D as a Better C

Simon Arneaud

https://theartofmachinery.com/
C and C++ are powerful languages for systems programming
D is not a perfect alternative

(https://wiki.dlang.org/Language_issues)
But D is better for many applications, today

(and getting even better, fast)
Super Short History of C

- Dennis Ritchie, et al, needed a high-level language to rewrite Unix in ~1970
- "The C Programming Language" ("K&R C") published in 1978
- <blink>NOT THE FIRST LANGUAGE AFTER ASSEMBLY</blink>
- Originally only intended for the PDP–11, which heavily influenced design
Super Short History of C++

• Bjarne Stroustrup wanted a fast and convenient language

• Made original "C with Classes" in 1979

• Initial implementation literally just a preprocessor for C

• Language has since diverged (slightly) from C
Super Short History of D

- Walter Bright was the author of Zortech, the first native C++ compiler, and is responsible for several advances in C++ compiler technology since then
- D1 released in 2001
- D2 released in 2007
- First DConf in 2013
Why a new language?
Why a "better C", not "better C++"?
C++'s biggest strength and weakness:

backwards compatibility with C
Why do we need a better C?
enum reactor_id
{
    REACTOR_A = 1,
    REACTOR_B = 2,
};

enum reactor_mode
{
    OFF = 0,
    NORMAL = 1,
    EXPERIMENTAL = 2, // TODO: Delete this. Too dangerous. Really bad idea.
};

reactor.id = REACTOR_B;
reactor.mode = REACTOR_B;
0[foo_array]
if (is_ready);
{
    launchMissile();
}
**const** and **volatile**

**volatile** is a broken mess

**const** is not so useful as a compiler hint

In C, pointers to pointers (e.g., arrays of strings) are broken (and unsound)
Lack of low-level systems stuff in standard
Preprocessor includes instead of modules

Generally have to be re-evaluated every time thanks to side effects

Leads to hacks like "inline variables" in C++

(Try \texttt{gcc -E} or \texttt{clang -E} for fun sometime)
What about C++?
Compilation times
Compiling C++:

Here[ not] be dragon[ book]s
Result doStuff(Message);
Thing thing(config);

std::map<int, std::pair<int, int>>
val>>2
struct Base
{
    void doStuff(double x)
    {
        std::cout << "Got a double: " << x << std::endl;
    }
};

struct Derived : Base
{
    void doStuff(int x)
    {
        std::cout << "Got an int: " << x << std::endl;
    }
};

...
*Batteries not included
Legacy

(E.g., enum vs enum class)
Low road:  
- C strings  
- Preprocessor  
- C I/O  

High road:  
- C++ strings  
- Templates  
- C++ I/O  

Integrates easily with C  
"Impedance mismatch" with C  

(This is why the term "C/C++" is kind of silly.)
### C Strings

| ) | H | e | l | l | o | \0 | @ | # | z |

- Null-terminated arrays
- Memory management totally manual
- Mutable reference types
- Must recalculate string length whenever needed
- Substrings usually need to be copies
C++ Strings

Implementation-defined templated class

GNU libstdc++ has

- Length
- Capacity
- Reference count
- Data pointer

Generally

- Memory management controlled by string class
- Mutable value types
D Strings

- Slices (pointer + length)
- BYO memory management
- Immutable reference types
Case Study

• Preprocess text file
• Answer queries

Approach #1:

1. Read file one buffer load at a time
2. Construct (copy) strings from buffer to insert into data structure
3. Run query loop
Case Study

- Preprocess text file
- Answer queries

Approach #2:

1. Memory map file (std.mmfile)
2. Slice strings without copying
3. Run query loop
Not only was the initialisation much faster, the main query loop was \(\sim 10\text{-}20\%\) faster thanks to better memory locality.
Could it work with C strings?

- Can fully control the memory layout
- Still need a copy to insert null bytes
- Copied and mutated data isn't implicitly shared with OS or other processes
Could it work with C++ strings?

• Short answer: no

• Longer answer: can partially control memory layout using a custom allocator, but this changes the string type

• Still can't use the memory-mapped file data effectively
What about D's overhead?
And who is this D. Runtime, anyway?
import io = std.stdio;

class Greeter
{
    void greet()
    {
        io.writeln("Hello");
    }
}

void main()
{
    auto greeter = new Greeter();
    greeter.greet();
}
import io = std.stdio;

class Greeter
{
    void greet()
    {
        io.writeln("Hello");
    }
}

void main()
{
    auto greeter = new Greeter();
    greeter.greet();
}

1. My code
import io = std.stdio;

class Greeter
{
    void greet()
    {
        io.writeln("Hello");
    }
}

void main()
{
    auto greeter = new Greeter();
    greeter.greet();
}

1. My code
2. Imports
import io = std.stdio;

class Greeter
{
    void greet()
    {
        io.writeln("Hello");
    }
}

void main()
{
    auto greeter = new Greeter();
    greeter.greet();
}

1. My code
2. Imports
3. Compiler-generated code
import io = std.stdio;

class Greeter
{
    void greet()
    {
        io.writeln("Hello");
    }
}

void main()
{
    auto greeter = new Greeter();
    greeter.greet();
}

1. My code
2. Imports
3. Compiler-generated code
4. Runtime library
• Garbage collection
• `Object` (Base class of all D classes)
• Initialisation/cleanup of modules and static data
• Associative arrays
• Operations like struct equality and array copying
• Threads and TLS
• Run-time type information (`TypeInfo`)

(Having a runtime isn't just a D thing.)
It is not running in the background.
long factorial(int n)
{
    long f = 1;
    int j;
    for (j = 1; j <= n; j++)
    {
        f *= j;
    }
    return f;
}

