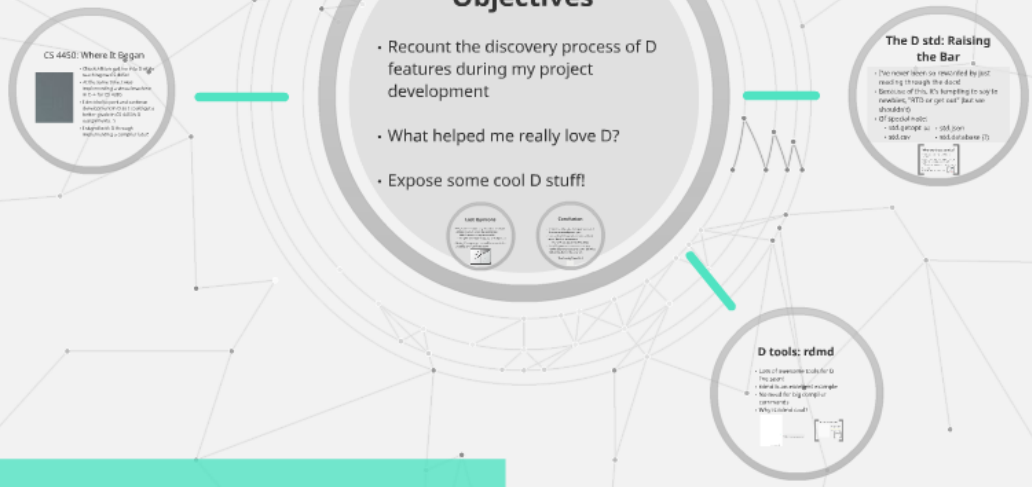
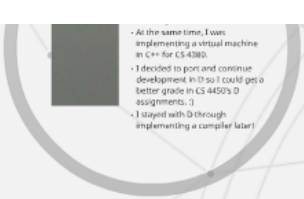


Why I Love D: An Undergrad Experience

Erich Gubler

Prezi



development

- What helped me really love D
- Expose some cool D stuff!

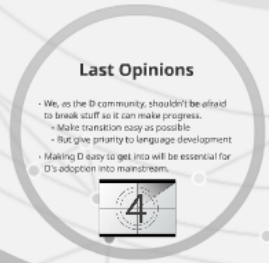
Why I Love D: An Undergrad Experience

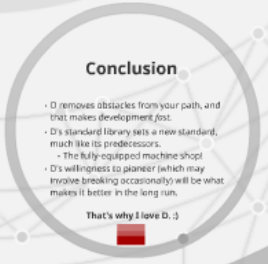
Erich Gubler

Prezi

Objectives

- Recount the discovery process of D features during my project development
- What helped me really love D?
- Expose some cool D stuff!





Prezi

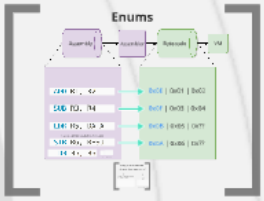
CS 4450: Where It Began

- Chuck Allison got me into D while teaching me CS 4450!
- At the same time, I was implementing a virtual machine in C++ for CS 4380.
- I decided to port and continue development in D so I could get a better grade in CS 4450's D assignments. :)
- I stayed with D through implementing a compiler later!

