D at 20
Hits and Misses

and a few things I learned about language design

by Walter Bright
D Language Foundation
dlang.org

No Preprocessor

Unicode all the way

Code pages, EBCDIC, Shift-JIS, etc., should all be processed as ubyte arrays, not char arrays

Miss: Agnostically Supporting UTF-16 and UCS-2

Turns out they're sideshows.
UTF-8 is the one.

32 bit machines or greater

- C is best for 16 bit machines
- Even C++ doesn't work for 16 bit machines
  - no exception handling
  - no RTTI
- so why pretend?

2's Complement Arithmetic

Seriously, I've never even seen a 1's complement machine in 45 years.

Miss: bit Data Type

```
bit b, c, d;
bit* p = &b; // !!!!
```

Having more than one pointer type is just a disaster.
I should have known better.

Fixed Integer Type Sizes

byte, short, int, long
ubyte, ushort, uint, ulong

Miss: Octal Integer Literals

Only good for two things:

- file permissions
- bugs

```
int i = 050;          // is that 40 or 50?

int i = octal!050; // definitely 40
```

https://dlang.org/phobos/std_conv.html#octal

Miss: Binary Literals

- invented them for Zortech C++
  - never used
- not easily discouraged, added to D
  - never used
- removed from D
- added to C++14 !!

IEEE 748 Floating Point

- Few understand floating point
- Fewer have a chance trying to write portable floating point
- Nobody wants to be bothered with these problems

Miss: 80 bit floating point

- Unable to convince people that more precision is worthwhile
- Neglected / abandoned by hardware

Miss: Complex Floating Point Type

- moved to library
- https://dlang.org/phobos/std_complex.html

Slicing

- Fixes C's biggest mistake
  - https://digitalmars.com/articles/b44.html

```
char[11] array = "hello world";
char[ ] slice = array[1 .. 5]
assert(slice == "ello");
```

Strings are Arrays

- No special string type!

https://dlang.org/articles/d-array-article.html

Miss: Then We Botched It

- autodecoding the strings
  - sometimes it decodes code units into code points
  - sometimes it does not
- still trying to dig our way out of that

No Bit Fields

```
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;
writeln(obj.x); // 2
writeln(obj.y); // 0
writeln(obj.z); // 2
```

https://dlang.org/phobos/std_bitmanip.html#bitfields

Built In Unittests

```
int add(int x, int y) {
    return x + y;
}

unittest {
    assert(add(3, 5) == 8);
}
```

https://dlang.org/spec/unittest.html

Built In Documentation Generation

```
/**************************
 * Adds the operands
 * Params:
 *   x = first operand
 *   y = second operand
 * Returns:
 *   sum of `x` and `y`
 */
int add(int x, int y) {
    return x + y;
}
```

https://dlang.org/spec/ddoc.html

Compile Time Function Execution

```
int square(int I) {
    return i * i;
}

void foo() {
    static j = square(3);     // CTFE
    writeln(j);
    assert(square(4));        // run time
    enum s = square(5);   // CTFE
    writeln(s);
}
```

https://dlang.org/spec/function.html#interpretation

Easy Template Syntax

```
int square(int i) { return i * i; }

T square(T)(T i) { return i * i; }
```

https://dlang.org/spec/template.html#function-templates

Modules

```
import core.stdc.stdio;
```

https://dlang.org/spec/module.html

C Compatibility

```
import core.stdc.stdio;

int main() {
    printf("hello world\n");
    return 0;
}
```

https://digitalmars.com/articles/betterC.html

Uniform Function Call Syntax

```
int cook(X x);
int eat(int i);

X x;
i = x.cook();
i = x.cook;
i = x.cook.eat;
```

https://dlang.org/spec/function.html#pseudo-member

Miss: Safety should be default, not opt-in

- D has always had an emphasis on memory safety, but I underestimated how strong the desire for it is.
- The current default is for system code. We'll be changing it to safe code.

static if

```
const int i = 3;

void func() {
   static if (i == 3) {
       int x;
   }
   bar();
   static if (i == 3) {
       x += 1;
   }
}
```

https://dlang.org/spec/version.html#staticif

Template Constraints

```
void foo(int N)()
    if (N & 1)
{
    ...
}
...
foo!(3)();  // OK, matches
foo!(4)();  // Error, no match
```

https://dlang.org/spec/template.html#template_constraints

Scope Guard

```
Transaction transaction() {
  Foo f = dofoo();
  scope(failure) dofoo_undo(f);

  Bar b = dobar();
  return Transaction(f, b);
}
```

https://dlang.org/articles/exception-safe.html

Transitive const and immutable

Necessary for functional programming,
and (future) ownership / borrowing system

shared As A Type Constructor

```
int* p;               // pointer to thread local int
shared(int)* p; // pointer to shared int
```

Miss: Postblit

```
struct S {
    int[ ] a;
    this(this) {
        a = a.dup;
    }
}
```

https://dlang.org/spec/struct.html#struct-postblit

Pure Functions

```
int x;

pure int foo(int i) {
    return i + x;   // error: access to global `x`
}
```

Along with const, can be used to do functional programming.

https://dlang.org/spec/function.html#pure-functions

Miss: Emphasis on GC

- GC is very good for batch programs, scripts, compile time function execution, memory safety
- Not so good for interactive programs because of pause time
- Uses 3x the memory of manual management

struct / class Dichotomy

- no more it's a floor wax / it's a dessert topping
- structs are value types
- classes are reference types

Miss: Contracts

- Preconditions
- Postconditions
- class / struct Invariants

https://dlang.org/spec/contracts.html

debug Keyword

```
debug printf("we got this far\n");
```

https://dlang.org/spec/version.html#debug

Miss: Exceptions are the default

- They're not zero cost, and never were, even when exceptions are not happening
  - expensive to insert unwinding code, needed or not
  - optimizations are disabled
- nothrow should be the default

Deprecation

```
deprecated void oldFeature();

oldFeature(); // Error: oldFeature is deprecated
```

https://dlang.org/spec/attribute.html#deprecated

Miss: Allow Throwing Destructors

- at risk for the dreaded double-fault exception
- destructors should be nothrow

Learned

- simple syntax wins
- convenience wins
- built-in wins over third party tool
- everybody hates bloat
- everybody loves colored error messages

dlang.org