D2

Karsten Heinze

karsten@sidenotes.de

09.01.2014

Where C++ is both powerful and complicated, D aims to be at least as powerful but less complicated.

Scott Meyers

1. Geschichte
2. Sprache
3. Compiler
4. Projekte
5. Literatur

Geschichte

Walter Bright

D1 (199?)

D2 (2006)

Sprache

D is a language that attempts to consistently do the right thing within the contstraints it chose: ***system-level access*** *to computing resources,* ***high performance****, and* ***syntactic similarity*** *with C-derived languages.*

Andrei Alexandrescu

Hello, world!

```
#!/usr/bin/env rdmd


import std.stdio;

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

Basic data types

Keyword	Default Initializer (.init)	Description
void	-	no type
bool	false	boolean value
byte	0	signed 8 bits
ubyte	0	unsigned 8 bits
short	0	signed 16 bits
ushort	0	unsigned 16 bits
int	0	signed 32 bits
uint	0	unsigned 32 bits
long	0L	signed 64 bits
ulong	0L	unsigned 64 bits
cent	0	signed 128 bits (reserved for future use)
ucent	0	unsigned 128 bits (reserved for future use)
float	float.nan	32 bit floating point
double	double.nan	64 bit floating point
real	real.nan	largest FP size implemented in hardware
ifloat	float.nan*1.0i	imaginary float
idouble	double.nan*1.0i	imaginary double
ireal	real.nan*1.0i	imaginary real
cfloat	float.nan+float.nan*1.0i	a complex number of two float values
cdouble	double.nan+double.nan*1.0i	complex double
creal	real.nan+real.nan*1.0i	complex real
char	0xFF	unsigned 8 bit (UTF-8 code unit
wchar	0xFFFF	unsigned 16 bit (UTF-16 code unit)
dchar	0x0000FFFF	unsigned 32 bit (UTF-32 code unit)

Dynamic Arrays

```
string[] dynamic;   // or = new string[ 0 ]

writefln( "dynamic has %s elements",
    dynamic.length );

// Iterate over elements of range [1,6)
foreach ( i; 1 .. 6 ) {
    dynamic ~= to!string( i );
}

writefln( "dynamic has %s elements: %s",
    dynamic.length, join( dynamic, ", " ) );
```

```
$ ./example02_arrays.d
dynamic has 0 elements
dynamic has 5 elements: 1, 2, 3, 4, 5
```

Array slicing

```
int[64] elems;
auto slice = elems[ 8 .. $ ];
auto copy = slice.dup

writefln( "%s elems [%s,%s)",
     elems.length, elems.ptr,
     elems.ptr + elems.length );
writefln( "%s elems [%s,%s)",
     slice.length, slice.ptr,
     slice.ptr + slice.length );
writefln( "%s elems [%s,%s)",
     copy.length, copy.ptr,
     copy.ptr + copy.length );
```

```
$ ./example03_slices.d
64 elems [7FFF0AEB5110,7FFF0AEB5210)
56 elems [7FFF0AEB5130,7FFF0AEB5210)
56 elems [7FC315710E00,7FC315710EE0)
```

Associative Arrays

```
int[ string ] count;
string s = "Wenn Hessen in Essen ...";

foreach_reverse ( w; split( s, " " ) ) {
    try {
        ++count[ w ];
    } catch ( RangeError e ) {
        count[ w ] = 1;
    }
}
foreach ( w, c; count ) {
    writefln( "%10s => %s", w, c );
}
```

```
$ ./example04_associative.d
    Essen. => 1
        in => 2
     Essen => 3
    Hessen => 2
     essen => 1
    essen, => 1
      Wenn => 1
```

Templates

```
T convert( T, S )( S value )
{
    return to!T( value );
}

void main()
{
  uint value = 1;
  string asString = convert!string( value );

  // Notice the is expression!
  writefln( "Converted to string? Mmh, %s!",
    is( typeof( asString ) == string ) );
}
```

```
$ ./example05_templates.d
Converted to string? Mmh, true!
```

Classes and structs

reference semantics

vs.

value semantics

```
class C {
    @property uint data1, data2;

    this( uint data1 ) {
        this.data1 = data1;
    }
    this( uint data1, uint data2 ) {
        this( data1 );
        this.data2 = data2;
    }

}

struct S
{
    uint data;
}
```

```
C cObj = new C( 0 );
S sObj;

C cObj2 = cObj;    // like a ref in C++
S sObj2 = sObj;

++cObj2.data1;
++sObj2.data;

writefln( "%s %s", &cObj, &cObj2 );
writefln( "%s %s", cObj.data1, cObj2.data1 );
writefln( "%s %s", &sObj, &sObj2 );
writefln( "%s %s", sObj.data, sObj2.data );
```

```
$ ./example06_classes.d
7FFF281B6850 7FFF281B6860
1 1
7FFF281B6858 7FFF281B6868
0 1
```

Unittests

```
mixin template Hey() {
   string ho() { return "Heyho!"; }
}
class C { mixin Hey; }

unittest {
   C c = new C();
   assert( c !is null );
   assert( c.ho() == "Heyho!" );
}

void main() {
   C c = new C(); writefln( "%s", c.ho() );
}
```

```
$ dmd -unittest -ofunittest example07_unittests.d
$ ./unittest
Heyho!
$ diff example07_unittests.d example07_unittests_failure.d
15c15
<   assert( c !is null );
---
>   assert( c is null );
$ dmd -unittest -offailing_unittest example07_unittests_fai
$ ./failing_unittest
core.exception.AssertError@example07_unittests_failure(15):
----------------
./failing_unittest(_d_unittestm+0x47) [0x477fa7]
./failing_unittest(void example07_unittests_failure.__unitt
./failing_unittest(void example07_unittests_failure.__unitt
...
```

OOD/OOP

packages and modules

interfaces

abstract and final classes

...

Parameter passing

```
// in (default)
int increment( /* in */ int i );

// out
bool increment( int i, out int o );

// ref (in and out)
bool increment( ref int v );
```

String literals

```
// WYSIWYG
auto a = r"str with " `"` " and " "
writeln( a );

auto u8 = "abc";
auto u16 = "abc"w;
auto u32 = "abc"d;

writefln( "%s < %s < %s",
    u8.length * u8[ 0 ].sizeof,
    u16.length * u16[ 0 ].sizeof,
    u32.length * u32[ 0 ].sizeof
  );
```

```
$ ./example08_string_literals.d
A string with " and `.
3 < 6 < 12
```

scope statement

```
void main()
{
   {
      scope( exit ) writeln( "fourth" );
      writeln( "first" );
      scope( exit ) writeln( "third" );
      scope( failure ) writeln( "never" );
      scope( success ) writeln( "second" );
   }
}
```

```
$ ./example09_scope.d
first
second
third
fourth
```

Was gibts noch?

- contracts
- conditional compilation
- compile time reflection (traits)
- inline assembler
- garbage collection
- threads (message passing)
- delegates

Compiler

DMD (Digital Mars)

http://dlang.org

GDC (GCC frontend)

http://gdcproject.org

LDC (LLVM based)

https://github.com/ldc-developers/ldc

Projekte

http://vibed.org

web framework written in D

event driven, asynchronous I/O

clean interface

lean programming experience

http://github.com/karheinz/pird

audio cd ripper written in D

uses libcdio

accurate rip support

rips also cd images

Literatur

Programming D

Andrei Alexandrescu, Addison-Wesley, 2010

Ende