Prezi

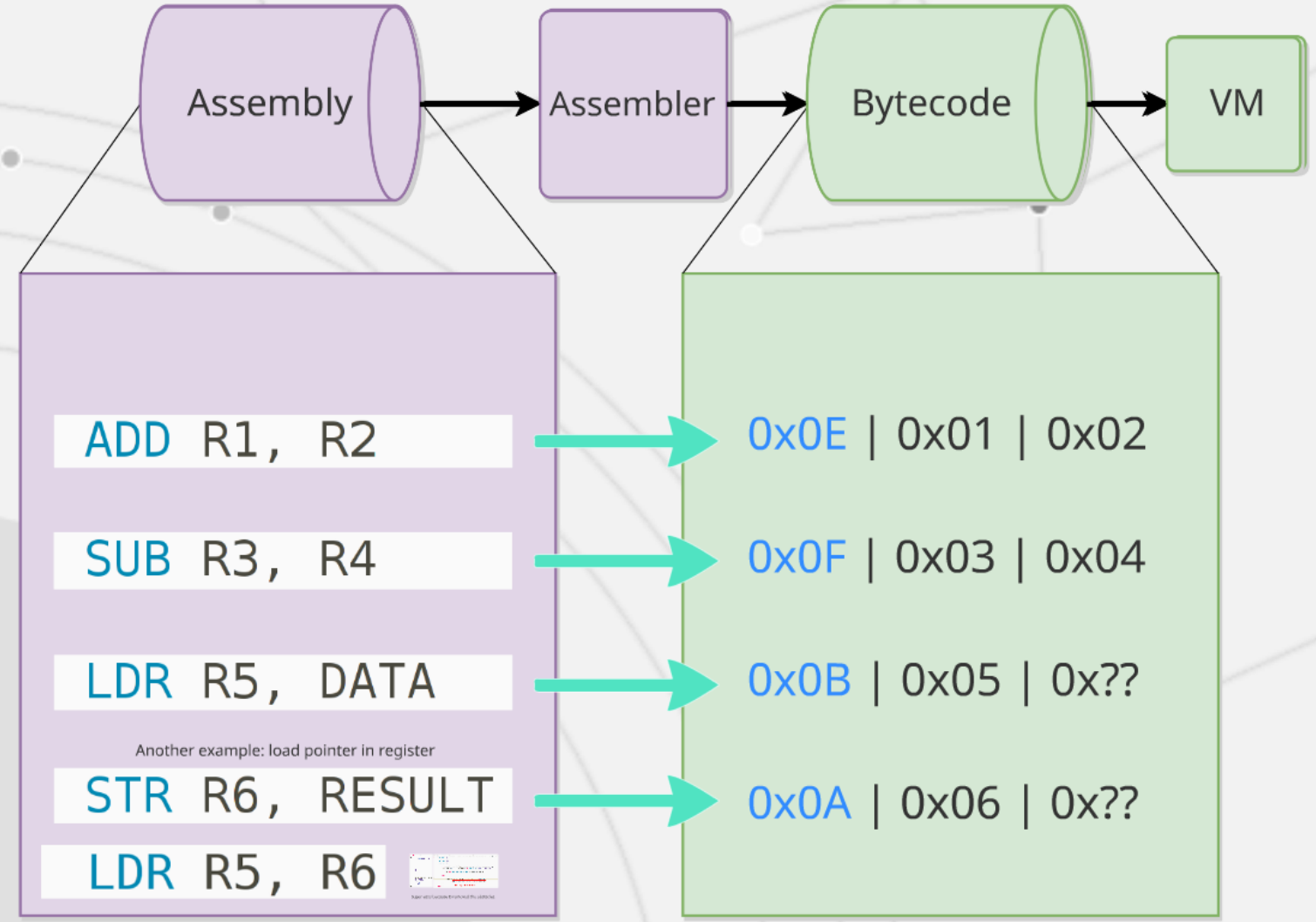
Porting Over to D

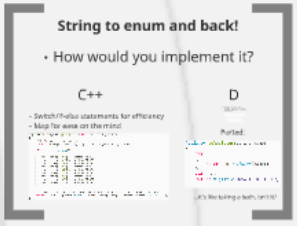
- No more *.h and *.cpp
- "I don't have to deal with pointers anymore?"
- Pointers are like an unsheathed knife in your pants pocket...
- No more namespace operator!
- 'Twas a snap! Mostly **stripping** stuff.



Prezi

Enums


Assembly
Assembler
Bytecode
VM
ADD R1, R2
SUB R3, R4
LDR R5, DATA
Another example: load pointer in register
STR R6, RESULT
LDR R5, R6
0x0E | 0x01 | 0x02
0x0F | 0x03 | 0x04
0x0B | 0x05 | 0x??
0x0A | 0x06 | 0x??


String to enum and back!
• How would you implement it?
C++
D

Prezi

String to enum and back!

- How would you implement it?

C++

- Switch/if-else statements for efficiency
- Map for ease on the mind

```
CapstoneVM::Register getRegister(const std::string &name)
{
    //Singleton-pattern goodness
    static std::map<std::string, CapstoneVM::Register> registerMap;

    if(registerMap.empty())
    {
        registerMap["R0"] = CapstoneVM::R0;
        registerMap["R1"] = CapstoneVM::R1;
        registerMap["R2"] = CapstoneVM::R2;
        registerMap["R3"] = CapstoneVM::R3;
        registerMap["R4"] = CapstoneVM::R4;
        registerMap["R5"] = CapstoneVM::R5;
        registerMap["R6"] = CapstoneVM::R6;
        registerMap["R7"] = CapstoneVM::R7;
        registerMap["PC"] = CapstoneVM::PC;
    }

    std::map<std::string, CapstoneVM::Register>::iterator it = registerMap.find(name);
    return ((it != registerMap.end()) ? registerMap[name] : CapstoneVM::INVALID_REGISTER);
}
```

D

- Can use std.conv's "to" function because enums are smart in D

```
string s = //...
auto r = s.to!Register;

   r = Register.R1;
   s = r.to!string;
```

Ported:

```
Register toRegister(string name)
{
    try
    {
        return to!Register(name);
    }
    catch {}
    return Register.INVALID_REGISTER;
}
```

...it's like taking a bath, isn't it?

Prezi

- Switch/if-else statements for efficiency
- Map for ease on the mind

```
CapstoneVM::Register getRegister(const std::string &name)
{
    //Singleton-pattern goodness
    static std::map<std::string, CapstoneVM::Register> registerMap;

    if(registerMap.empty())
    {
        registerMap["R0"] = CapstoneVM::R0;
        registerMap["R1"] = CapstoneVM::R1;
        registerMap["R2"] = CapstoneVM::R2;
        registerMap["R3"] = CapstoneVM::R3;
        registerMap["R4"] = CapstoneVM::R4;
        registerMap["R5"] = CapstoneVM::R5;
        registerMap["R6"] = CapstoneVM::R6;
        registerMap["R7"] = CapstoneVM::R7;
        registerMap["PC"] = CapstoneVM::PC;
    }

    std::map<std::string, CapstoneVM::Register>::iterator it = registerMap.find(name);
    return ((it != registerMap.end()) ? registerMap[name] : CapstoneVM::INVALID_REGISTER);
}
```

Prezi

String to enum and back!

