Transcompilation into D

DConf 2019
London

Bastiaan
Veelo

Transcompilation into D

DConf 2019
London

Bastiaan
Veelo

Transcompilation into D

DConf 2019
London

Bastiaan
Veelo

Intermediate Representation / Computer Aided Porting

Transcompilation into D

DConf 2019
London

Bastiaan
Veelo

Intermediate Representation / Computer Aided Porting

Transcompilation into D

DConf 2019
London

Bastiaan
Veelo

Intermediate Representation / Computer Aided Porting

Transcompilation into D in D

DConf 2019
London

Bastiaan
Veelo

~~Intermediate Representation~~ / Computer Aided Porting

Extended Pascal → D

ca. 500 kloc

In parallel with development

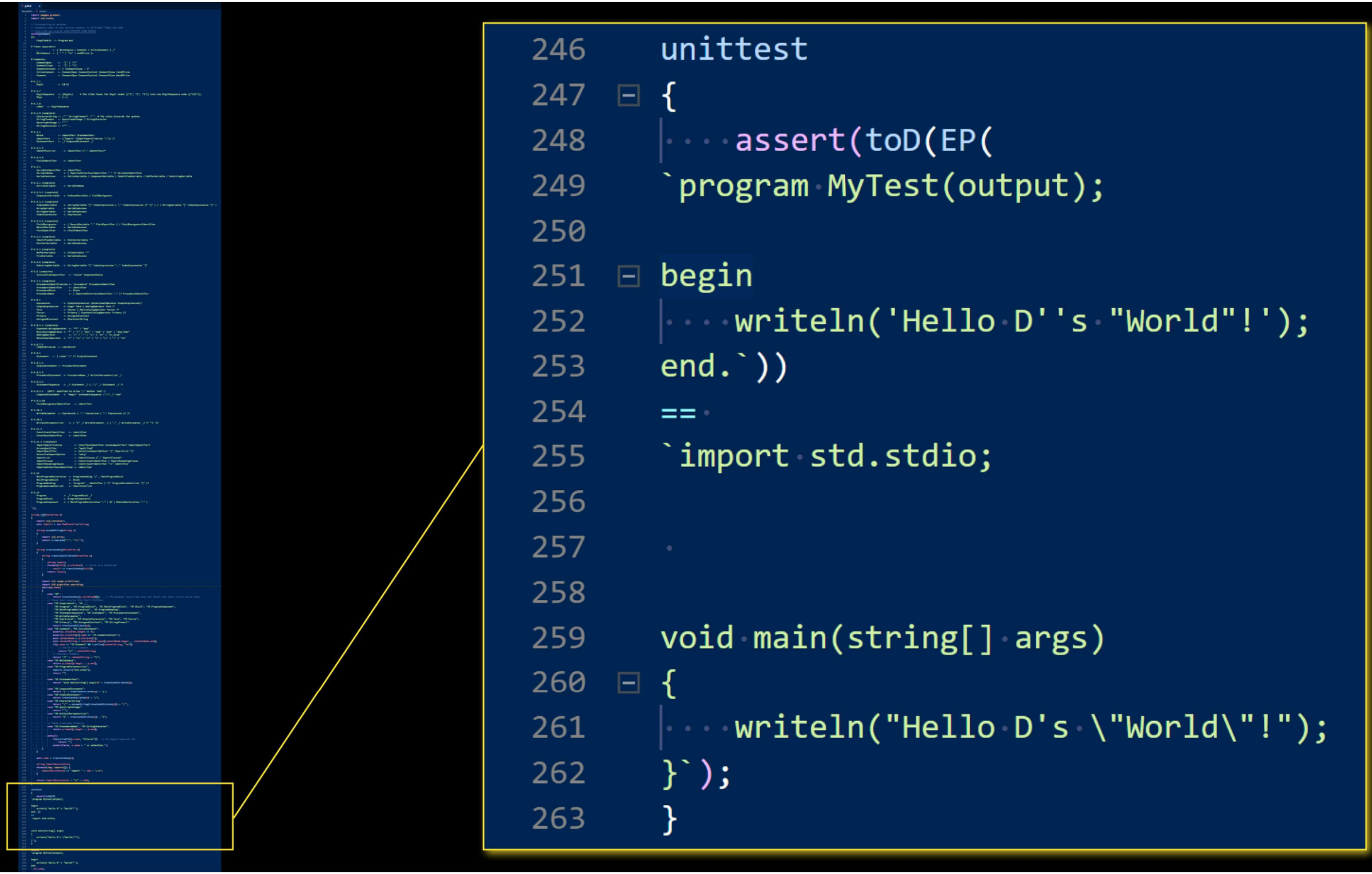
Conversion of the DMD frontend from C++ to D

DConf 2015

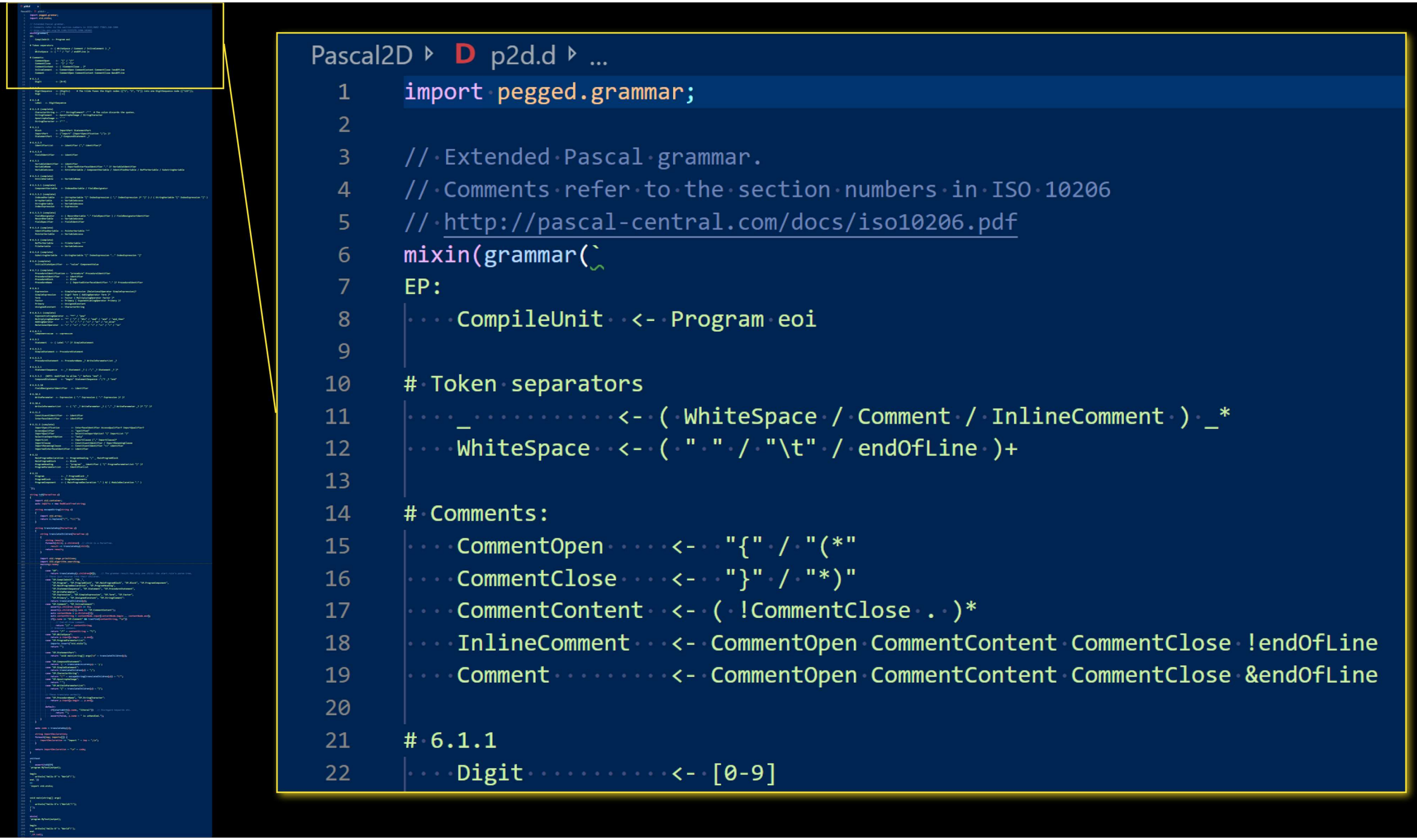
22 May 2015
commit #8

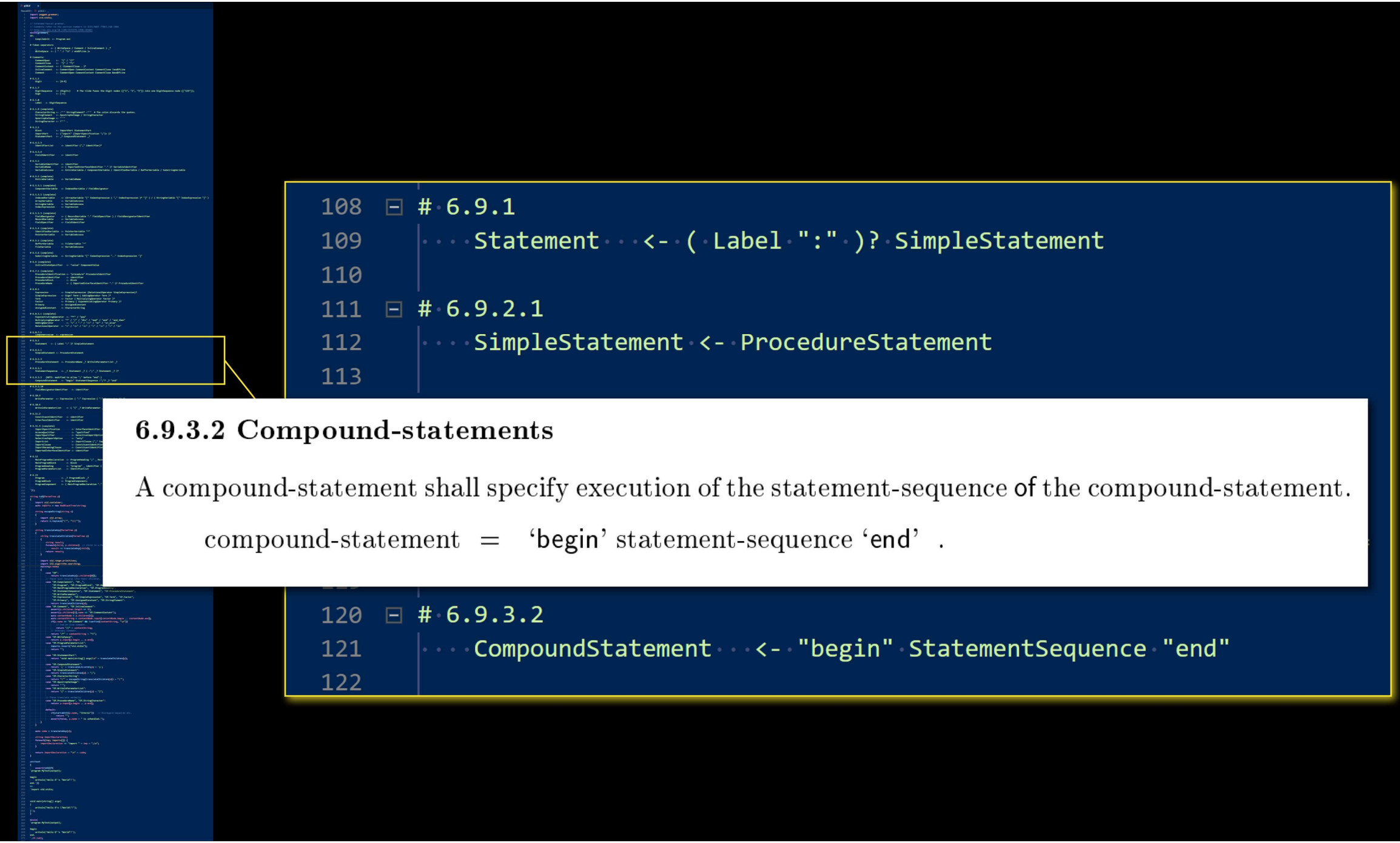
Transcompile "Hello, World!"

263 loc



```
246     unittest
247 ⊟   {
248         assert(toD(EP(
249     `program MyTest(output);
250
251 ⊟   begin
252         writeln('Hello D''s "World"!');
253     end.`))
254     ==
255     `import std.stdio;
256
257
258
259     void main(string[] args)
260 ⊟   {
261         writeln("Hello D's \"World\"!");
262     }`);
263     }
```


Pascal2D ▸ p2d.d ▸ ...
1 import pegged.grammar;
2
3 // Extended Pascal grammar.
4 // Comments refer to the section numbers in ISO 10206
5 // http://pascal-central.com/docs/iso10206.pdf
6 mixin(grammar(`
7 EP:
8 CompileUnit <- Program eoi
9
10 # Token separators
11 _ <- (WhiteSpace / Comment / InlineComment)_*
12 WhiteSpace <- (" " / "\t" / endOfLine)+
13
14 # Comments:
15 CommentOpen <- "{" / "(*"
16 CommentClose <- "}" / "*)"
17 CommentContent <- (!CommentClose .)*
18 InlineComment <- CommentOpen CommentContent CommentClose !endOfLine
19 Comment <- CommentOpen CommentContent CommentClose &endOfLine
20
21 # 6.1.1
22 Digit <- [0-9]

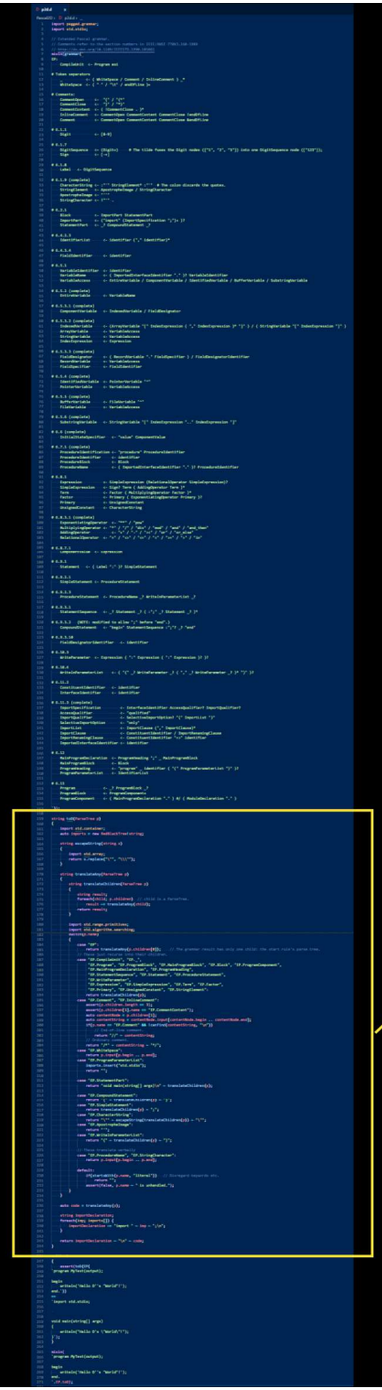


```
108  # 6.9.1
109      Statement   <- ( Label ":" )? SimpleStatement
110
111  # 6.9.2.1
112      SimpleStatement <- ProcedureStatement
113
120  # 6.9.3.2
121      CompoundStatement   <- "begin" StatementSequence "end"
122
```

