

- Full power of D is available
- rdmd handles the compiling, linking
  - Binary is cached in temporary directory
  - Automatically recompiled if it changes
- Can completely replace use of bash, etc.

Heavy Metal Programming

```
int *_memset32(int *p, int value, size_t count) {
    asm   {
        mov     EDI,p          ;
        mov     EAX,value      ;
        mov     ECX,count      ;
        mov     EDX,EDI        ;
        rep                    ;
        stosd                  ;
        mov     EAX,EDX        ;
    }
}
```



Notes

- Function prolog/epilog added automatically
  - Including which registers to save/restore
- Intel syntax
  - X86 syntax portable across 4 operating systems
- Stack variable addressing modes

Object Oriented

```
class Shape {
  abstract void Draw ();
}

class Square : Shape {
  this(int x, int y, int w) {
    xpos = x; ypos = y;
    width = w;
  }

  void Draw() {
    writefln("Drawing Square at (%s,%s), width %s\n",
      x, y, width);
  }

  private int x, y, width;
}
```

Notes

- Single inheritance with interfaces
- Objects are always accessed by ref
  - Always by reference (never by value)
  - Use structs for by value
- Compatible with COM
- Compatible with C++ single inheritance
- Destructors are always virtual

RAII

```
struct Buffer {
  this(size_t s) {
    buf = malloc(s)[0 .. s];
  }
  this(this) {
    auto p = malloc(buf.length)[0..buf.length];
    p[] = buf[];
    buf = p;
  }
  ~this() {
    free(buf.ptr);
  }
  void[] buf;
}
```

Notes

- Used for value types (structs)
- Can be used to implement storage allocation
  - Including ref counting
- Unlike C++, values can be moved in memory
- Postblit is used to “adjust” things after a move
- No inheritance of structs, no virtual members
- opAssign is synthesized by compiler
  - (if not provided)

Functional

```
pure sum_of_squares (immutable double[] a) {
    auto sum = 0;
    foreach (i; a)
        sum += i * i;
    return sum;
}
```

Notes

- Immutability and purity are cornerstones of functional programming
- Of course, nested functions, lambdas and closures are supported

Compile Time

```
string decimaldigit(int n) {
    return "0123456789"[n..n+1];
}

string uitoa(uint n) {
    if ( n < 10 )
        return decimaldigit(n);
    else
        return uitoa( n / 10 ) ~ decimaldigit( n % 10 );
}
```

```
string showHowMany(int n, string where, bool needcapital = false) {
  if ( n > 1 )
     return uitoa(n) ~ " bottles of beer" ~ where ~ "\n";
  else if ( n == 1 )
     return "1 bottle of beer" ~ where ~ "\n";
  else if ( needcapital )
     return "No more bottles of beer" ~ where ~ "\n";
  else
     return "no more bottles of beer" ~ where ~ "\n";
}
```

```
string beer(int maxbeers, int n = -1) {
  if (n < 0)
    n = maxbeers;
  if ( n > 0 )
    return showHowMany(n, " on the wall,", true)
        ~ showHowMany(n, ".")
        ~ "Take one down and pass it around, " ~ "\n"
        ~ showHowMany( n - 1 , " on the wall.")
        ~ "\n" ~ beer(maxbeers, n - 1);
  else
    return showHowMany(n, " on the wall,", true)
        ~ showHowMany(n, ".")
        ~ "Go to the store and buy some more, " ~ "\n"
        ~ showHowMany( maxbeers, " on the wall.");
}

pragma(msg, beer(99));
```

Notes

- Operates completely at compile time
- No executable generated
- No libraries used
- Code is regular functions
- Functions must be pure
  - No side effects

Generic

```
size_t levenshteinDistance
  (alias equals = "a == b", Range1, Range2)
  (Range1 s, Range2 t)
  if (isForwardRange!Range1 &&
      isForwardRange!Range2){
  ...
}
```

Notes

- Works with arbitrary predicates
  - Function literals
  - Delegates
  - Strings
- Lightweight concepts in the form of template constraints
  - Anything that can be expressed as code that can be executed at compile time can form a constraint

Generative

```
struct A {
   int a;
   mixin(bitfields!(
       uint, "x",    2,
       int,  "y",    3,
       uint, "z",    2,
       bool, "flag", 1));
}
A obj;
obj.x = 2;
obj.z = obj.x;
```

Notes

- Makes use of compile time execution to form strings, which are then “mixed in”
- Mixed in strings are D code
- Of course, the strings and the D code they contain can be arbitrarily complex

Concurrent

```
import std.algorithm, std.concurrency, std.stdio;

void main() {
 enum bufferSize = 1024 * 100;
 auto tid = spawn(&fileWriter);
 // Read loop
 foreach (immutable(ubyte)[] buffer;
          stdin.byChunk(bufferSize)) {
   send(tid, buffer);
 }
}

void fileWriter() {
 // Write loop
 for (;;) {
   auto buffer = receiveOnly!(immutable(ubyte)[])();
   stdout.write(buffer);
 }
}
```

from The D Programming Language chapter 13

Notes

- Message passing the preferred method
- Can also do:
  - Shared memory
  - Lock free
  - Atomics, etc.

Compilers

- Digital Mars D compiler
  - Based on the Digital Mars compiler suite
- Gnu D compiler
  - Based on the gnu compiler collection
- LDC compiler
  - Based on the LLVM compiler

Full source code available for all of them

Platform Support

- Windows
- Linux
- OS X
- FreeBSD

Conclusion

- D supports many styles of programming
- Non-trivial programs rarely fit nicely into one paradigm
- No need to produce hybrid programs using multiple languages
- Investment in libraries becomes reusable across a wide range of applications