- How would you implement it?

C++

- Switch/if-else statements for efficiency
- Map for ease on the mind

```
CapstoneVM::Register getRegister(const std::string &name)
{
    //Singleton-pattern goodness
    static std::map<std::string, CapstoneVM::Register> registerMap;

    if(registerMap.empty())
    {
        registerMap["R0"] = CapstoneVM::R0;
        registerMap["R1"] = CapstoneVM::R1;
        registerMap["R2"] = CapstoneVM::R2;
        registerMap["R3"] = CapstoneVM::R3;
        registerMap["R4"] = CapstoneVM::R4;
        registerMap["R5"] = CapstoneVM::R5;
        registerMap["R6"] = CapstoneVM::R6;
        registerMap["R7"] = CapstoneVM::R7;
        registerMap["PC"] = CapstoneVM::PC;
    }

    std::map<std::string, CapstoneVM::Register>::iterator it = registerMap.find(name);
    return ((it != registerMap.end()) ? registerMap[name] : CapstoneVM::INVALID_REGISTER);
}
```

D

- Can use std.conv's "to" function because enums are smart in D

```
string s = //...
auto r = s.to!Register;

   r = Register.R1;
   s = r.to!string;
```

Ported:

```
Register toRegister(string name)
{
    try
    {
        return to!Register(name);
    }
    catch {}
    return Register.INVALID_REGISTER;
}
```

...it's like taking a bath, isn't it?

Prezi

D

- Can use std.conv's "to" function because enums are smart in D

```
string s = //...
auto r = s.to!Register;
```

```
r = Register.R1;
s = r.to!string;
```

Prezi

cy

- Can use std.conv's "to" function because enums are smart in D

```
string s = //...
auto r = s.to!Register;

r = Register.R1;
s = r.to!string;
```

Ported:

```
Register toRegister(string name)
{
    try
    {
        return to!Register(name);
    }
    catch {}
    return Register.INVALID_REGISTER;
}
```

me);
GISTER);

...it's like taking a bath, isn't it?

Prezi

C++

- Switch/if-else statements for efficiency
- Map for ease on the mind

```
CapstoneVM::Register getRegister(const std::string &name)
{
    //Singleton-pattern goodness
    static std::map<std::string, CapstoneVM::Register> registerMap;

    if(registerMap.empty())
    {
        registerMap["R0"] = CapstoneVM::R0;
        registerMap["R1"] = CapstoneVM::R1;
        registerMap["R2"] = CapstoneVM::R2;
        registerMap["R3"] = CapstoneVM::R3;
        registerMap["R4"] = CapstoneVM::R4;
        registerMap["R5"] = CapstoneVM::R5;
        registerMap["R6"] = CapstoneVM::R6;
        registerMap["R7"] = CapstoneVM::R7;
        registerMap["PC"] = CapstoneVM::PC;
    }

    std::map<std::string, CapstoneVM::Register>::iterator it = registerMap.find(name);
    return ((it != registerMap.end()) ? registerMap[name] : CapstoneVM::INVALID_REGISTER);
}
```

D

- Can use std.conv's "to" function because enums are smart in D

```
string s = //...
auto r = s.to!Register;
    r = Register.R1;
    s = r.to!string;
```

Ported:

```
Register toRegister(string name)
{
    try
    {
        return to!Register(name);
    }
    catch {}
    return Register.INVALID_REGISTER;
}
```

...it's like taking a bath, isn't it?

Prezi

```
LDR R5, DATA
```

Another example: load pointer in register

```
STR R6, RESULT
```

```
LDR R5, R6
```

Super easy because D removed the obstacles.

Prezi

```
public enum Opcode
{
    INVALID_OPCODE = -1,

    // ...

    STR,
    LDR,
    STB,
    LDB,

    // ...

    //Indirect load/stores
    STR_INDIRECT,
    LDR_INDIRECT,
    STB_INDIRECT,
    LDB_INDIRECT,
}
```

```
// Instruct
case Opcode
    , Opcoc
    , Opcoc
    , Opcoc
    {
        aut
        if(
        {

        }
        els
        {

        }
    }
    break;
```

Prezi

```
e

E = -1,

ad/stores
```

```
// Instructions that need some top-down parsing to decide what they need
case Opcode.STR
    , Opcode.LDR
    , Opcode.STB
    , Opcode.LDB:
    {
        auto match = matchAll(argumentString, doubleArgumentRegex);
        if(match.front[2].toRegister == Register.INVALID_REGISTER)
        {
            p = new RegisterLabelParser(argumentString);
        }
        else
        {
            oToUse = (o.to!string ~ "_INDIRECT").to!Opcode;
            p = new RegisterRegisterParser(argumentString);
        }
    }
    break;
```

This is pretty neat!

easy because D removed the obstacles.