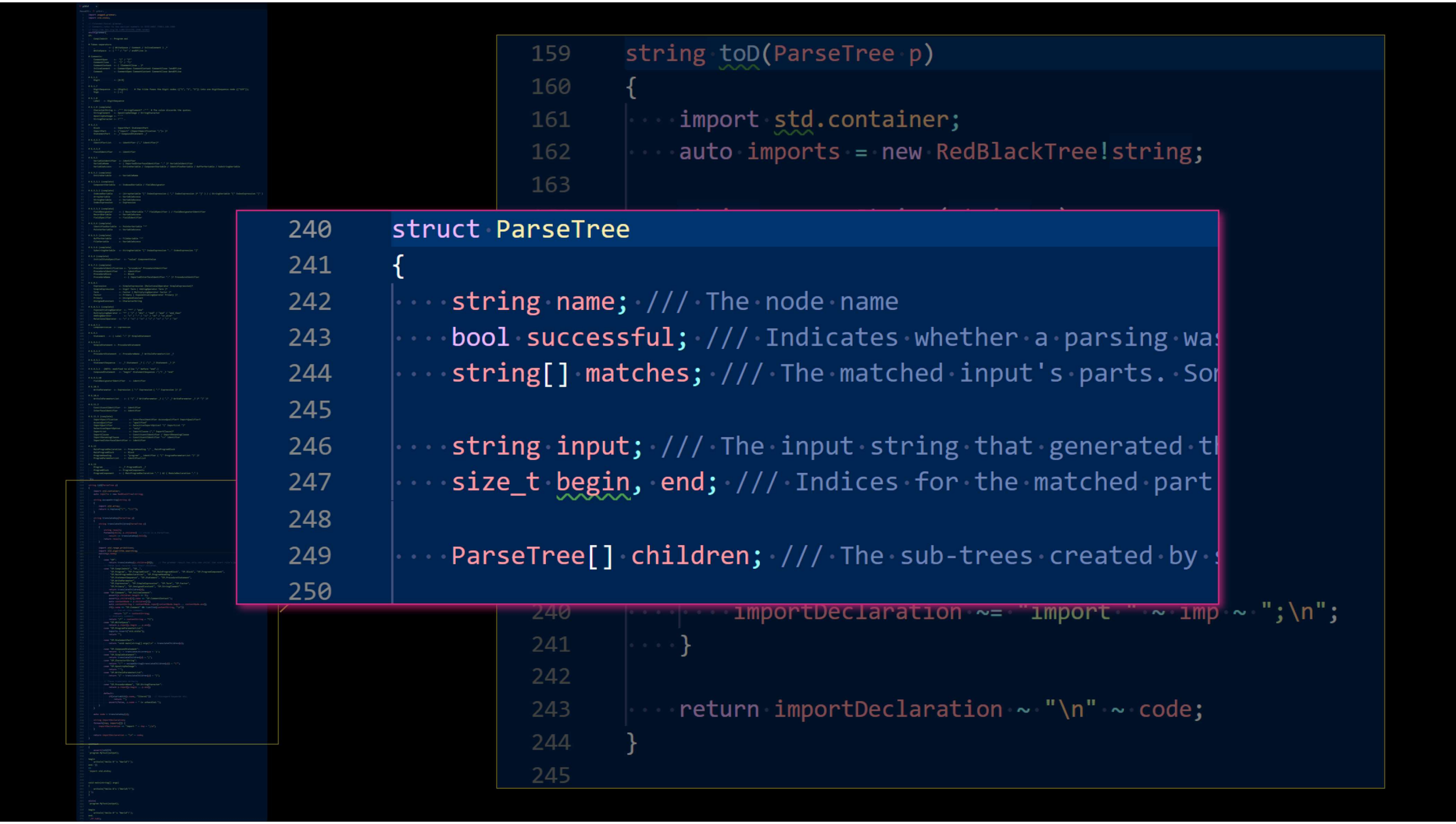
6.9.3.2 Compound-statements

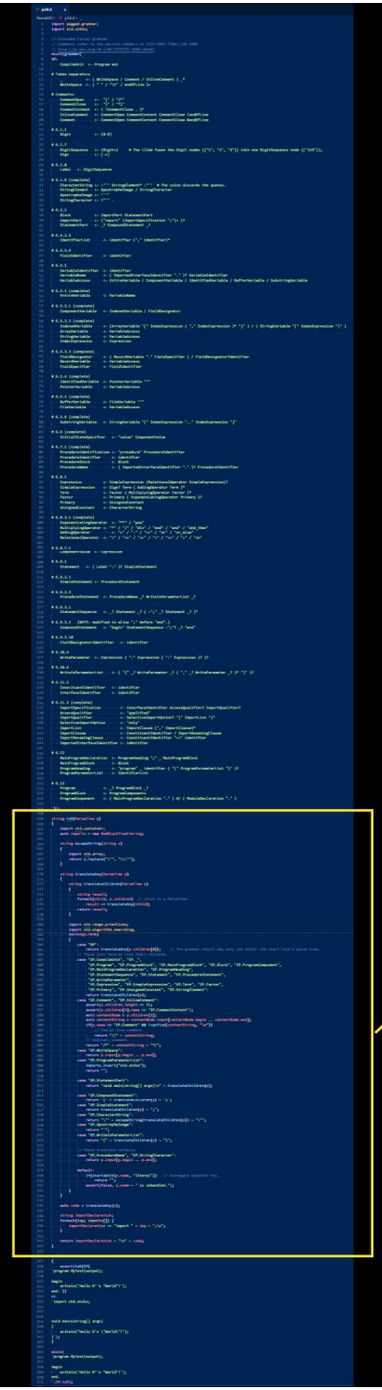
A compound-statement shall specify execution of the statement-sequence of the compound-statement.

compound-statement = ‘begin’ statement-sequence ‘end’ .




```d
159    string toD(ParseTree p)
160    {
161        import std.container;
162        auto imports = new RedBlackTree!string;
163
164        string escapeString(string s)
165        {…
168        }
169
170        string translateAny(ParseTree p)
171        {…
234        }
235
236        auto code = translateAny(p);
237
238        string importDeclaration;
239        foreach(imp; imports[]) {
240            importDeclaration ~= "import " ~ imp ~ ";\n";
241        }
242
243        return importDeclaration ~ "\n" ~ code;
244    }
245
```

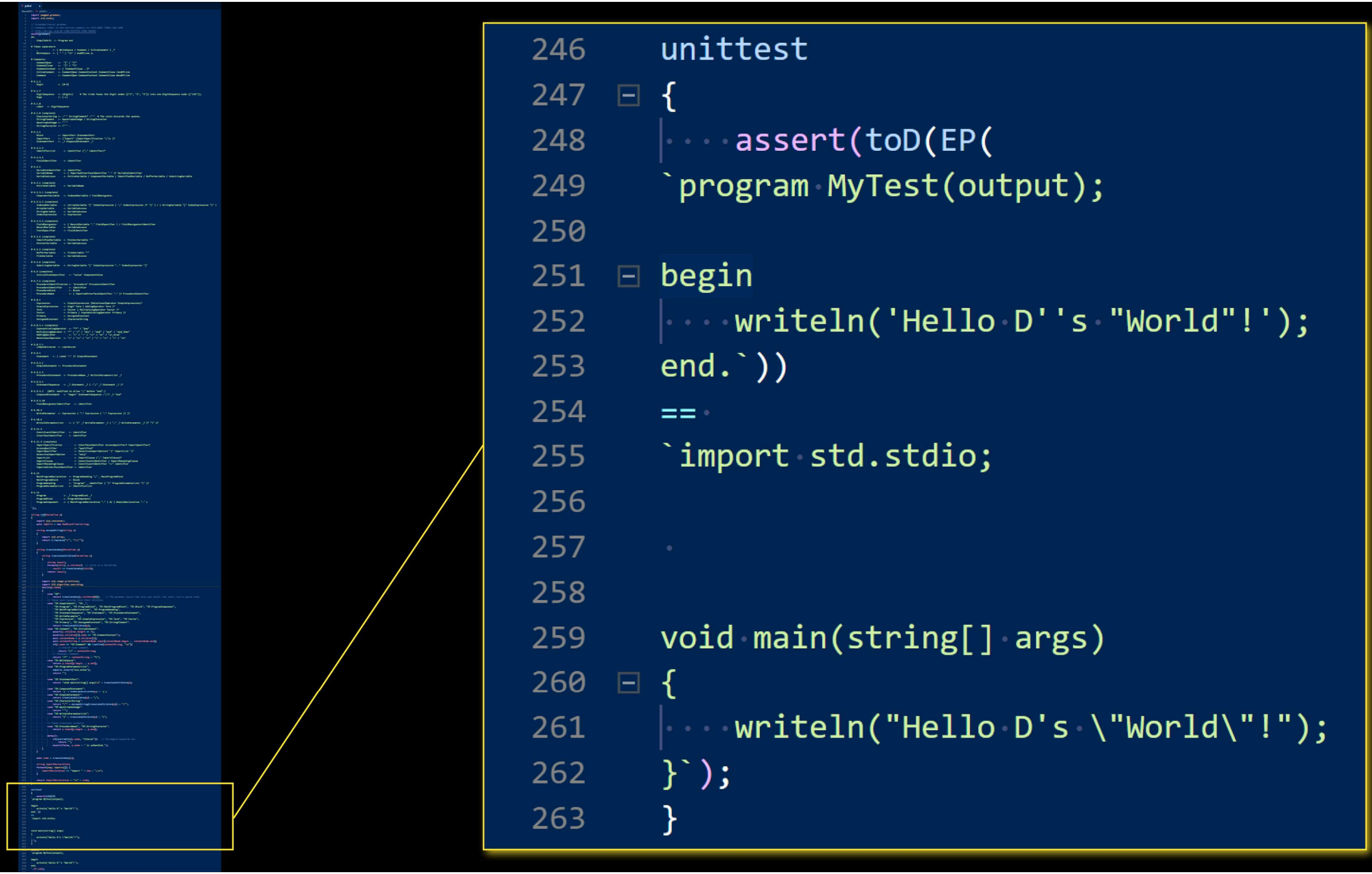

159 string toD(ParseTree p)
160 {
161 import std.container;
162 auto imports = new RedBlackTree!string;
163
240 struct ParseTree
241 {
242 string name; /// The node name
243 bool successful; /// Indicates whether a parsing was
244 string[] matches; /// The matched input's parts. Son
245
246 string input; /// The input string that generated th
247 size_t begin, end; /// Indices for the matched part
248
249 ParseTree[] children; /// The sub-trees created by s
250
240 importDeclaration ~= "import " ~ imp ~ ";\n";
241 }
242
243 return importDeclaration ~ "\n" ~ code;
244 }
245



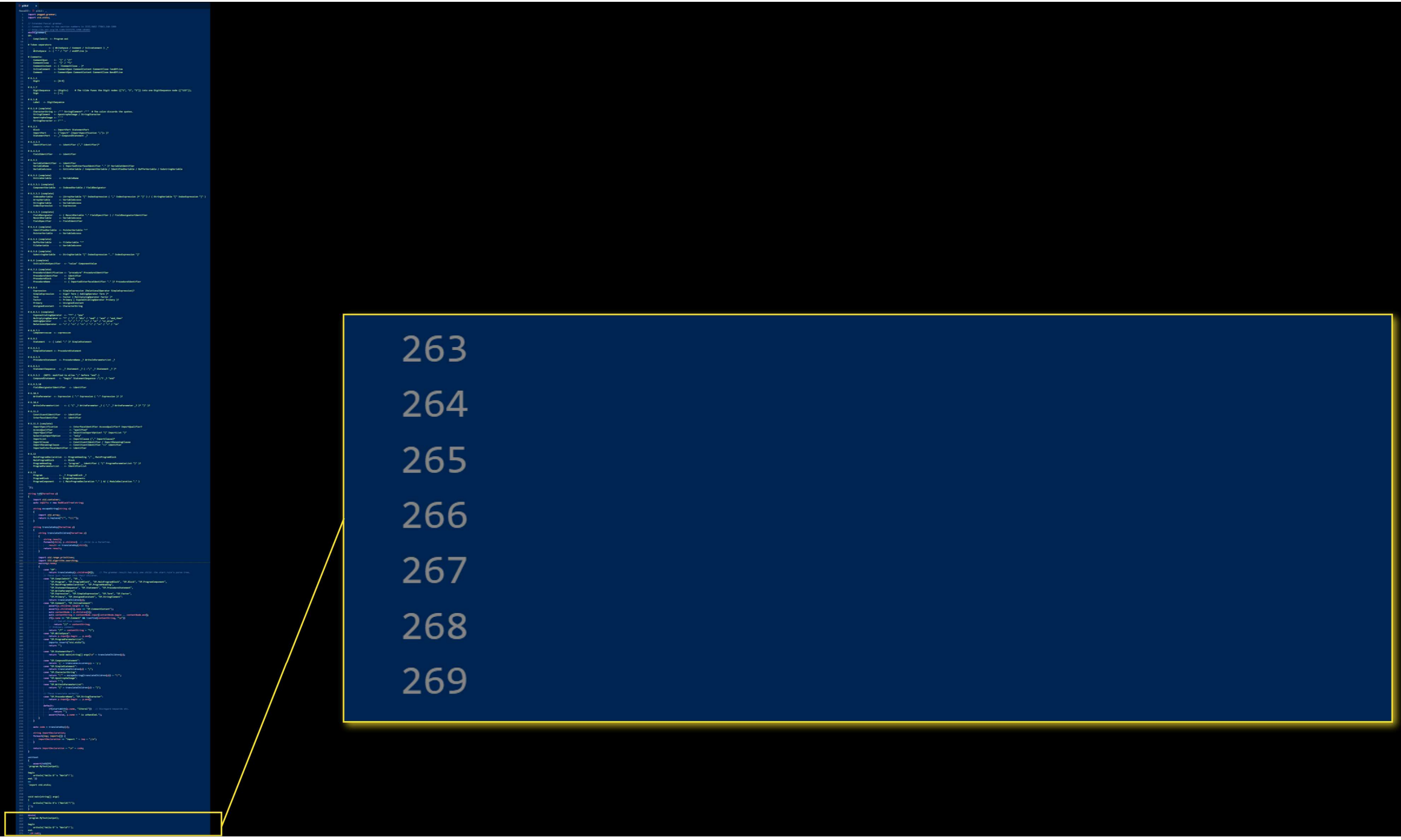

```d
159    string toD(ParseTree p)
160    {
161        import std.container;
162        auto imports = new RedBlackTree!string;
163
164        string escapeString(string s)
165        {…
168        }
169
170        string translateAny(ParseTree p)
171        {…
234        }
235
236        auto code = translateAny(p);
237
238        string importDeclaration;
239        foreach(imp; imports[]) {
240            importDeclaration ~= "import " ~ imp ~ ";\n";
241        }
242
243        return importDeclaration ~ "\n" ~ code;
244    }
245
```

