Leverage

Andrei Alexandrescu

Research Scientist

Facebook

Why D?

— Gilad Bracha

ⓒ 2014 Andrei Alexandrescu

Why?

- Party line
  - convenience
  - modeling power
  - efficiency
- Actual reasons
  - Produces fast binaries, fast
  - Easier to get into than alternatives
  - Fun

Why not D?

— Charles Torre

Why not?

- Party line
  -
- Actual reasons
  - Poor on formal specification
  - Little corporate pickup, support
  - Dearth of libraries
  - Large

Gently provoke.

— Mads Torgersen

© 2014 Andrei Alexandrescu

Andrei’s Conjecture

Many language design decisions look goofy if efficiency is not a concern

Dual of Andrei's Conjecture

Many language design decisions look great if efficiency is not a concern

Turtles all the way down

Hello, World!

```
#!/usr/bin/rdmd
import std.stdio;
void main() {
  writeln("Hello, world!");
}
```

- “Meh”worthy
- However:
  - Simple
  - Correct
  - Scriptable
  - Features turtles

Them turtles

```
#!/usr/bin/rdmd
void main() {
  import std.stdio;
  writeln("Hello, world!");
}
```

- Most everything can be scoped everywhere
- Better scoping, reasoning, dependency mgmt
- Functions
- Types (Voldermort types)
- Even generics

© 2014 Andrei Alexandrescu

Segue into generics

```
void log(T)(T stuff) {
  import std.datetime, std.stdio;
  writeln(Clock.currTime(), ' ', stuff);
}
void main() {
  log("hello");
}
```

- If not instantiated, no `import`
- `import`s cached once realized
- Generics faster to build, import
- Less pressure on linker

Universal Function Call Syntax

Simple Lowering

```
expr1.fun(expr2, expr3)
```

- If symbol `fun` found within `typeof(expr1)`'s scope, proceed
- Otherwise, rewrite as:

```
fun(expr1, expr2, expr3)
```

- Hat tip to Cecil

Benefits

- Allow extending types non-invasively
- Allow extending built-in types
    - UTF string iteration implemented this way
- Great for ”pipes and filters” programming
- Quickly won the minds and hearts of users
- Seamless ”default” for methods

ⓒ 2014 Andrei Alexandrescu

Drawbacks

- Occasional odd constructs

```
"%s %s".writefln("hello", 42)
```

- The whitespace that could

```
iota(10)
10.iota
iota(10.)
10..iota // error
10. .iota
```

Optional Trailing Parens

- If `fun` is a function, just `fun` evaluates `fun()`
- Also `obj.fun` evaluates `obj.fun()`
- To take the address thereof, place & before

+ Prints money
+ Great with pipes and filters
− Pascal haters gonna hate
− Ambiguity: what if a function/method returns a function?

Example

```
import std.algorithm, std.stdio, std.range, std.conv;
void main() {
  stdin
    .byLine()
    .filter!(s => !s.empty && s.front != '#')()
    .map!(s => to!double(s))()
    .array()
    .sort!((a, b) => a < b)()
    .take(10)
    .writeln();
}
```

Example

```
import std.algorithm, std.stdio, std.range, std.conv;
void main() {
  stdin
    .byLine
    .filter!(s => !s.empty && s.front != ’#’)
    .map!(s => s.to!double)
    .array
    .sort!((a, b) => a < b)
    .take(10)
    .writeln;
}
```

Static Introspection

Static Introspection

- Unique to D?
- Enumerate module members
- Enumerate `struct`/`class` fields
- Query type, attributes, qualifiers, . . .

+ Terse, simple, systematic genericity
+ Astonishing code leverage
– Cognitive load
– Delayed/nonstandard dynamic introspection

Example

```
void scan(T)(T* obj) if (is(T == struct)) {
  if (!obj) return;
  if (!heap.markAsUsed(obj)) return;
  foreach (ref f; obj.tupleof) {
    alias F = typeof(f);
    static if (hasIndirections!F)
      scan(f);
  }
}
```

Compile-Time Evaluation

Compile-Time Evaluation

- Can evaluate entire program fragments during compilation
- Same code as regular runtime code
- Even create strings, arrays, and ”allocate” objects
- Even compile strings computed thusly

$+$ No need to learn a different sublanguage
$+$ ”Free” computation
$+$ DSLs
$-$ May increase compilation time ($\leq \infty$)

`static if`

`static if`

- Statically include/exclude code depending on Boolean expressions
- . . . that can be computed during compilation
- . . . using introspection and stuff

+ Would you write ordinary code without `if`?

– ”Race conditions” during compilation

They Work *Together*

Example

```
bool reallocate(Allocator)(ref Allocator a, ref void[] b, size_t s)
{
    if (b.length == s) return true;
    static if (hasMember!(Allocator, "expand"))
    {
        if (b.length <= s && a.expand(b, s - b.length)) return true;
    }
    auto newB = a.allocate(s);
    if (newB.length <= b.length) newB[] = b[0 .. newB.length];
    else newB[0 .. b.length] = b[];
    static if (hasMember!(Allocator, "deallocate"))
    {
        a.deallocate(b);
    }
    b = newB;
    return true;
}
```

© 2014 Andrei Alexandrescu

Summary

Summary

- General theme is *leverage*
- Static code molding ⇒ lossless code compression
- Feature interoperation is key

- Generic + static introspection = generative

© 2014 Andrei Alexandrescu