Prezi

```d
public enum Opcode
{
    INVALID_OPCODE = -1,

    // ...

    STR,
    LDR,
    STB,
    LDB,

    // ...

    //Indirect load/stores
    STR_INDIRECT,
    LDR_INDIRECT,
    STB_INDIRECT,
    LDB_INDIRECT,
}
```

```d
// Instructions that need some top-down parsing to decide what they need
case Opcode.STR
    , Opcode.LDR
    , Opcode.STB
    , Opcode.LDB:
    {
        auto match = matchAll(argumentString, doubleArgumentRegex);
        if(match.front[2].toRegister == Register.INVALID_REGISTER)
        {
            p = new RegisterLabelParser(argumentString);
        }
        else
        {
            oToUse = (o.to!string ~ "_INDIRECT").to!Opcode;
            p = new RegisterRegisterParser(argumentString);
        }
    }
    break;
```

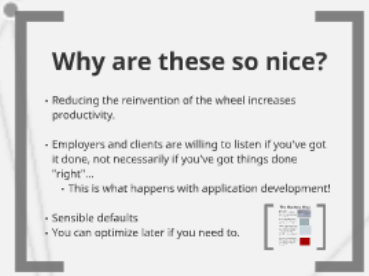
This is pretty neat!

Super easy because D removed the obstacles.

Prezi

The D std: Raising the Bar

- I've never been so rewarded by just reading through the docs!
- Because of this, it's tempting to say to newbies, "RTD or get out" (but we shouldn't)
- Of special note:
  - std.getopt
  - std.csv
  - std.json
  - std.database (?)



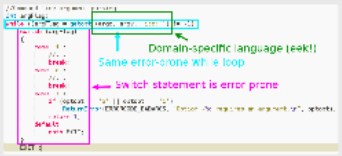
Prezi

std.getopt

This needs exposition! Some comments heard:

- "...that's standardized?"
- "That's cheating!"
- "No way..."

Using the original GNU getopt



As fast as prototyping

```d
import std.stdio;
import std.getopt;

int main(string[] args)
{
    // Opt variables
    string myString;
    bool verboseFlag = false;

    auto result = getopt(args
        , "s|myString", &myString // -s or --myString
        , "verboseFlag", &verboseFlag); // --verboseFlag=[...]

    writeln("myString: ", myString);
    writeln("verboseFlag: ", verboseFlag);

    writeln("args left: ", args);

    // Print help
    if(result.helpWanted)
        defaultGetoptPrinter("Help:", result.options);

    return 0;
}
```

Modular getopt calls

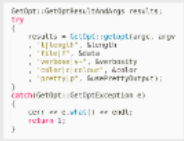
- IO

- Logging with channels

- Putting it all together

Where was this in C++?

- After using D's getopt, I nerdraged from not having something nice like it in C++...so I wrote it. :)
- https://github.com/ErichDonGubler/getopt



Prezi

Using the original GNU getopt

```
//Command-line argument parsing
int argFlag;
while ((argFlag = getopt (argc, argv, "i:o:")) != -1)
    switch (argFlag)
    {
        case 'i':
            //...
            break;
        case 'o':
            //...
            break;
        case '?':
            if (optopt == 'o' || optopt == 'i')
                ReturnError(ERRORCODE_BADARGS, "Option -%c requires an argument.\n", optopt);
            return 1;
        default:
            goto EXIT;
    }
    EXIT:;
```

Domain-specific language (eek!)

Same error-prone while loop

Switch statement is error prone

Prezi

As fast as prototyping

```d
import std.stdio;
import std.getopt;

int main(string[] args)
{
    // Opt variables
    string myString;
    bool verboseFlag = false;

    auto result = getopt(args
        , "s|myString", &myString // -s or --myString
        , "verboseFlag", &verboseFlag); // --verboseFlag=[...]

    writeln("myString: ", myString);
    writeln("verboseFlag: ", verboseFlag);

    writeln("args left: ", args);

    // Print help
    if(result.helpWanted)
        defaultGetoptPrinter("Help:", result.options);

    return 0;
}
```

Prezi

Modular getopt calls

- IO

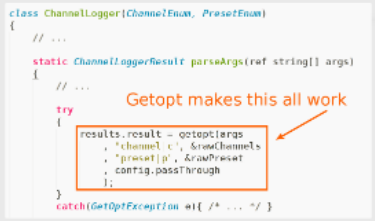
```d
/// Grabs a single output file and any number of input files from command args
IOOptResult parseIOFiles(ref string[] args)
{
    IOOptResult io;
    try
    {
        getopt(args
            , "outfile|o", &io.output
            , config.passThrough
            );
    }
    catch(GetOptException e){ /* ... */ }
    io.inputs = args[1..$];
    args.length = 1;
    return io;
}
```

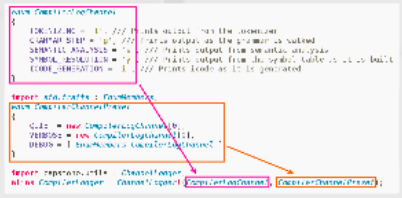
getopt call

Return data structure

- Logging with channels

```
./compiler --channel=tg
```





- Putting it all together

```d
void main(string[] args)
{
    auto logging = CompilerLogger.parseArgs(args);
    auto io = parseIOFiles(args);

    if(logging.result.helpWanted || io.result.helpWanted)
    {
        import std.getopt : defaultGetoptPrinter;
        defaultGetoptPrinter("Usage: " ~ args[0] ~ " <source file>", logging.result.options ~ io.result.options);
    }

    // Validate IO args
    if(io.inputs.length == 0)
        returnError!(ErrorLevel.BAD_ARGS)("no input file specified");
    else if(io.inputs.length > 1)
        returnError!(ErrorLevel.BAD_ARGS)("multiple input files not supported");
```

Call opt parsing modules

Use builtin help

Misc. validation

Reuse and factorization of getopt options = win!