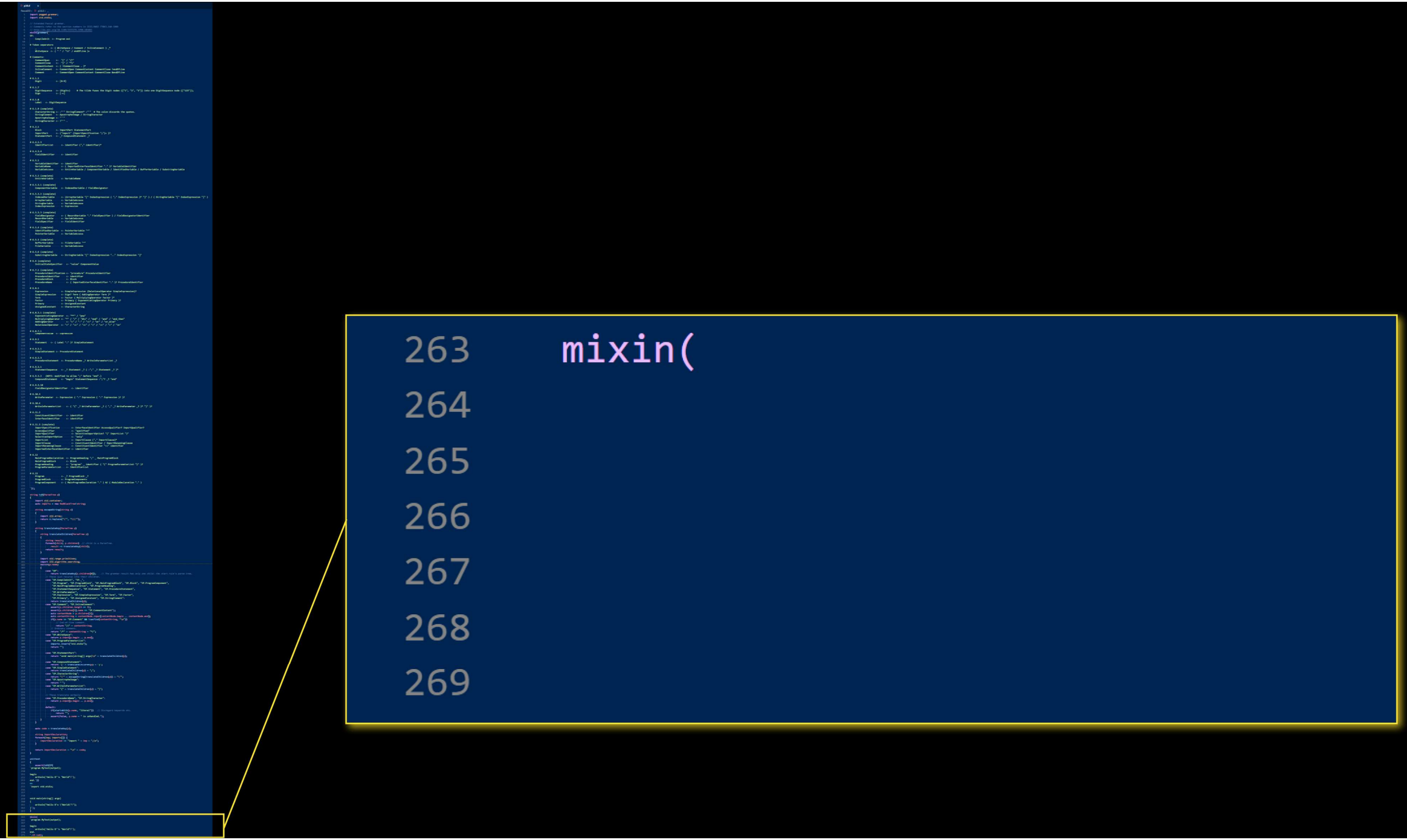
```
169     string translateAny(ParseTree p)
170     {
171         string translateChildren(ParseTree p)
172         {
173             string result;
174             foreach(child; p.children)  // child is a ParseTree.
175                 result ~= translateAny(child);
176             return result;
177         }
178
179         import std.algorithm.searching;
180         switch(p.name)
181         {
182             case "EP":
183                 return translateAny(p.children[0]);   // The grammar result has only
184             // These just recurse into their children.
185             case "EP.CompileUnit", "EP._",
186                  "EP.Program", "EP.ProgramBlock", "EP.MainProgramBlock", "EP.Block",
187                  "EP.ProgramComponent", "EP.MainProgramDeclaration",
188                  "EP.ProgramHeading", "EP.StatementSequence", "EP.Statement",
189                  "EP.ProcedureStatement", "EP.WriteParameter",
190                  "EP.Expression", "EP.SimpleExpression", "EP.Term", "EP.Factor",
191                  "EP.Primary", "EP.UnsignedConstant", "EP.StringElement":
192                 return translateChildren(p);
```

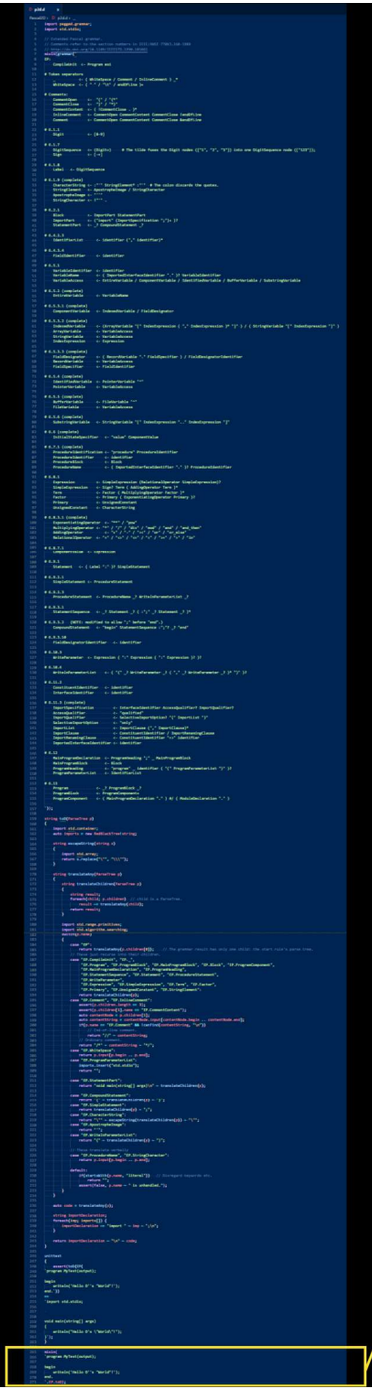
```
212             case "EP.CompoundStatement":
213                 return "{" ~ translateChildren(p) ~ "}";
214             case "EP.SimpleStatement":
215                 return translateChildren(p) ~ ";";
216             case "EP.CharacterString":
217                 return "\"" ~ escapeString(translateChildren(p)) ~ "\"";
218             case "EP.ApostropheImage":
219                 return "'";
220             case "EP.WritelnParameterList":
221                 return "(" ~ translateChildren(p) ~ ")";
222
223             // These translate verbally
224             case "EP.ProcedureName", "EP.StringCharacter":
225                 return p.input[p.begin .. p.end];
226
227             default:
228                 if(startsWith(p.name, "literal"))   // Disregard keywords etc.
229                     return "";
230                 assert(false, p.name ~ " is unhandled.");
231         }
```



```
246     unittest
247 ⊟   {
248         assert(toD(EP(
249     `program MyTest(output);
250
251 ⊟   begin
252         writeln('Hello D''s "World"!');
253     end.`))
254     ==
255     `import std.stdio;
256
257
258
259     void main(string[] args)
260 ⊟   {
261         writeln("Hello D's \"World\"!");
262     }`);
263     }
```


263
264
265
266
267
268
269


263 mixin(
264
265
266
267
268
269

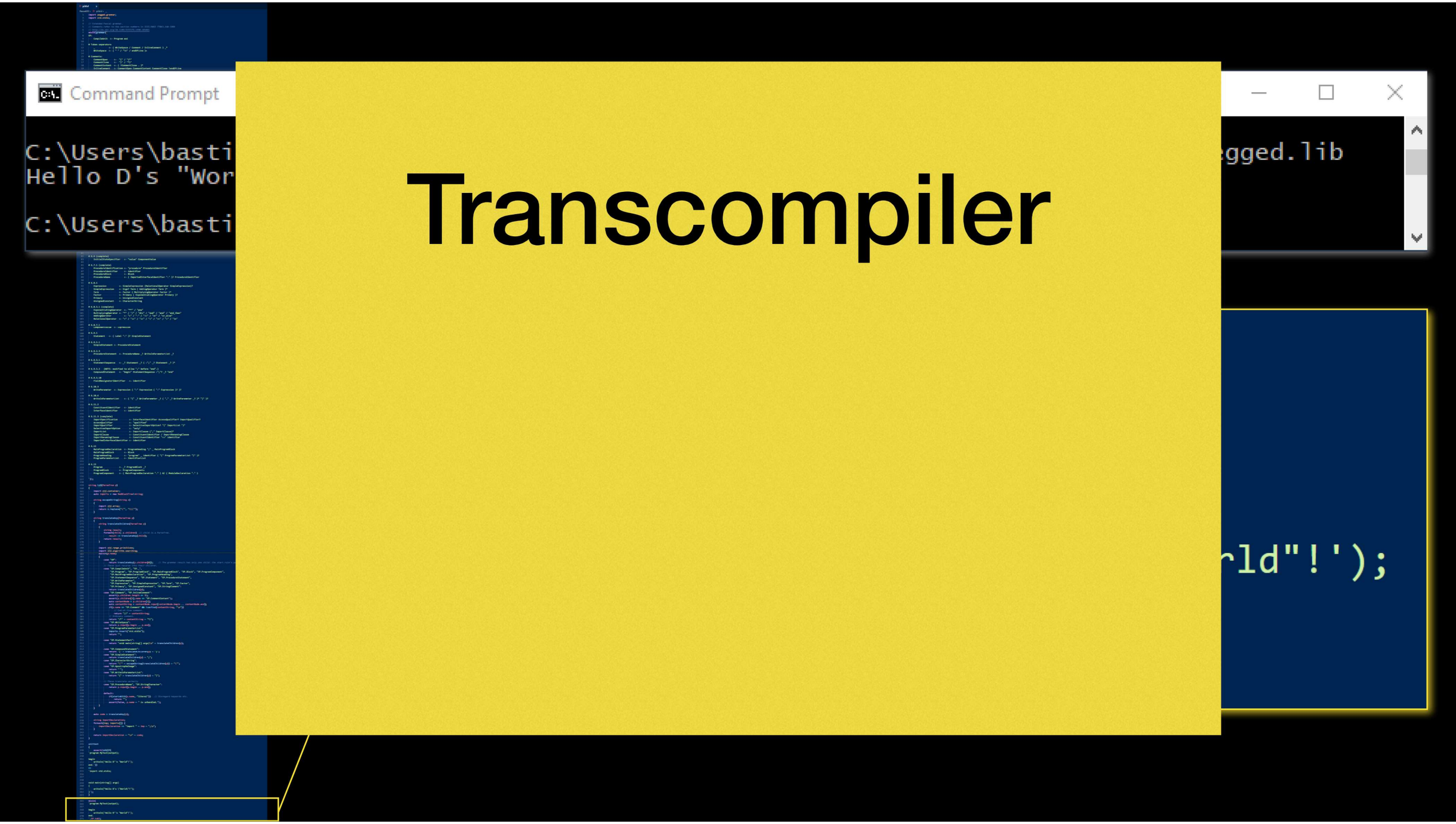


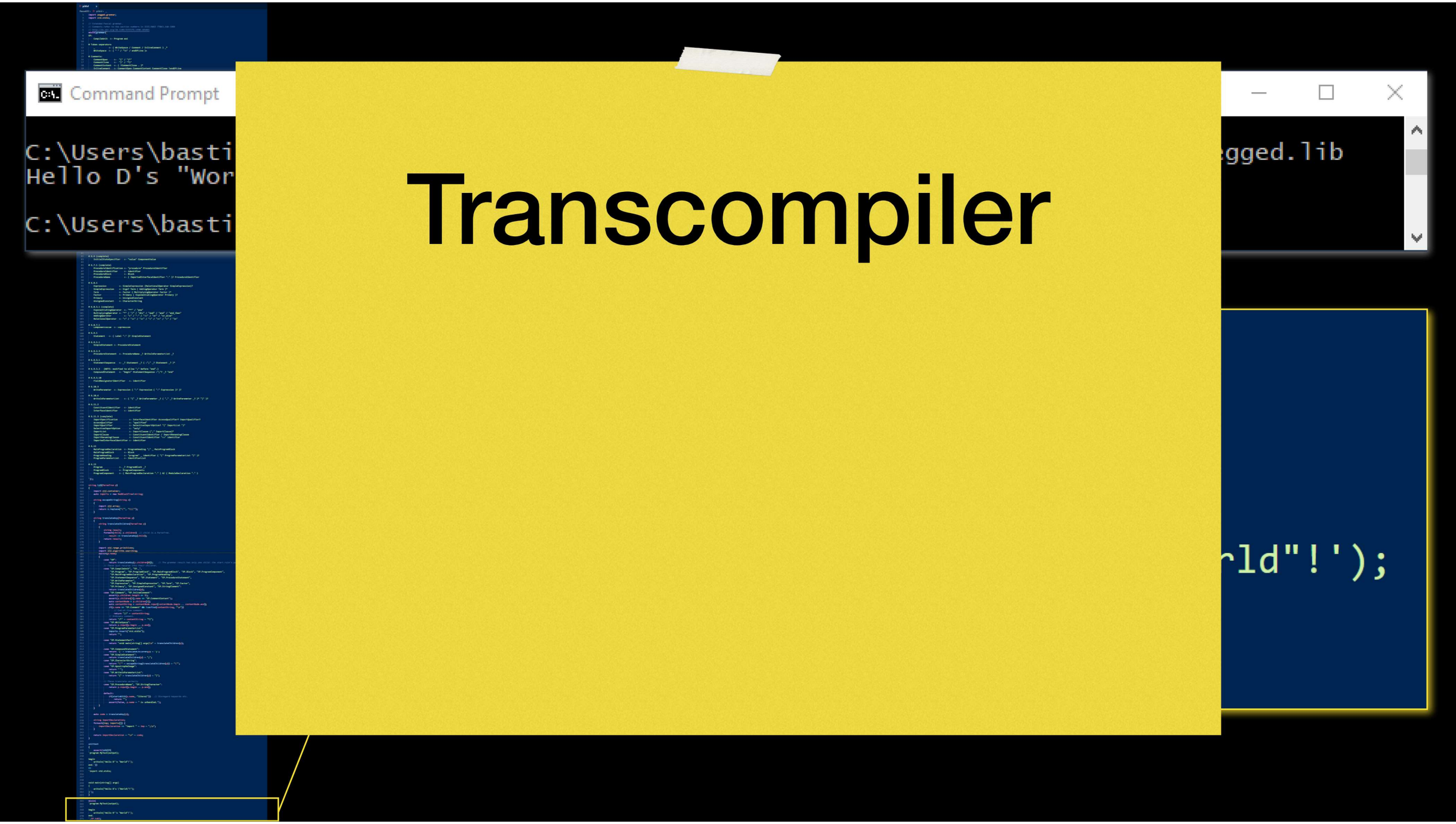
```
263    mixin(`
264    program MyTest(output);
265
266    begin
267    ····writeln('Hello D''s "World"!');
268    end.
269    `
```