In a single-threaded program, this is all that's running.
BTW, this is colloquially known as "Better C"
A subset of D with no D runtime dependencies
(I.e., all C−like code + some other features)
Garbage collection can only happen on GC-based allocation

(or explicit GC.collect())

I.e., normal systems programming idioms like allocating up front will avoid GC pauses
import core.memory;

void main()
{
    GC.disable();
    doSomethingLatencySensitive();
    GC.enable();

    // About to do lots of memory-hungry stuff
    // so improve performance by reserving GC memory up front
    GC.reserve(1024 * 1024 * 1024);
    doSomethingMemoryHungry();
    GC.minimize();

    message.sendToSomeServer();
    GC.collect();  // Might as well run GC while waiting
    waitForResponseFromSomeServer();
}
D supports

- Static allocation
- Stack allocation
- Plain-old heap allocation
- Garbage-collected heap allocation
- BYO memory with `emplace`

(No, really, it all works.)
"But I'm doing systems programming so I can't use the runtime."
NB: This concern is not just about ricing performance.
"Why should I care? I have a multicore machine with several gigs of RAM and terabytes of storage. Most of the D runtime is in a shared library, anyway."
"I want to write mobile browser code in D. I can't use shared libraries, and every downloaded kilobyte counts."
Different applications have different needs.
More Case Studies
tsv-utils-dlang

https://github.com/eBay/tsv-utils-dlang

The "Keep Calm and Write Sensible Code" approach

- Tools for processing delimited text files (CSV, TSV, etc)
- Made by Jon Degenhardt for data mining at eBay
- Did not worry about avoiding features like GC
- Performance due to common sense like avoiding redundant copying and allocating
<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Tool/Time</th>
<th>Tool/Time</th>
<th>Tool/Time</th>
<th>Tool/Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Numeric row filter</td>
<td><code>tsv-filter</code></td>
<td>mawk</td>
<td>GNU awk</td>
<td>Toolkit 1</td>
</tr>
<tr>
<td>(4.8 GB, 7M lines)</td>
<td>4.34</td>
<td>11.71</td>
<td>22.02</td>
<td>53.11</td>
</tr>
<tr>
<td>Regex row filter</td>
<td><code>tsv-filter</code></td>
<td>GNU awk</td>
<td>mawk</td>
<td>Toolkit 1</td>
</tr>
<tr>
<td>(2.7 GB, 14M lines)</td>
<td>7.11</td>
<td>15.41</td>
<td>16.58</td>
<td>28.59</td>
</tr>
<tr>
<td>Column selection</td>
<td><code>tsv-select</code></td>
<td>mawk</td>
<td>GNU cut</td>
<td>Toolkit 1</td>
</tr>
<tr>
<td>(4.8 GB, 7M lines)</td>
<td>4.09</td>
<td>9.38</td>
<td>12.27</td>
<td>19.12</td>
</tr>
<tr>
<td>Join two files</td>
<td><code>tsv-join</code></td>
<td>Toolkit 1</td>
<td>Toolkit 2</td>
<td>Toolkit 3</td>
</tr>
<tr>
<td>(4.8 GB, 7M lines)</td>
<td>20.78</td>
<td>104.06</td>
<td>194.80</td>
<td>266.42</td>
</tr>
<tr>
<td>Summary statistics</td>
<td><code>tsv-summarize</code></td>
<td>Toolkit 1</td>
<td>Toolkit 2</td>
<td>Toolkit 3</td>
</tr>
<tr>
<td>(4.8 GB, 7M lines)</td>
<td>15.83</td>
<td>40.27</td>
<td>48.10</td>
<td>62.97</td>
</tr>
<tr>
<td>CSV-to-TSV</td>
<td><code>csv2tsv</code></td>
<td>csvtk</td>
<td>xsv</td>
<td></td>
</tr>
<tr>
<td>(2.7 GB, 14M lines)</td>
<td>27.41</td>
<td>36.26</td>
<td>40.40</td>
<td></td>
</tr>
</tbody>
</table>
Mir numerical library

https://github.com/libmir/mir

The "D as a Better C" approach

- Collection of numerical libraries in D (think BLAS, NumPy) by Ilya Yaroshenko
- Uses `-betterC` flag and avoids D runtime features
- Mir GLAS can be linked to plain C code as BLAS implementation
- High performance through solid engineering and effective use of CPU features like SIMD
Auburn Sounds

https://www.auburnsounds.com/index.html

The `@nogc` approach

- Commercial audio plugins in D
- Mostly relies on `@nogc` for latency-sensitive code

Alternative: put audio handling in thread detached from GC (see `core.thread`)
PowerNex

https://github.com/Vild/PowerNex

https://dlang.org/blog/2016/06/24/project-highlight-the-powernex-kernel/

The stub runtime approach

- An x64 OS project in D started by Dan Printzell
- Ports a minimal subset of the D runtime to bare metal (based on package by Adam Ruppe)
- Intended to eventually support a complete D development environment
Xanthe

https://gitlab.com/sarneaud/xanthe

https://theartofmachinery.com/2017/02/28/bare_metal_d.html

The horrible hacks approach

- Short vertical-scrolling shooter game demo that boots on bare metal x86
- Freestanding D
  - No D runtime
  - No C runtime
  - No OS
Even more case studies:

Weka.IO

Distributed data storage system

https://www.youtube.com/watch?v=q7wyQHF6SXY

Vibe.d

Event-loop-based web (and network) application framework

https://dlang.org/blog/2017/03/01/project-highlight-vibe-d/
Questions?

Simon Arneaud

https://theartofmachinery.com/
enquiries@taom.systems