Prezi

- IO

```d
/// Grabs a single output file and any number of input files from command args
IOOptResult parseIOFiles(ref string[] args)
{
    IOOptResult io;
    try
    {
        getopt(args
            , "outfile|o", &io.output
            , config.passThrough
            );
    }
    catch(GetOptException e){ /* ... */ }
    io.inputs = args[1..$];
    args.length = 1;
    return io;
}
```


Prezi

getopt calls

- Logging with channels

```
./compiler --channel=tg
```

```
class ChannelLogger(ChannelEnum, PresetEnum)
{
    // ...

    static ChannelLoggerResult parseArgs(ref string[] args)
    {
        // ...

        try
        {
            results.result = getopt(args
                , "channel|c", &rawChannels
                , "preset|p", &rawPreset
                , config.passThrough
                );
        }
        catch(GetOptException e){ /* ... */ }
```

Getopt makes this all work

```
enum CompilerLogChannel
{
    TOKENIZING = 't', /// Prints output from the tokenizer
    GRAMMAR_STEP = 'g', /// Prints output as the grammar is walked
    SEMANTIC_ANALYSIS = 'e', /// Prints output from semantic analysis
    SYMBOL_RESOLUTION = 'y', /// Prints output from the symbol table as it is built
    ICODE_GENERATION = 'i', /// Prints icode as it is generated
}

import std.traits : EnumMembers;
enum CompilerChannelPreset
{
    QUIET = new CompilerLogChannel[0],
    VERBOSE = new CompilerLogChannel[0],
    DEBUG = [ EnumMembers!CompilerLogChannel ]
}

import capstone.utils : ChannelLogger;
alias CompilerLogger = ChannelLogger!(CompilerLogChannel, CompilerChannelPreset);
```

Reuse and factorization of getopt options = win!

Prezi

ogging with channe

```
./compiler --channel=tg
```

```
annelEnum, PresetEnum)

erResult parseArgs(ref string[] args)
```

etopt makes this all work

```
sult = getopt(args
nnel|c", &rawChannels
set|p", &rawPreset
ig.passThrough
```

```
enum CompilerLogChannel
{
    TOKENIZING = 't', /// Prints output from the tokenizer
    GRAMMAR_STEP = 'g', /// Prints output as the grammar i
    SEMANTIC_ANALYSIS = 'e', /// Prints output from semant
    SYMBOL_RESOLUTION = 'y', /// Prints output from the sy
    ICODE_GENERATION = 'i', /// Prints icode as it is gene
}

import std.traits : EnumMembers;
enum CompilerChannelPreset
{
    QUIET = new CompilerLogChannel[0],
    VERBOSE = new CompilerLogChannel[0],
    DEBUG = [ EnumMembers!CompilerLogChannel ]
}

import capstone.utils : ChannelLogger;
alias CompilerLogger = ChannelLogger!(CompilerLogChannel,
```

```d
class ChannelLogger(ChannelEnum, PresetEnum)
{
    // ...

    static ChannelLoggerResult parseArgs(ref string[] args)
    {
        // ...

        try
        {
            results.result = getopt(args
                , "channel|c", &rawChannels
                , "preset|p", &rawPreset
                , config.passThrough
                );
        }
        catch(GetOptException e){ /* ... */ }
```

Getopt makes this all work

Prezi

--channel=tg

```d
enum CompilerLogChannel
{
    TOKENIZING = 't', /// Prints output from the tokenizer
    GRAMMAR_STEP = 'g', /// Prints output as the grammar is walked
    SEMANTIC_ANALYSIS = 'e', /// Prints output from semantic analysis
    SYMBOL_RESOLUTION = 'y', /// Prints output from the symbol table as it is built
    ICODE_GENERATION = 'i', /// Prints icode as it is generated
}

import std.traits : EnumMembers;
enum CompilerChannelPreset
{
    QUIET = new CompilerLogChannel[0],
    VERBOSE = new CompilerLogChannel[0],
    DEBUG = [ EnumMembers!CompilerLogChannel ]
}

import capstone.utils : ChannelLogger;
alias CompilerLogger = ChannelLogger!(CompilerLogChannel, CompilerChannelPreset);
```

Prezi

```
}
```

• Putting it all together

```d
void main(string[] args)
{
    auto logging = CompilerLogger.parseArgs(args);
    auto io = parseIOFiles(args);

    if(logging.result.helpWanted || io.result.helpWanted)
    {
        import std.getopt : defaultGetoptPrinter;
        defaultGetoptPrinter("Usage: " ~ args[0] ~ " <source file>", logging.result.options ~ io.result.options);
    }

    // Validate IO args
    if(io.inputs.length == 0)
        returnError!(ErrorLevel.BAD_ARGS)("no input file specified");
    else if(io.inputs.length > 1)
        returnError!(ErrorLevel.BAD_ARGS)("multiple input files not supported");
```

Call opt parsing modules

Use builtin help

Misc. validation

Prezi

Reuse and factorization of getopt options = win!