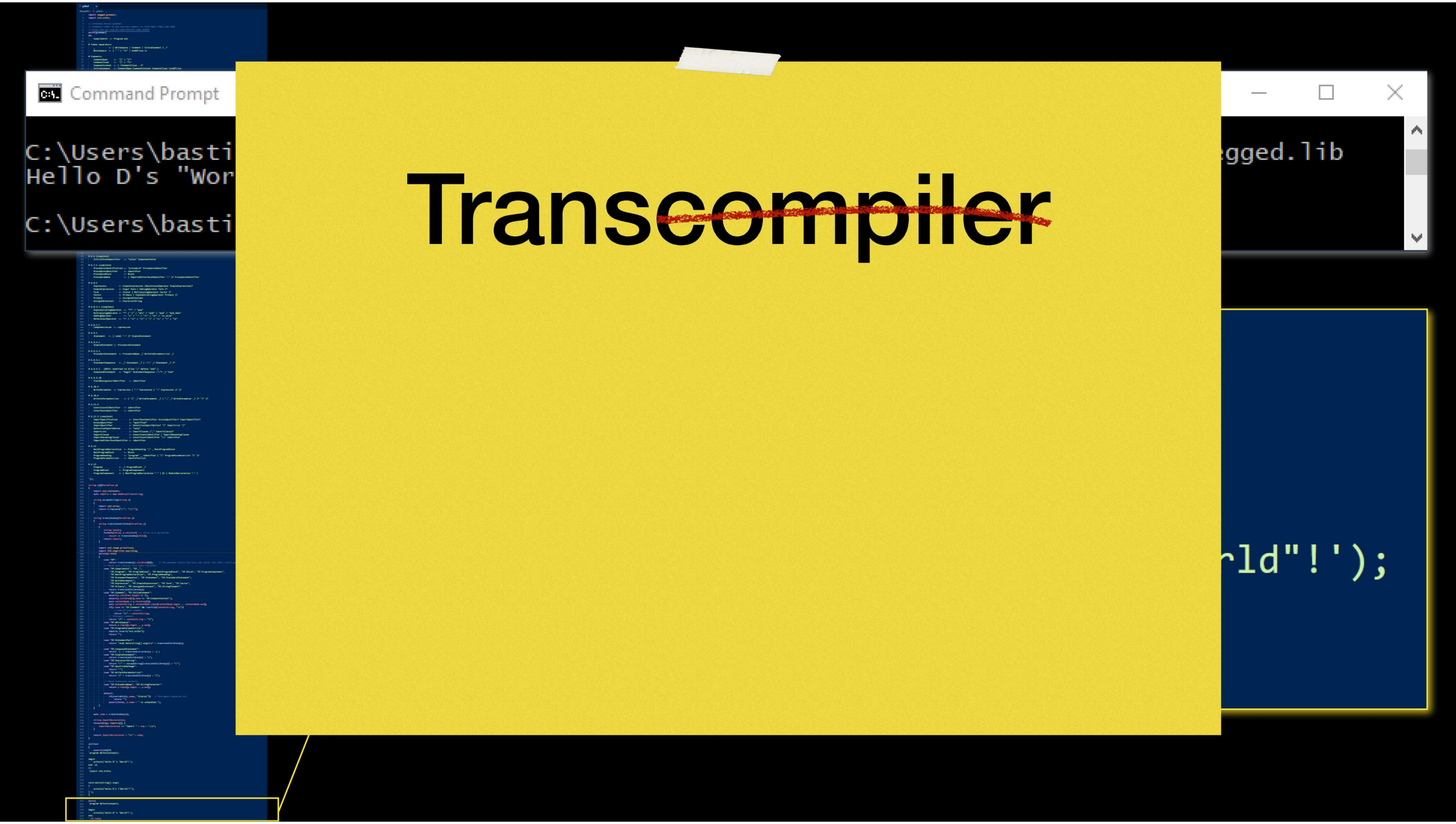
Command Prompt

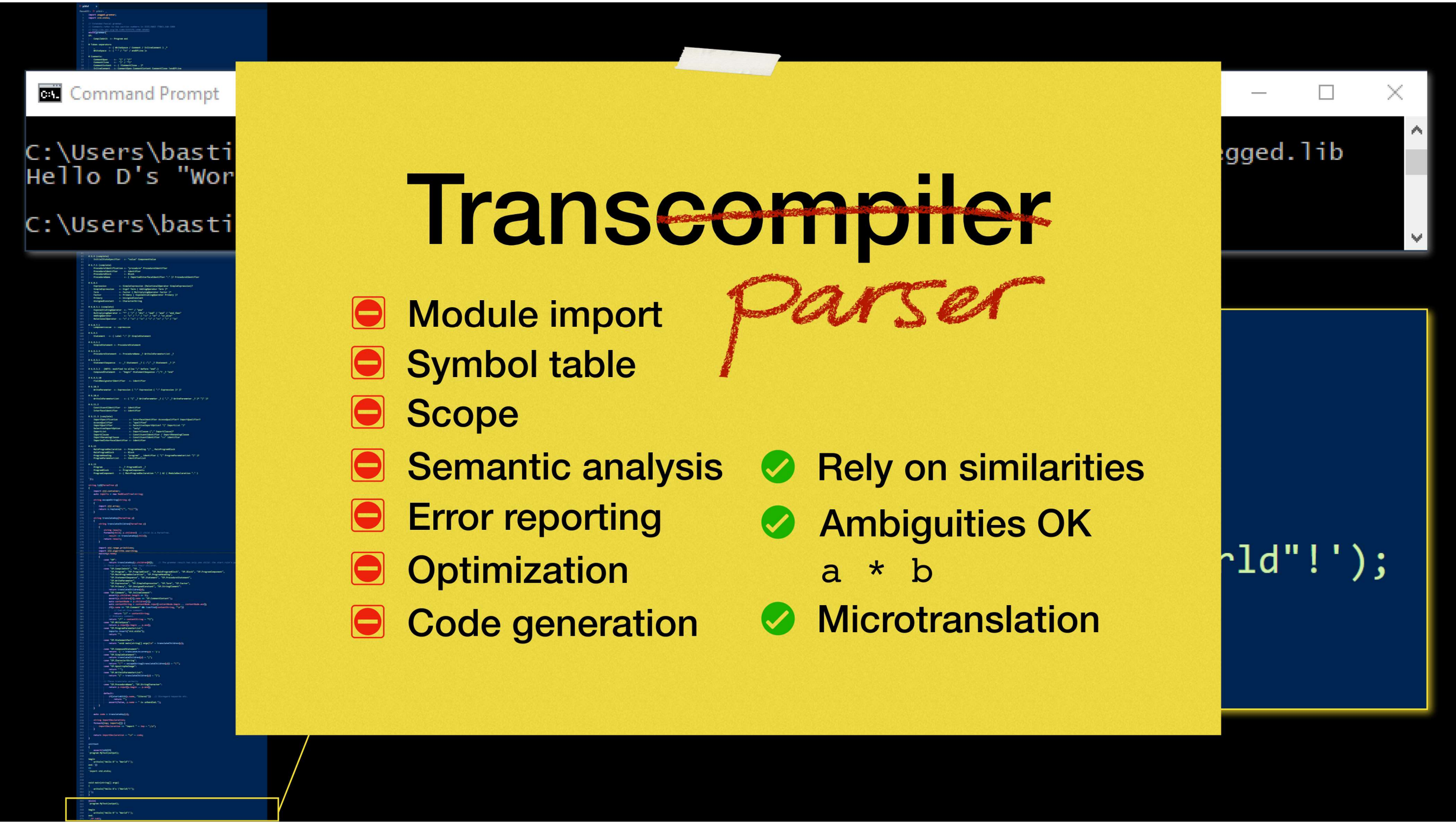
```
C:\Users\bastiaan\Documents\D\Pascal2D>rdmd -m64 -I=..\Pegged p2d.d ..\Pegged\pegged.lib
Hello D's "World"!

C:\Users\bastiaan\Documents\D\Pascal2D>
```

```
263    mixin(`
264    program MyTest(output);
265
266    begin
267        writeln('Hello D''s "World"!');
268    end.
269    `.EP.toD);
```


Command Prompt
C:\Users\basti
Hello D's "Wor
C:\Users\basti
gged.lib
rld"!');
Transcompiler


Command Prompt
C:\Users\basti
Hello D's "Wor
C:\Users\basti
gged.lib
rld"!');
Transcompiler


Command Prompt
C:\Users\basti
Hello D's "Wor
C:\Users\basti
gged.lib
rld"!');
Transcompiler



Transcompiler

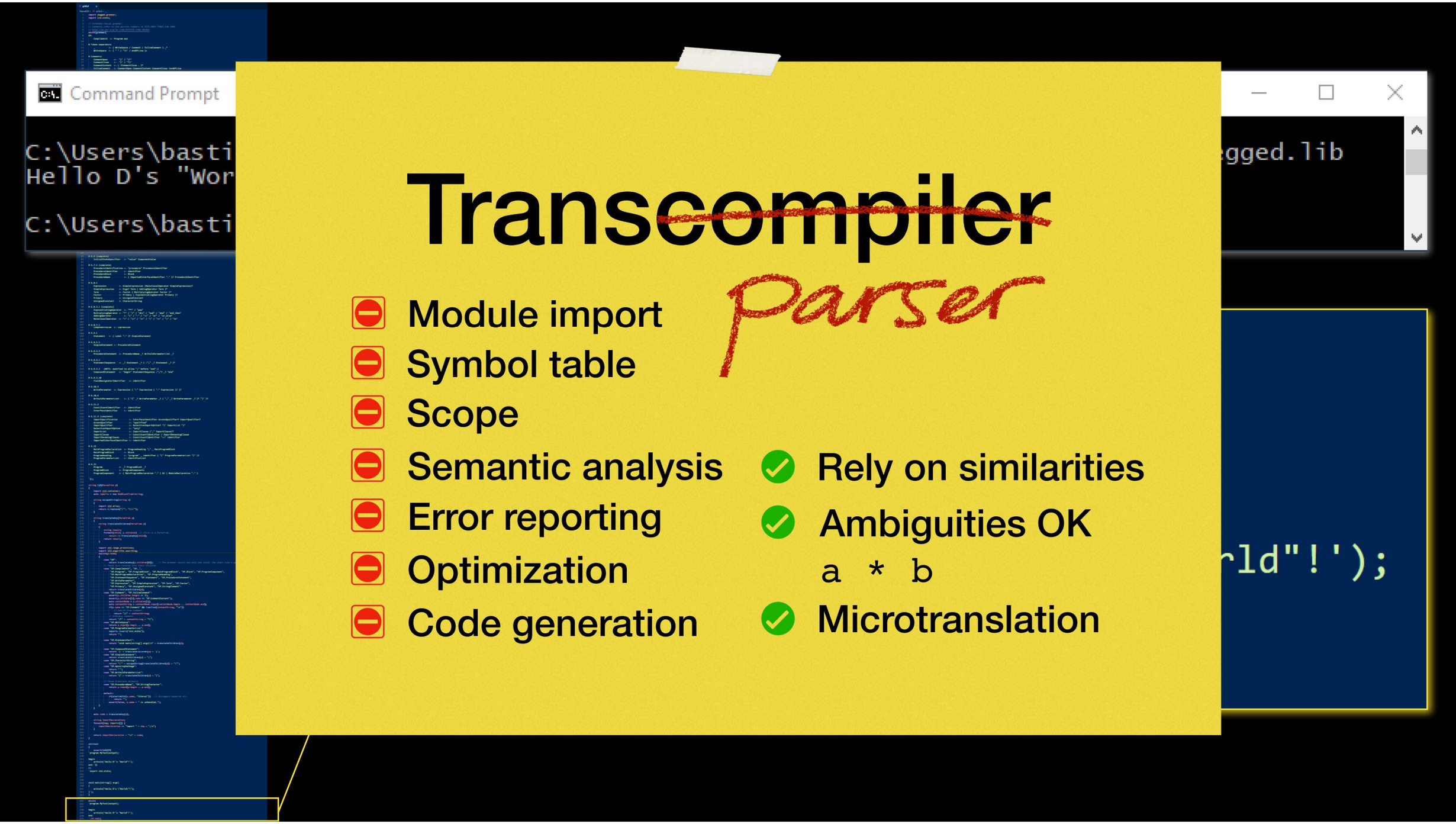
parser

- ⛔ Module import
- ⛔ Symbol table
- ⛔ Scope
- ⛔ Semantic analysis
- ⛔ Error reporting
- ⛔ Optimization
- ⛔ Code generation

- ✅ Rely on similarities
- ✅ Ambiguities OK

```
a * b
```

- ✅ Microtranslation



Transcompiler

parser

- ⛔ Module import
- ⛔ Symbol table
- ⛔ Scope
- ⛔ Semantic analysis
- ⛔ Error reporting
- ⛔ Optimization
- ⛔ Code generation

- ✅ Rely on similarities
- ✅ Ambiguities OK

```
a * b
```

- ✅ Microtranslation

Command Prompt

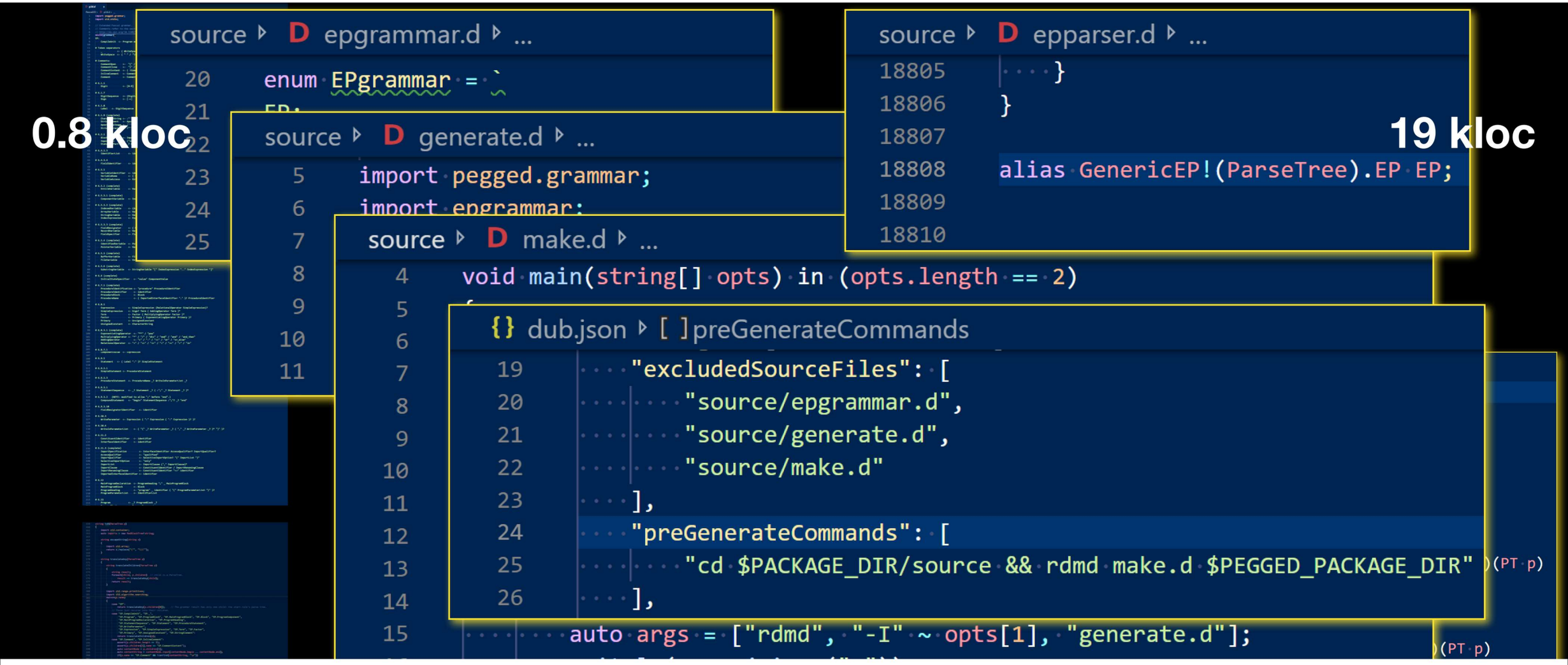
```
C:\Users\bastiaan\Documents\D\Pascal2D>rdmd -m64 -I=..\Pegged p2d.d ..\Pegged\pegged.lib
Hello D's "World"!

C:\Users\bastiaan\Documents\D\Pascal2D>
```

```
263    mixin(`
264    program MyTest(output);
265
266    begin
267        writeln('Hello D''s "World"!');
268    end.
269    `.EP.toD);
```

```
p2d.d ×
Pascal2D › p2d.d › ...
import pegged.grammar;
import std.stdio;

// Extended Pascal grammar.
// Comments refer to the section numbers in IEEE/ANSI 770X3.160-1989
// http://dx.doi.org/10.1109/IEEESTD.1990.101061
mixin(grammar(`
EP:
    CompileUnit  <- Program eoi

# Token separators
    _            <- ( WhiteSpace / Comment / InlineComment ) _*
    WhiteSpace   <- ( " " / "\t" / endOfLine )+

# Comments:
    CommentOpen     <-  "{" / "(*"
    CommentClose    <-  "}" / "*)"
    CommentContent  <- ( !CommentClose . )*
    InlineComment   <- CommentOpen CommentContent CommentClose !endOfLine
    Comment         <- CommentOpen CommentContent CommentClose &endOfLine

# 6.1.1
    Digit           <- [0-9]

# 6.1.7
    DigitSequence   <- ~(Digit+)     # The tilde fuses the Digit nodes (["1", "2", "3"]) into one DigitSequence node (["123"]);
    Sign            <- [-+]

# 6.1.8
    Label   <- DigitSequence

# 6.1.9 (complete)
    CharacterString <- :"'" StringElement* :"'"  # The colon discards the quotes.
    StringElement   <- ApostropheImage / StringCharacter
    ApostropheImage <- "''"
    StringCharacter <- !"'" .

# 6.2.1
    Block           <- ImportPart StatementPart
    ImportPart      <- ("import" (ImportSpecification ";")+ )?
    StatementPart   <- _? CompoundStatement _?

# 6.4.2.3
    IdentifierList      <- identifier ("," identifier)*

# 6.4.3.4
    FieldIdentifier     <- identifier

# 6.5.1
    VariableIdentifier  <- identifier
    VariableName        <- ( ImportedInterfaceIdentifier "." )? VariableIdentifier
    VariableAccess      <- EntireVariable / ComponentVariable / IdentifiedVariable / BufferVariable / SubstringVariable

# 6.5.2 (complete)
    EntireVariable      <- VariableName

# 6.5.3.1 (complete)
    ComponentVariable   <- IndexedVariable / FieldDesignator

# 6.5.3.2 (complete)
    IndexedVariable     <- (ArrayVariable "[" IndexExpression ( "," IndexExpression )* "]" ) / ( StringVariable "[" IndexExpression "]" )
    ArrayVariable       <- VariableAccess
    StringVariable      <- VariableAccess
    IndexExpression     <- Expression

# 6.5.3.3 (complete)
    FieldDesignator     <- ( RecordVariable "." FieldSpecifier ) / FieldDesignatorIdentifier
    RecordVariable      <- VariableAccess
    FieldSpecifier      <- FieldIdentifier

# 6.5.4 (complete)
    IdentifiedVariable  <- PointerVariable "^"
    PointerVariable     <- VariableAccess

# 6.5.5 (complete)
    BufferVariable      <- FileVariable "^"
    FileVariable        <- VariableAccess

# 6.5.6 (complete)
    SubstringVariable   <- StringVariable "[" IndexExpression ".." IndexExpression "]"

# 6.6 (complete)
    InitialStateSpecifier   <- "value" ComponentValue

# 6.7.1 (complete)
    ProcedureIdentification <- "procedure" ProcedureIdentifier
    ProcedureIdentifier     <- identifier
    ProcedureBlock          <- Block
    ProcedureName           <- ( ImportedInterfaceIdentifier "." )? ProcedureIdentifier

# 6.8.1
    Expression          <- SimpleExpression (RelationalOperator SimpleExpression)?
    SimpleExpression    <- Sign? Term ( AddingOperator Term )*
    Term                <- Factor ( MultiplyingOperator Factor )*
    Factor              <- Primary ( ExponentiatingOperator Primary )?
    Primary             <- UnsignedConstant
    UnsignedConstant    <- CharacterString

# 6.8.3.1 (complete)
    ExponentiatingOperator  <- "**" / "pow"
    MultiplyingOperator <- "*" / "/" / "div" / "mod" / "and" / "and_then"
    AddingOperator          <- "+" / "-" / "><" / "or" / "or_else"
    RelationalOperator  <- "=" / "<>" / "<=" / "<" / ">=" / ">" / "in"

# 6.8.7.1
    ComponentValue  <- Expression

# 6.9.1
    Statement   <- ( Label ":" )? SimpleStatement

# 6.9.2.1
    SimpleStatement <- ProcedureStatement

# 6.9.2.3
    ProcedureStatement  <- ProcedureName _? WritelnParameterList _?

# 6.9.3.1
    StatementSequence   <- _? Statement _? ( :";" _? Statement _? )*

# 6.9.3.2   (NOTE: modified to allow ";" before "end".)
    CompoundStatement   <- "begin" StatementSequence :";"? _? "end"

# 6.9.3.10
    FieldDesignatorIdentifier   <- identifier

# 6.10.3
    WriteParameter  <- Expression ( ":" Expression ( ":" Expression )? )?

# 6.10.4
    WritelnParameterList    <- ( "(" _? WriteParameter _? ( "," _? WriteParameter _? )* ")" )?

# 6.11.2
    ConstituentIdentifier   <- identifier
    InterfaceIdentifier     <- identifier

# 6.11.3 (complete)
    ImportSpecification         <- InterfaceIdentifier AccessQualifier? ImportQualifier?
    AccessQualifier             <- "qualified"
    ImportQualifier             <- SelectiveImportOption? "(" ImportList ")"
    SelectiveImportOption       <- "only"
    ImportList                  <- ImportClause ("," ImportClause)*
    ImportClause                <- ConstituentIdentifier / ImportRenamingClause
    ImportRenamingClause        <- ConstituentIdentifier "=>" identifier
    ImportedInterfaceIdentifier <- identifier

# 6.12
    MainProgramDeclaration  <- ProgramHeading ";" _ MainProgramBlock
    MainProgramBlock        <- Block
    ProgramHeading          <- "program" _ identifier ( "(" ProgramParameterList ")" )?
    ProgramParameterList    <- IdentifierList

# 6.13
    Program             <- _? ProgramBlock _?
```

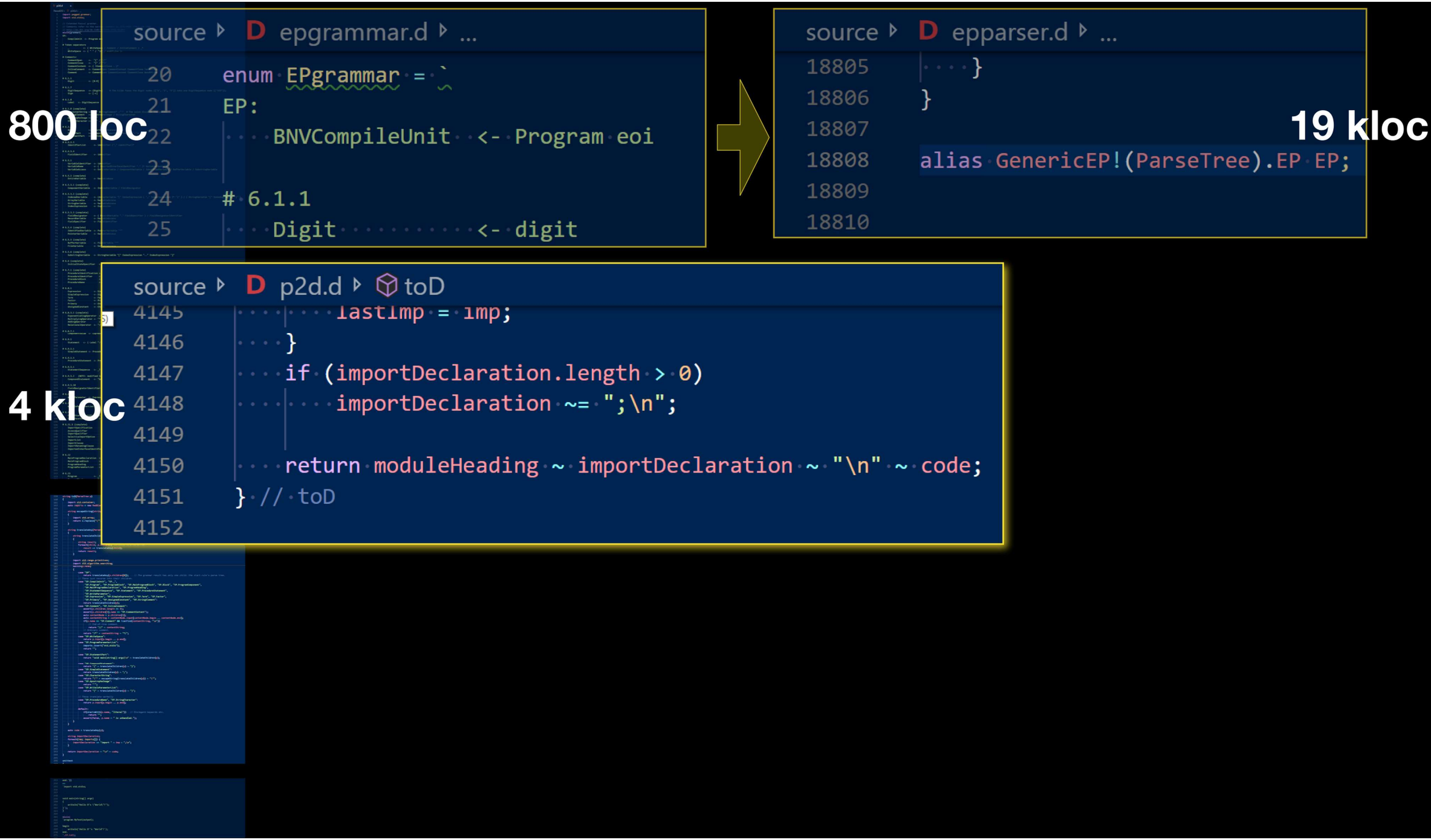
```
string toD(ParseTree p)
{
    import std.container;
    auto imports = new RedBlackTree!string;

    string escapeString(string s)
    {
        import std.array;
        return s.replace("\"", "\\\"");
    }

    string translateAny(ParseTree p)
    {
        string translateChildren(ParseTree p)
        {
            string result;
            foreach(child; p.children)  // child is a ParseTree.
                result ~= translateAny(child);
            return result;
        }

        import std.range.primitives;
        import std.algorithm.searching;
        switch(p.name)
        {
            case "EP":
                return translateAny(p.children[0]);    // The grammar result has only one child: the start rule's parse tree.
            // These just recurse into their children.
            case "EP.CompileUnit", "EP._",
                 "EP.Program", "EP.ProgramBlock", "EP.MainProgramBlock", "EP.Block", "EP.ProgramComponent",
                 "EP.MainProgramDeclaration", "EP.ProgramHeading",
                 "EP.StatementSequence", "EP.Statement", "EP.ProcedureStatement",
                 "EP.WriteParameter",
                 "EP.Expression", "EP.SimpleExpression", "EP.Term", "EP.Factor",
                 "EP.Primary", "EP.UnsignedConstant", "EP.StringElement":
                return translateChildren(p);
            case "EP.Comment", "EP.InlineComment":
                assert(p.children.length == 3);
                assert(p.children[1].name == "EP.CommentContent");
                auto contentNode = p.children[1];
                auto contentString = contentNode.input[contentNode.begin .. contentNode.end];
                if(p.name == "EP.Comment" && !canFind(contentString, "\n"))
                    // End-of-line comment.
                    return "//" ~ contentString;
                // Ordinary comment.
                return "/*" ~ contentString ~ "*/";
            case "EP.WhiteSpace":
                return p.input[p.begin .. p.end];
            case "EP.ProgramParameterList":
                imports.insert("std.stdio");
                return "";

            case "EP.StatementPart":
                return "void main(string[] args)\n" ~ translateChildren(p);

            case "EP.CompoundStatement":
                return "{" ~ translateChildren(p) ~ "}";
            case "EP.SimpleStatement":
                return translateChildren(p) ~ ";";
            case "EP.CharacterString":
                return "\"" ~ escapeString(translateChildren(p)) ~ "\"";
            case "EP.ApostropheImage":
                return "'";
            case "EP.WritelnParameterList":
                return "(" ~ translateChildren(p) ~ ")";

            // These translate verbally
            case "EP.ProcedureName", "EP.StringCharacter":
                return p.input[p.begin .. p.end];

            default:
                if(startsWith(p.name, "literal"))   // Disregard keywords etc.
                    return "";
                assert(false, p.name ~ " is unhandled.");
        }
    }

    auto code = translateAny(p);

    string importDeclaration;
    foreach(imp; imports[]) {
        importDeclaration ~= "import " ~ imp ~ ";\n";
    }

    return importDeclaration ~ "\n" ~ code;
}

unittest
```

```
end.`))
==
`import std.stdio;



void main(string[] args)
{
    writeln("Hello D's \"World\"!");
}`);
}

mixin(
`program MyTest(output);

begin
    writeln('Hello D''s "World"!');
end.
`.EP.toD);
```



examples/generated-source does not work. Regression in preGenerateCommands. #1474

Open veelo opened this issue on 17 May 2018 · 3 comments


800 loc
source ▸ D epgrammar.d ▸ ...
20 enum EPgrammar = `
21 EP:
22 BNVCompileUnit <- Program eoi
23
24 # 6.1.1
25 Digit <- digit
19 kloc
source ▸ D epparser.d ▸ ...
18805 }
18806 }
18807
18808 alias GenericEP!(ParseTree).EP EP;
18809
18810
4 kloc
source ▸ D p2d.d ▸ toD
4145 lastImp = imp;
4146 }
4147 if (importDeclaration.length > 0)
4148 importDeclaration ~= ";\n";
4149
4150 return moduleHeading ~ importDeclaration ~ "\n" ~ code;
4151 } // toD
4152

800 loc

source ▸ D epgrammar.d ▸ ...

```
20  enum EPgrammar = `
21  EP:
22      BNVCompileUnit <- Program eoi
23
24  # 6.1.1
25      Digit          <- digit
```

19 kloc

source ▸ D epparser.d ▸ ...

```
18805       }
18806   }
18807
18808   alias GenericEP!(ParseTree).EP EP;
18809
18810
```

4 kloc

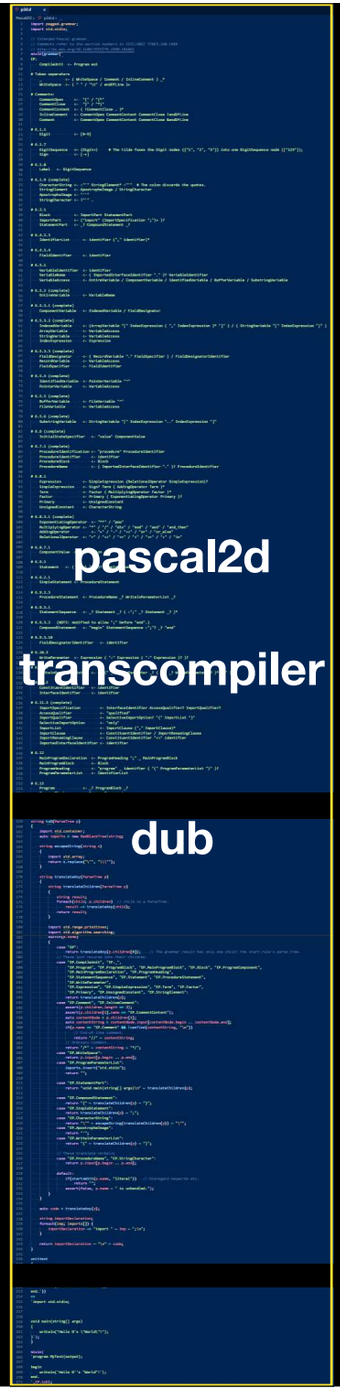
source ▸ D p2d.d ▸ toD

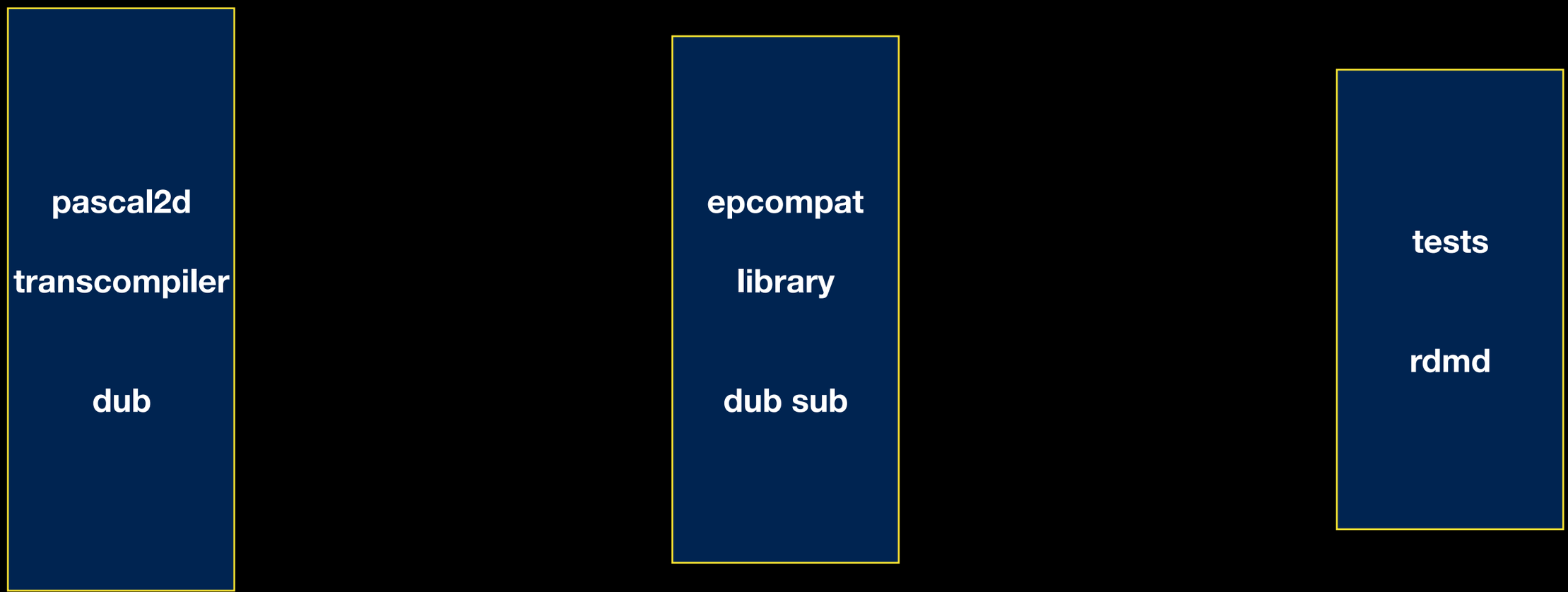
```
4145            lastImp = imp;
4146        }
4147        if (importDeclaration.length > 0)
4148            importDeclaration ~= ";\n";
4149
4150        return moduleHeading ~ importDeclaration ~ "\n" ~ code;
4151    } // toD
4152
```