Prezi

Where was this in C++?

- After using D's getopt, I nerdraged from not having something nice like it in C++...so I wrote it. :)

- https://github.com/ErichDonGubler/getopt

```
GetOpt::GetOptResultAndArgs results;
try
{
    results = GetOpt::getopt(argc, argv
    , "l|length", &length
    , "file|f", &data
    , "verbose|v+", &verbosity
    , "color|c|colour", &color
    , "pretty|p", &usePrettyOutput);
}
catch(GetOpt::GetOptException e)
{
    cerr << e.what() << endl;
    return 1;
}
```

Prezi

```
GetOpt::GetOptResultAndArgs results;
try
{
    results = GetOpt::getopt(argc, argv
    , "l|length", &length
    , "file|f", &data
    , "verbose|v+", &verbosity
    , "color|c|colour", &color
    , "pretty|p", &usePrettyOutput);
}
catch(GetOpt::GetOptException e)
{
    cerr << e.what() << endl;
    return 1;
}
```

Prezi

Why are these so nice?

- Reducing the reinvention of the wheel increases productivity.
- Employers and clients are willing to listen if you've got it done, not necessarily if you've got things done "right"...
  - This is what happens with application development!
- Sensible defaults
- You can optimize later if you need to.

Prezi

The Machine Shop

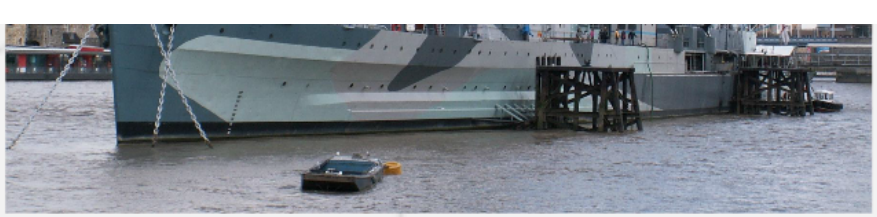
Quote from Walter:
When I was in London for the 2010 ACCU (when the volcano stranded me there), I took a chance to tour the Belfast cruiser sitting in the Thames. One interesting aspect of it was the ship's machine shop.

It was full of carefully selected machine tools. It was pretty clear to me that an expert machinist could quickly and accurately make or repair about anything that broke on that ship.

Sure, you can make do with fewer, more general purpose machines. But it'll take you considerably longer, and the result won't be as good. For example, I've used electric drills for years. I was

Prezi

the Thames. One interesting aspect of it was the ship's machine shop.

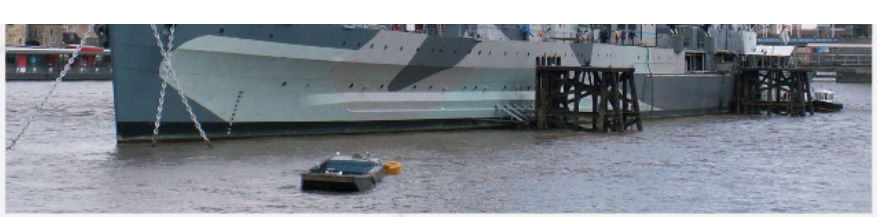


It was full of carefully selected machine tools. It was pretty clear to me that an expert machinist could quickly and accurately make or repair about anything that broke on that ship.

Sure, you can make do with fewer, more general purpose machines. But it'll take you considerably longer, and the result won't be as good. For example, I've used electric drills for years. I was never able to get it to drill a hole perfectly perpendicular. I finally got a drill press, and problem solved. Not only is it far more accurate, it's much faster when you've got a lot of holes to drill.

I prefer to view D as a fully equipped machine shop with the right tools for the right job. Yes, it

the Thames. One interesting aspect of it was the ship's machine shop.



It was full of carefully selected machine tools. It was pretty clear to me that an expert machinist could quickly and accurately make or repair about anything that broke on that ship.

Sure, you can make do with fewer, more general purpose machines. But it'll take you considerably longer, and the result won't be as good. For example, I've used electric drills for years. I was never able to get it to drill a hole perfectly perpendicular. I finally got a drill press, and problem solved. Not only is it far more accurate, it's much faster when you've got a lot of holes to drill.

I prefer to view D as a fully equipped machine shop with the right tools for the right job. Yes, it

Sure, you can make do with fewer, more general purpose machines. But it'll take you considerably longer, and the result won't be as good. For example, I've used electric drills for years. I was never able to get it to drill a hole perfectly perpendicular. I finally got a drill press, and problem solved. Not only is it far more accurate, it's much faster when you've got a lot of holes to drill.