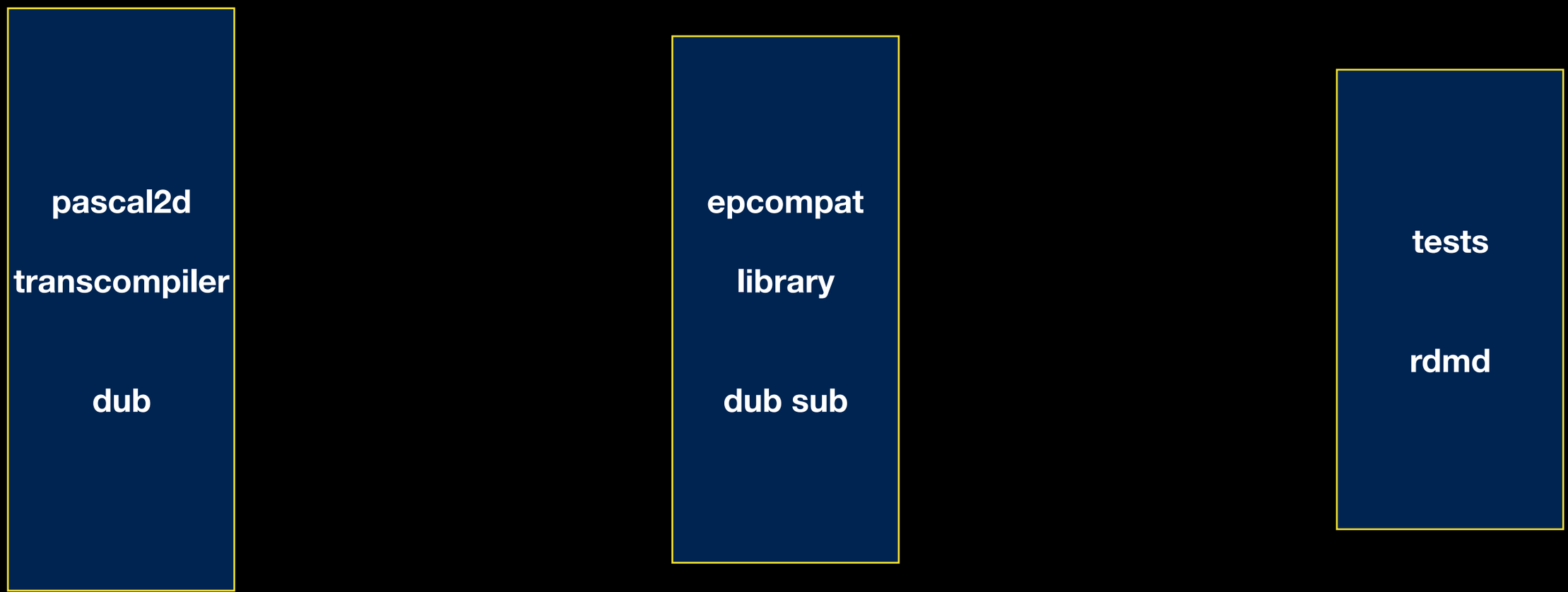
80 loc

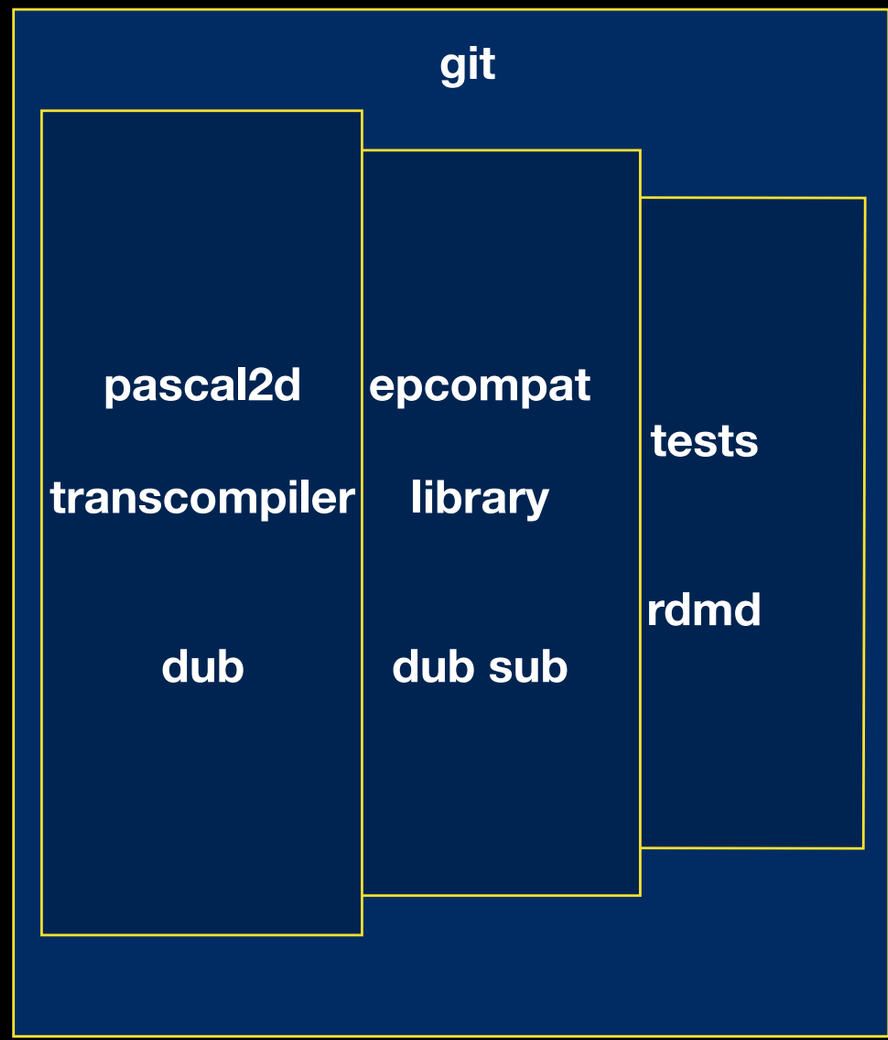
source ▸ D main.d ▸ ...

```
65          auto parsed = args.length == 3 ?
66              EP(readText(args[1])) ~ readText(args[2])) :
67              EP(readText(args[1]));
68      +   if (syntax_tree) ...
70      +   if (!parsed.successful) { ...
76          }
77          writeln(toD(parsed));
```


pascal2d
transcompiler
dub


pascal2d
transcompiler
dub
epcompat
library
dub sub
tests
rdmd


pascal2d
transcompiler
dub
epcompat
library
dub sub
tests
rdmd


git
pascal2d
transcompiler
dub
epcompat
library
dub sub
tests
rdmd

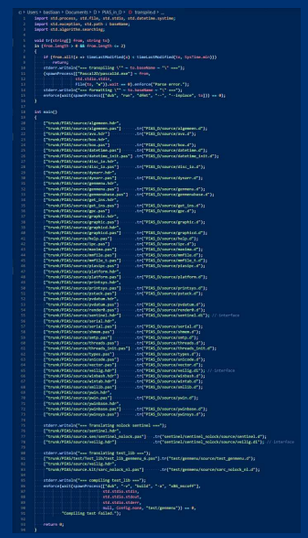
git

dub
app

dub
lib

svn
Pascal

rdmd
transpile.d



git

pascal2d
transcompiler

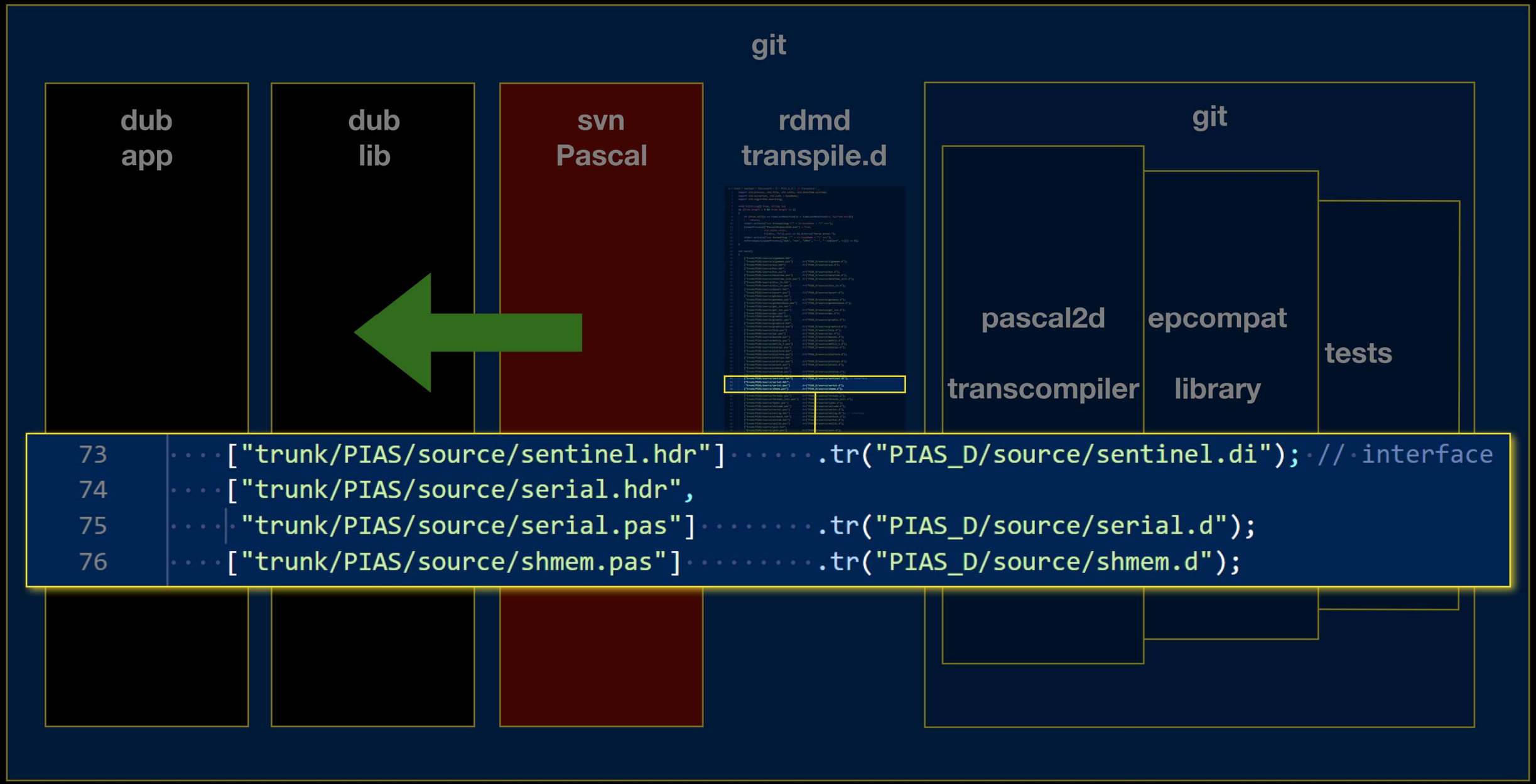
epcompat
library

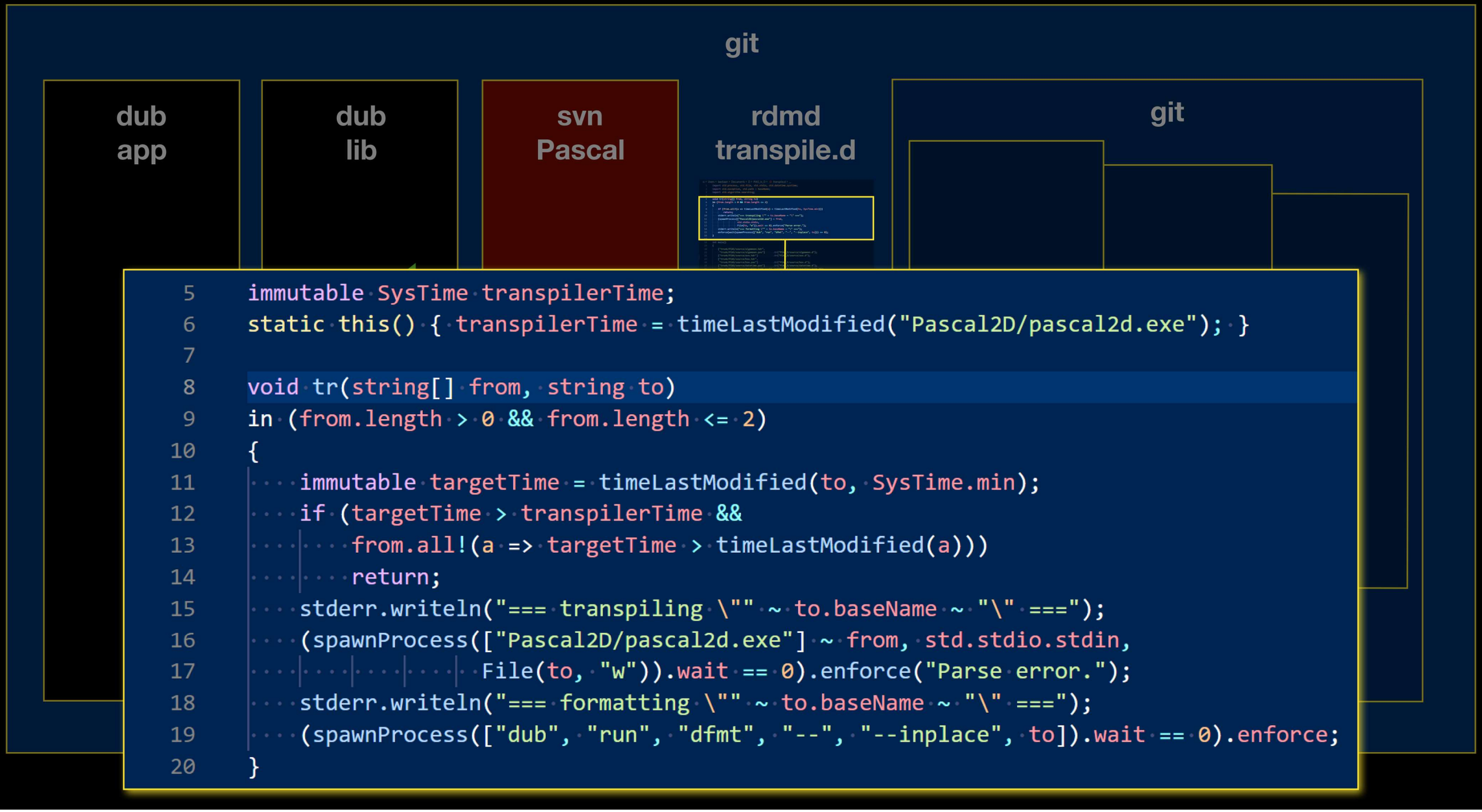
dub

dub sub

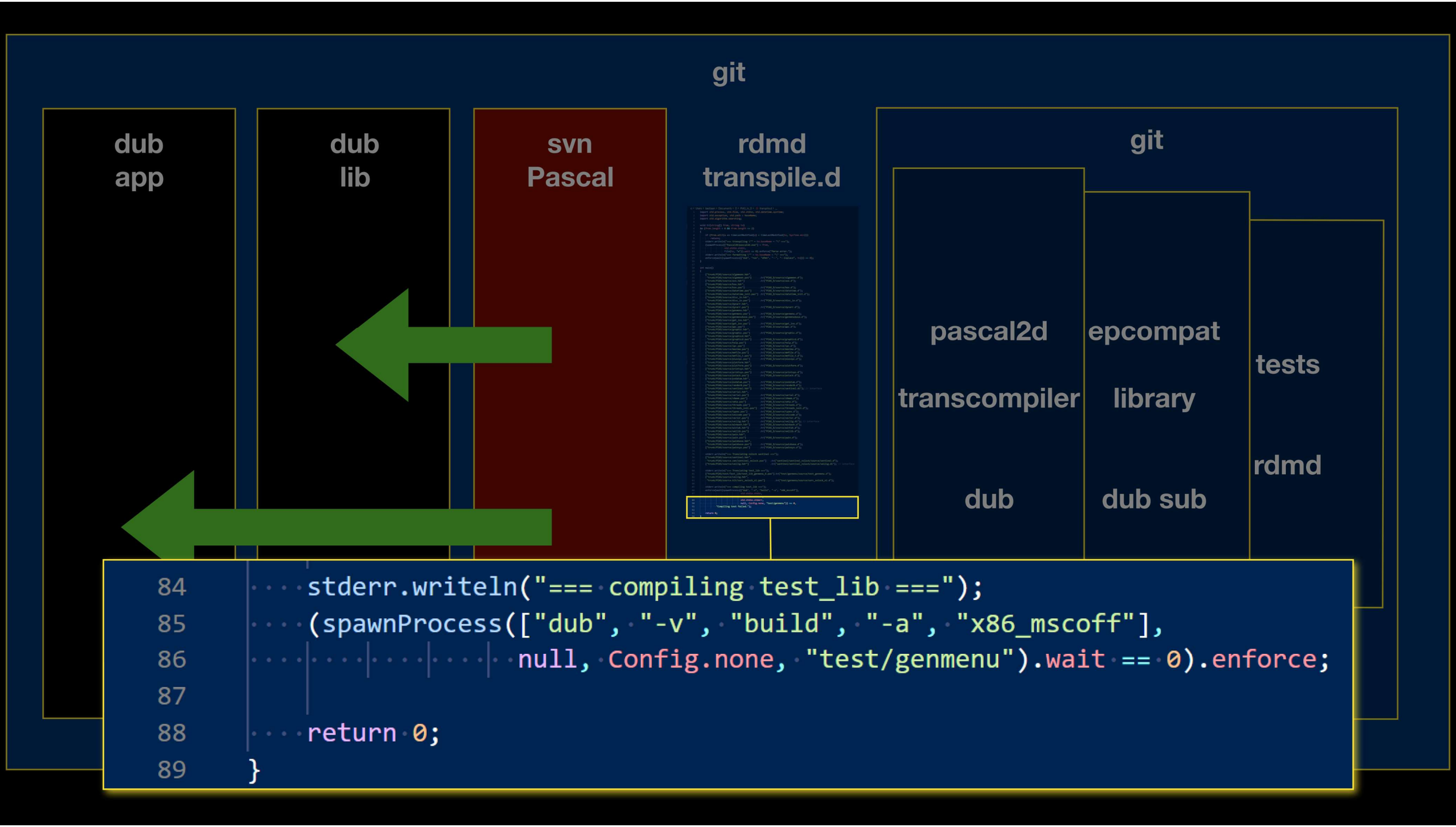
tests

rdmd

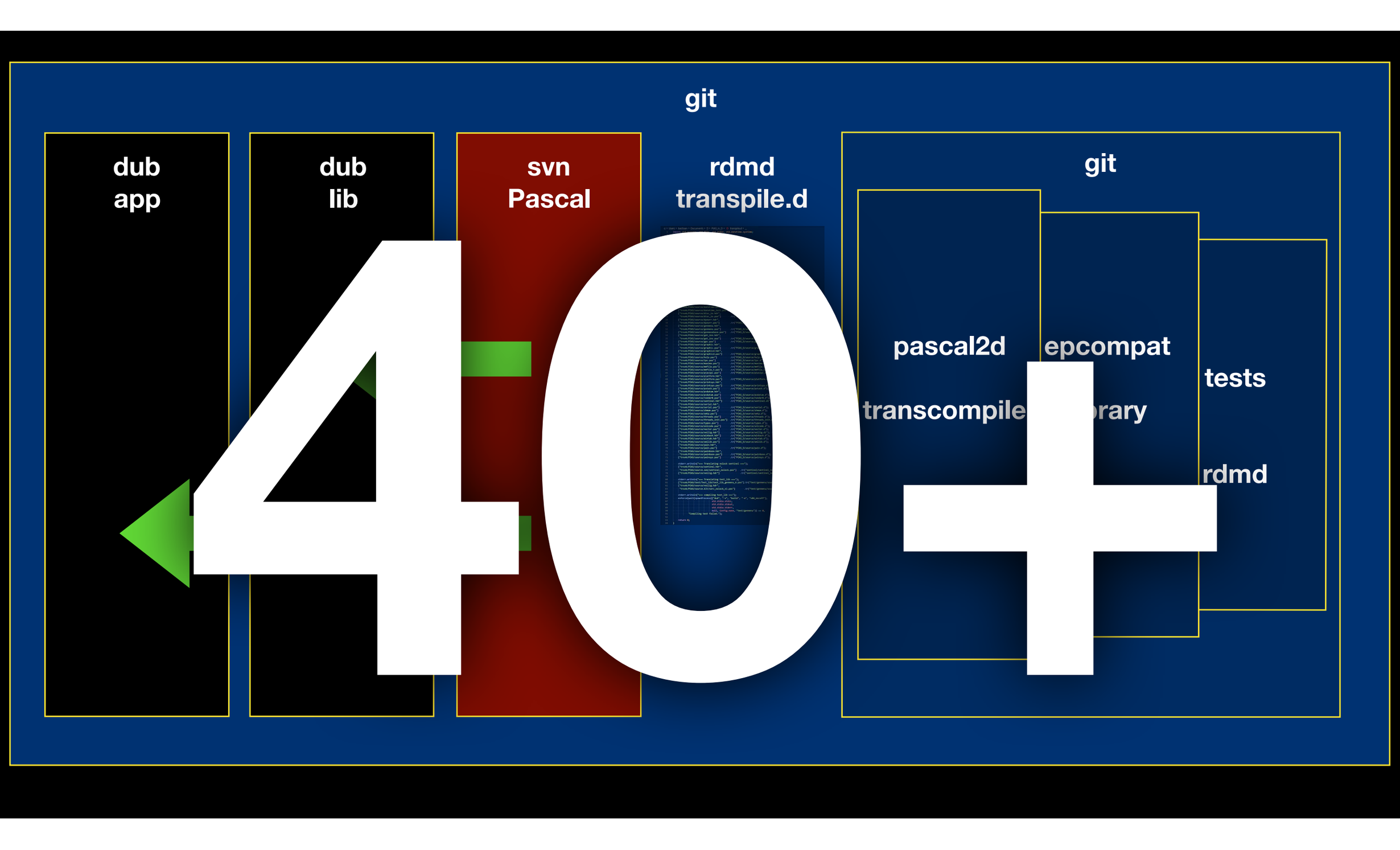

git
dub
app
dub
lib
svn
Pascal
rdmd
transpile.d
git
pascal2d
transcompiler
epcompat
library
tests
73 ····["trunk/PIAS/source/sentinel.hdr"]·····.tr("PIAS_D/source/sentinel.di");·//·interface
74 ····["trunk/PIAS/source/serial.hdr",
75 ·····"trunk/PIAS/source/serial.pas"]·····.tr("PIAS_D/source/serial.d");
76 ····["trunk/PIAS/source/shmem.pas"]·····.tr("PIAS_D/source/shmem.d");




```d
immutable SysTime transpilerTime;
static this() { transpilerTime = timeLastModified("Pascal2D/pascal2d.exe"); }

void tr(string[] from, string to)
in (from.length > 0 && from.length <= 2)
{
    immutable targetTime = timeLastModified(to, SysTime.min);
    if (targetTime > transpilerTime &&
        from.all!(a => targetTime > timeLastModified(a)))
        return;
    stderr.writeln("=== transpiling \"" ~ to.baseName ~ "\" ===");
    (spawnProcess(["Pascal2D/pascal2d.exe"] ~ from, std.stdio.stdin,
                  File(to, "w")).wait == 0).enforce("Parse error.");
    stderr.writeln("=== formatting \"" ~ to.baseName ~ "\" ===");
    (spawnProcess(["dub", "run", "dfmt", "--", "--inplace", to]).wait == 0).enforce;
}
```




```d
    stderr.writeln("=== compiling test_lib ===");
    (spawnProcess(["dub", "-v", "build", "-a", "x86_mscoff"],
                  null, Config.none, "test/genmenu").wait == 0).enforce;

    return 0;
}
```

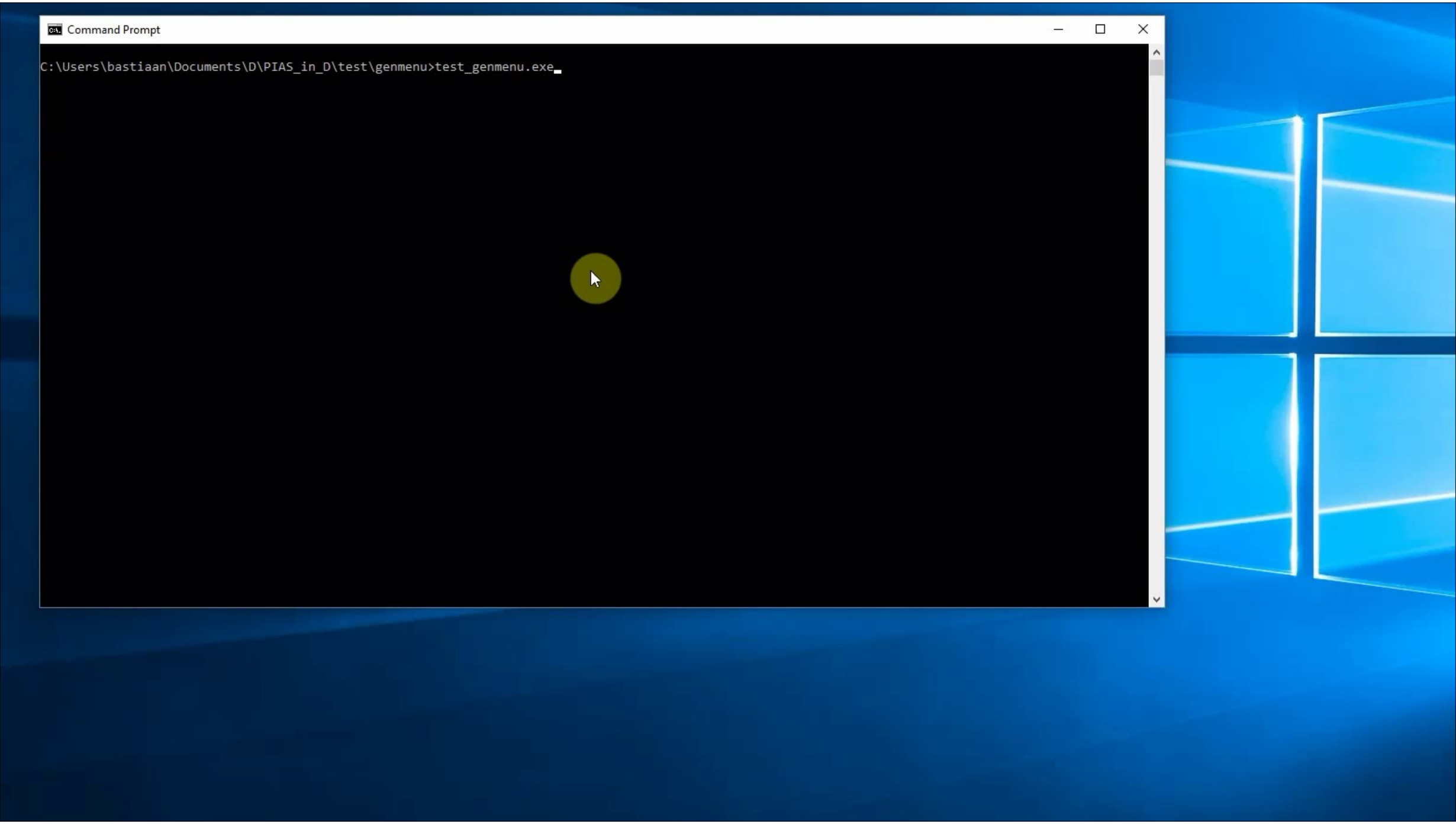

git
dub
app
dub
lib
svn
Pascal
rdmd
transpile.d
git
pascal2d
epcompat
transcompile
tests
rdmd
40+

1. Case sensitivity
2. Use of identifiers that are D keywords
3. Type conversions that are implicit in Pascal but need a cast in D
4. Dealing with mutable strings
5. Calls into the WinAPI that need an `&` on arguments in D

Passes

- read files
- lex
- parse
- create symbol table
- semantic 1
- semantic 2
- semantic 3
- inline
- glue
- optimize
- generate code
- write to object file


Windows systeemfoutmelding
Windows foutnummer 1812 (0x0000000000000714). De omschrijving hiervan luidt: The specified image file did not contain a resource section.
PIAS actie op dat moment: Ywin/Loading PIAS icon.
Dit is geen foutmelding van PIAS, maar van Windows.
OK


Command Prompt
C:\Users\bastiaan\Documents\D\PIAS_in_D\test\genmenu>test_genmenu.exe

Tips & Tricks

```
program test;

procedure foo;
begin
    {Complicated stuff}
end;

begin
end.

```

```
// Program name: test
void foo()
{
    //Complicated stuff
}

void main(string[] args)
{
}

```

```
program test;

procedure foo;
begin
    {$P2D_substitute_begin}
    {Complicated stuff}
    {$P2D_substitute_end}
end;

begin
end.

```

```
// Program name: test
void foo()
{
}

void main(string[] args)
{
}

```

```
program test;

procedure foo;
begin
    {$P2D_substitute_begin
    // Manual cleverness
    }
    {Complicated stuff}
    {$P2D_substitute_end}
end;

begin
end.

```

```
// Program name: test
void foo()
{
    // Manual cleverness
}

void main(string[] args)
{
}

```

```
program test;

procedure foo;
begin
    {$P2D_substitute_begin
    // Manual cleverness
    }
    {Complicated stuff}
    {$P2D_substitute_end}
end;

begin
    {$P2D_insert import std;
    writeln("Running D.");}
end.

```

```
// Program name: test
void foo()
{
    // Manual cleverness
}

void main(string[] args)
{
    import std;

    writeln("Running D.");
}

```

```
program test;

{$P2D_substitute_begin
void foo()
{
    // Manual cleverness
}
}
procedure foo;
begin
    {Complicated stuff}
end;
{$P2D_substitute_end}

begin
end.

```

```
// Program name: test
void foo()
{
    // Manual cleverness
}

void main(string[] args)
{
}

```

```
program test;

{$P2D_substitute_begin
void foo()
{
    assert(false, "TODO");
}
procedure foo;
begin
    {Complicated stuff}
end;
{$P2D_substitute_end}

begin
end.