I prefer to view D as a fully equipped machine shop with the right tools for the right job. Yes, it will take longer to master it than a simpler language. But we're professionals, we program all day. The investment of time to master it is trivial next to the career productivity improvement.

Prezi

The Machine Shop

Quote from Walter:
When I was in London for the 2010 ACCU (when the volcano stranded me there), I took a chance to tour the Belfast cruiser sitting in the Thames. One interesting aspect of it was the ship's machine shop.

It was full of carefully selected machine tools. It was pretty clear to me that an expert machinist could quickly and accurately make or repair about anything that broke on that ship.

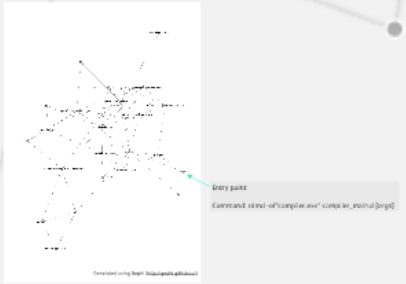
Sure, you can make do with fewer, more general purpose machines. But it'll take you considerably longer, and the result won't be as good. For example, I've used electric drills for years. I was never able to get it to drill a hole perfectly perpendicular. I finally got a drill press, and problem solved. Not only is it far more accurate, it's much faster when you've got a lot of holes to drill.

I prefer to view D as a fully equipped machine shop with the right tools for the right job. Yes, it will take longer to master it than a simpler language. But we're professionals, we program all day. The investment of time to master it is trivial next to the career productivity improvement.

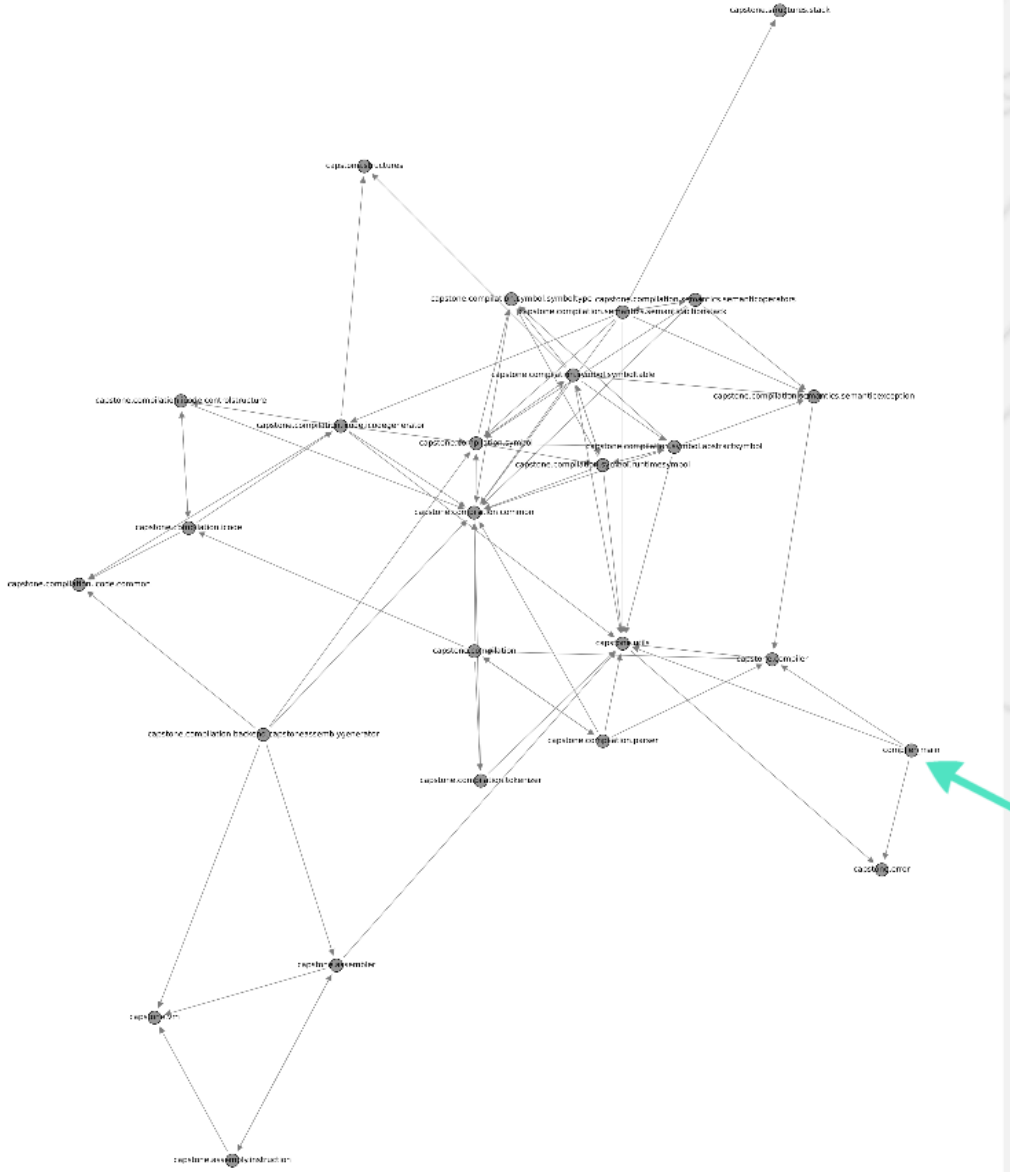
Prezi

D tools: rdmd

- Lots of awesome tools for D I've seen!
- rdmd is an excellent example
- No need for big compiler commands
- Why is rdmd cool?



Prezi



Generated using Gephi (http://gephi.github.io/)

Entry point

Command: rdmd -of"compiler.exe" compiler_main.d [args]

Unit tests in a large project

```d
enum Register
{
    INVALID_REGISTER = -1,
    //Data registers
    R0,
    R1,
    R2,
    R3,
    R4,
    R5,
    R6,
    R7,
    PC,//Program Counter
    SL,//Stack Limit
    SP,//Stack Pointer
    FP,//Frame Pointer
    SB//Stack Base
}

unittest
{
    writeln("Hello, world!");
    import std.conv : to;
    assert("R0".to!Register == Register.R0);
    assert(Register.FP.to!string == "FP");
}
```

- I want to test just *this* module.
- But package root is the parent directory!

```
rdmd -I.. -unittest -main vm.d
```

- I have no excuse but to test!

Another hack with contract programming

- At one point during development, I became convinced that a bug was in one of my harder-to-debug modules because of how it was called
- Because of logging, it was easy to track where I was, but I didn't want to insert a ton of code...

```d
class ICodeGenerator : SimpleLogger
{

    // TONS of functions here
    // ...but which one is the problem? D:

    invariant
    {
        import std.stdio : writeln;
        writeln("Bloop!");
    }
}
```

Prezi

Another hack with contract programming

- At one point during development, I became convinced that a bug was in one of my harder-to-debug modules because of how it was called
- Because of logging, it was easy to track where I was, but I didn't want to insert a ton of code...

```
class ICodeGenerator : SimpleLogger
{

    // TONS of functions here
    // ...but which one is the problem? D:

    invariant
    {
        import std.stdio : writeln;
        writeln("Bloop!");
    }
}
```

Prezi

```
class ICodeGenerator : SimpleLogger
{

    // TONS of functions here
    // ...but which one is the problem? D:

    invariant
    {
        import std.stdio : writeln;
        writeln("Bloop!");
    }
}
```

Prezi

Last Opinions

- We, as the D community, shouldn't be afraid to break stuff so it can make progress.
  - Make transition easy as possible
  - But give priority to language development
- Making D easy to get into will be essential for D's adoption into mainstream.

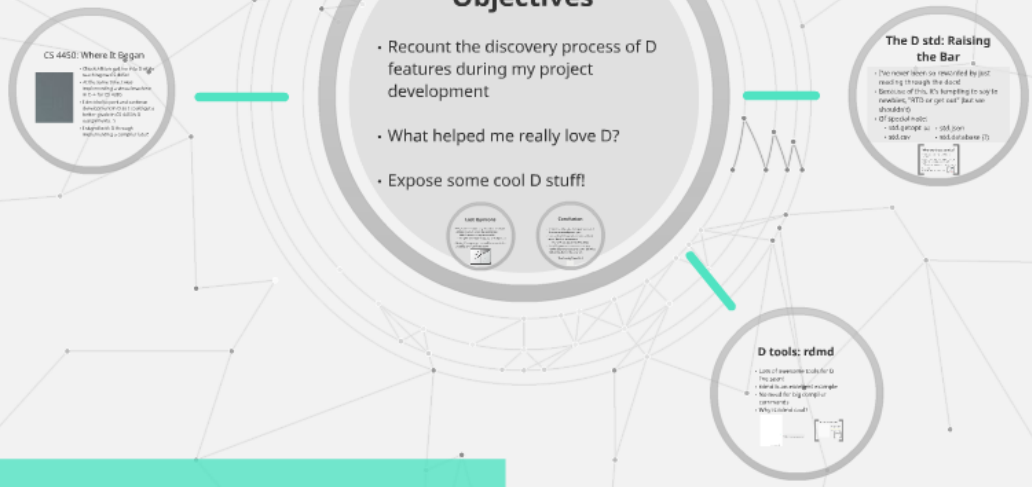
Prezi

Conclusion

- D removes obstacles from your path, and that makes development *fast.*
- D's standard library sets a new standard, much like its predecessors.
  - The fully-equipped machine shop!
- D's willingness to pioneer (which may involve breaking occasionally) will be what makes it better in the long run.

That's why I love D. :)

Prezi



Why I Love D: An Undergrad Experience

Erich Gubler

Prezi

Credits

- **Quote from Walter about the Belfast:**
  - http://forum.dlang.org/thread/fbdeybmbxpbgyxflmvny@forum.dlang.org?page=3#post-ki59uk:24fjc:241:40digitalmars.com
- **Images of the HMS Belfast:**
  - http://www.practicalmachinist.com/vb/machinery-photos/machine-shop-hms-belfast-173211/
- **Countdown image**
  - http://www.iab.net/media/image/countdown1.jpg
- **D Logo:**
  - https://github.com/D-Programming-Language/dlang.org/blob/master/images/dlogo.svg

Prezi