```

```
// Program name: test
void foo()
{
    assert(false, "TODO");
}

void main(string[] args)
{
}

```

```
program test;

{$P2D_substitute_begin
void foo()
{
    static assert(false, "TODO"
}
procedure foo;
begin
    
end;
{$P2
begi
end.
```

```
// Program name: test
void foo()
{
    static assert(false, "TODO"
}

void main(string[] args)
{
}
```

Command Prompt

```
C:\Users\bastiaan\Documents\D\PIAS_in_D\dconf>dmd test.d
test.d(4): Error: static assert:  "TODO"

C:\Users\bastiaan\Documents\D\PIAS_in_D\dconf>
```

```
1  program test;
2
3  {$P2D_substitute_begin
4  deprecated("TODO")
5  void foo()
6  {
7      assert(false, "TODO");
8  }
9  }
10 procedure foo;
11 begi
12     ...
13 end;
14 {$P2
15
16 begi
17 end.
18
```

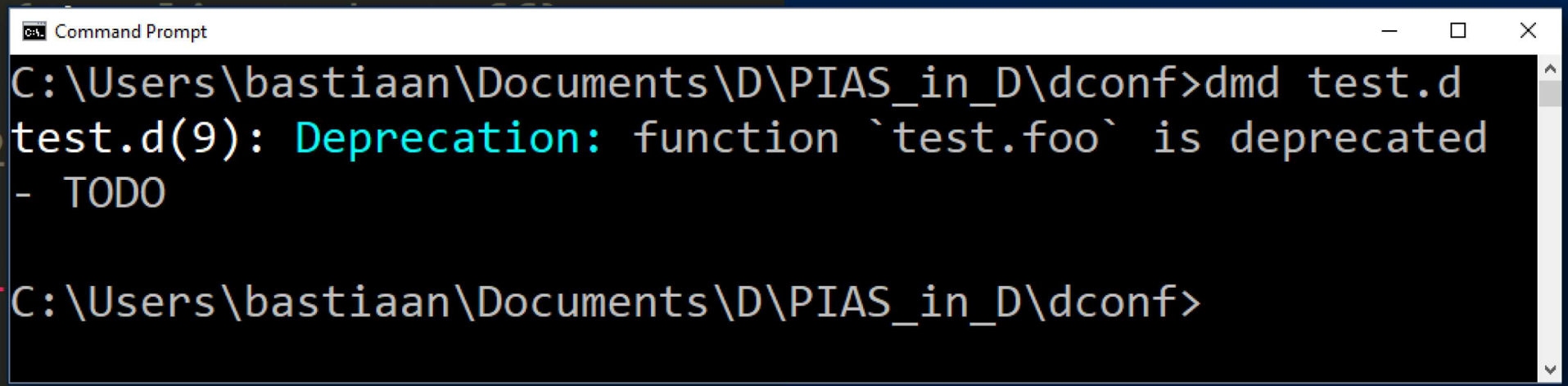
```
1   // Program name: test
2   deprecated("TODO") void foo()
3   {
4       assert(false, "TODO");
5   }
6
7   void main(string[] args)
8   {
9   }
10
```

Command Prompt

```
C:\Users\bastiaan\Documents\D\PIAS_in_D\dconf>dmd test.d

C:\Users\bastiaan\Documents\D\PIAS_in_D\dconf>
```

```
program test;

{$P2D_substitute_begin
deprecated("TODO") void foo()
{
    assert(false, "TODO");
}
procedure foo;
begin
    
end;
{$P2
begi
    
end.
```

```
// Program name: test
deprecated("TODO") void foo()
{
    assert(false, "TODO");
}

void main(string[] args)
{
    foo;
}
```



```pascal
program test(output);

var i, j : boolean;
begin
    if i and j then
        writeln('both true');
end.
```

```
Command Prompt
test ~master: building configuration "application"...
test.d(9,5): Error: static assert:  "Cannot be bitwise factor, AND should be AND_THEN in the Pas
cal soure."
C:\D\dmd2\windows\bin\dmd.exe failed with exit code 1.

C:\Users\bastiaan\Documents\D\PIAS_in_D\dconf>
```

```d
import epcompat;
import std.stdio;
import std.traits : isCallable, ReturnTy

// Program name: test
bool i, j;
void main(string[] args)
{
                                    (isCallable!j && isInte
        "Cannot be bitwise factor, A
    if (i & j)
    {
        writeln("both true");
    }
}
```

```
program test(output);

var i, j : boolean;
begin
    if i and_then j then
        writeln('both true');
end.

```

```
import epcompat;
import std.stdio;

// Program name: test
bool i, j;
void main(string[] args)
{
    if (i && j)
        writeln("both true");
}

```

```
program test(output);

var i : boolean;
begin
    if not i then
        writeln('not i');
end.

```

```
import epcompat;
import std.stdio;
import std.traits : isBoolean, Bool

// Program name: test
bool i;
void main(string[] args)
{
    static assert(is(BooleanTypeOf!
                     (isCallable!i &&
                     "Boolean type tes
    if (!i)
    {
        writeln("not i");
    }
}

```

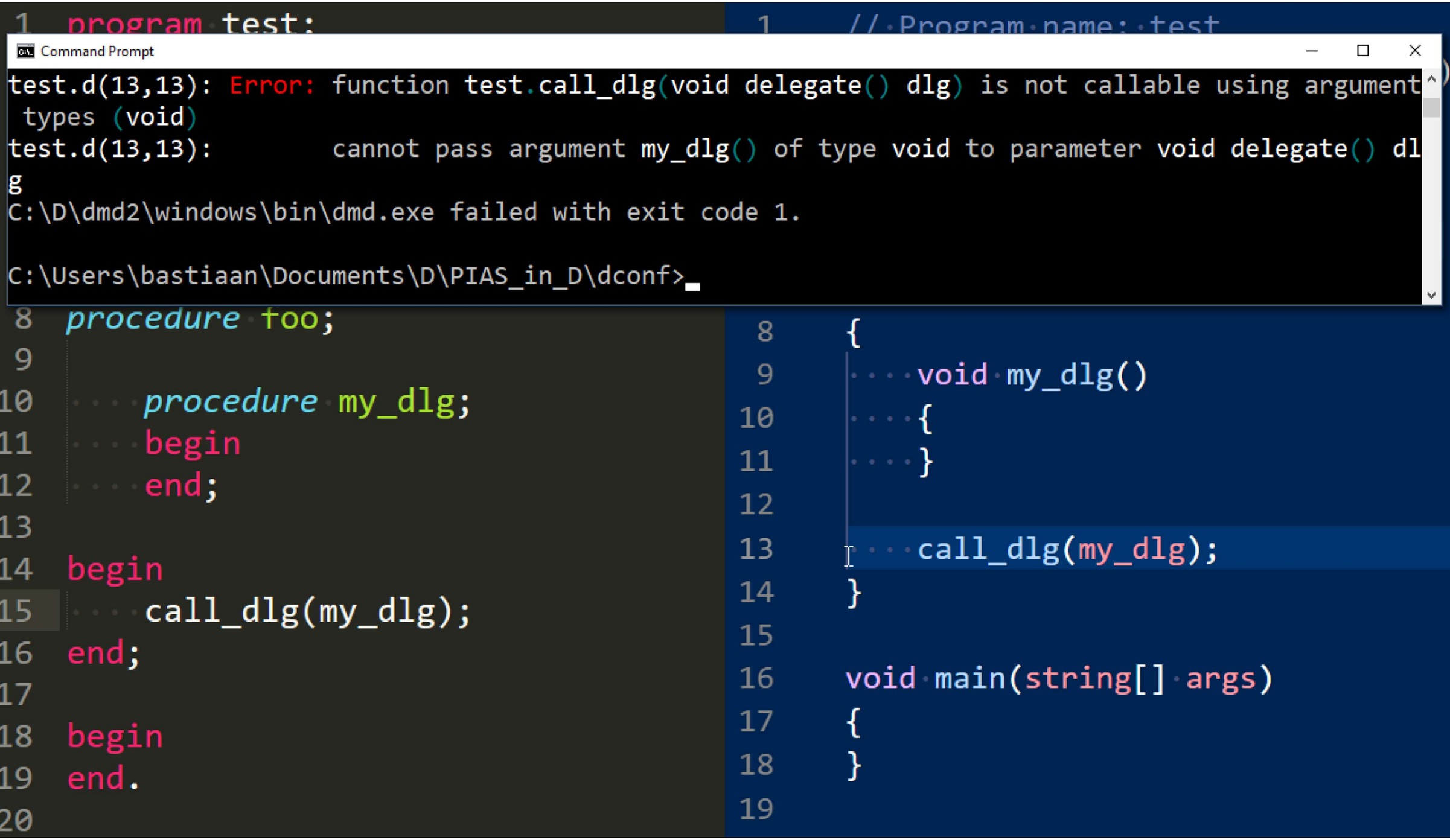
```
program test(output);

var i, j : integer;
begin
    j := not i;
end.

```

```
import std.stdio;
import std.traits : isBoolean, BooleanTypeOf, is

// Program name: test
int i, j;
void main(string[] args)
{
    static assert(is(BooleanTypeOf!(typeof(i)))
                  (isCallable!i && isBoolean!(Re
                  "Boolean type test failed. If
    j = !i;
}
```

```
Command Prompt
test.d(8,5): Error: static assert:  "Boolean type test failed. If it should be Boolean then make
 sure to not use a custom type like "boolean0". To force bitwise negation, use "{$P2D_bitwise}NO
T" in the Pascal soure."
C:\D\dmd2\windows\bin\dmd.exe failed with exit code 1.

C:\Users\bastiaan\Documents\D\PIAS_in_D\dconf>
```

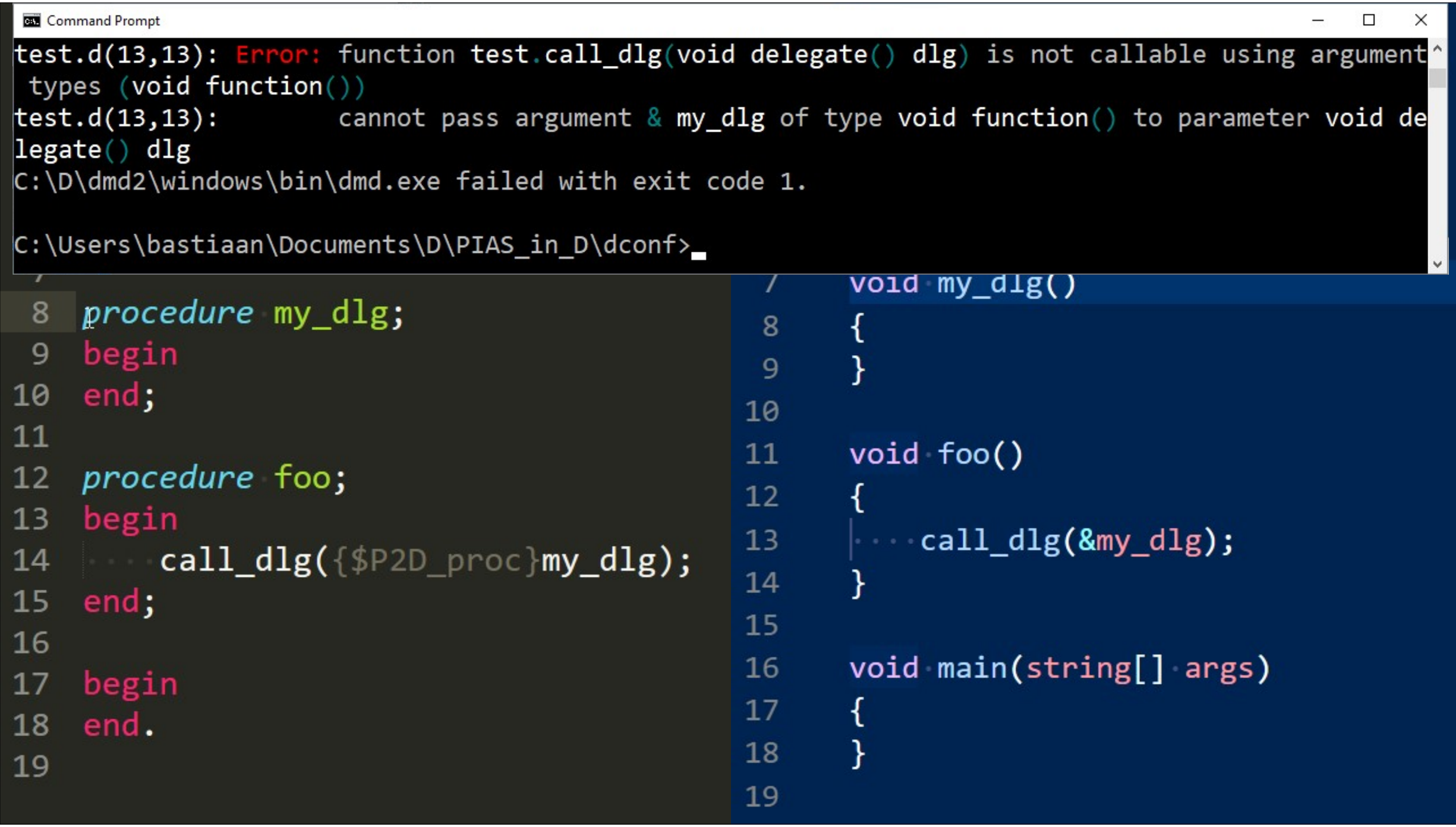
```
program test(output);

var i, j : integer;
begin
    j := {$P2D_bitwise}not i;
end.

```

```
import std.stdio;
import std.traits : isCallable, ReturnTy

// Program name: test
int i, j;
void main(string[] args)
{
    static assert(is(IntegralTypeOf!(typ
                     (isCallable!i && isInt
                     "Cannot be bitwise neg
    j = ~i;
}

```


Command Prompt
test.d(13,13): Error: function test.call_dlg(void delegate() dlg) is not callable using argument types (void)
test.d(13,13): cannot pass argument my_dlg() of type void to parameter void delegate() dlg
C:\D\dmd2\windows\bin\dmd.exe failed with exit code 1.
C:\Users\bastiaan\Documents\D\PIAS_in_D\dconf>
program test;
procedure foo;
procedure my_dlg;
begin
end;
begin
call_dlg(my_dlg);
end;
begin
end.
// Program name: test
{
void my_dlg()
{
}
call_dlg(my_dlg);
}
void main(string[] args)
{
}

```
program test;

procedure call_dlg(procedure dlg);
begin
    dlg;
end;

procedure foo;

    procedure my_dlg;
    begin
    end;

begin
    call_dlg({$P2D_proc}my_dlg);
end;

begin
end.

```

```
// Program name: test
void call_dlg(void delegate() dlg)
{
    dlg;
}

void foo()
{
    void my_dlg()
    {
    }

    call_dlg(&my_dlg);
}

void main(string[] args)
{
}

```


Command Prompt
test.d(13,13): Error: function test.call_dlg(void delegate() dlg) is not callable using argument types (void function())
test.d(13,13): cannot pass argument & my_dlg of type void function() to parameter void delegate() dlg
C:\D\dmd2\windows\bin\dmd.exe failed with exit code 1.
C:\Users\bastiaan\Documents\D\PIAS_in_D\dconf>
8 procedure my_dlg;
9 begin
10 end;
11
12 procedure foo;
13 begin
14 call_dlg({$P2D_proc}my_dlg);
15 end;
16
17 begin
18 end.
19
7 void my_dlg()
8 {
9 }
10
11 void foo()
12 {
13 call_dlg(&my_dlg);
14 }
15
16 void main(string[] args)
17 {
18 }
19

```
program test;

procedure call_dlg(procedure dlg);
begin
    dlg;
end;

procedure my_dlg;
begin
end;

procedure foo;
begin
    call_dlg({$P2D_toDelegate}my_d
end;

begin
end.

```

```
import std.functional : toDelegate;

// Program name: test
void call_dlg(void delegate() dlg)
{
    dlg;
}

void my_dlg()
{
}

void foo()
{
    call_dlg(toDelegate(&my_dlg));
}

void main(string[] args)
{
```

{$P2D_preserve_index}

{$P2D_UDA}

{$P2D_string}

{$P2D_cat}

{$P2D_array}

{$P2D_packed}

{$P2D_method}

Strings

1. Mutability (in place manipulation and passing to WinAPI)
2. Indexing
3. file i/o
4. winAPI (UTF16 W vs A, taking address of slice, lifetime of toUTF16z)
5. fixed capacity
6. String pointers and newing strings

Take any small opportunity
do your best
to make it a bigger one