D lang Lexer

Gleb Popov, Timur Usmanov
Compiler Construction
Innopolis University
Fall 2025

Project Context

Dynamic Lang

object types are not specified and can change while program execution

the language assumes **interpretation**

C++ Language

the implementation language is C++

it provides extensive memory management and optimization features

Personal parser

hand-written parser

if you want a thing done well, do it yourself :)

Tokens

Lexical Analyzer splits the initial code into tokens.

Tokens

Lexical Analyzer splits the initial code into tokens.

```
var x := 5
print x
```

Tokens

Lexical Analyzer splits the initial code into tokens.

```
var x := 5
print x
```

```
{tkVar, tkIdent(x), tkAssign, tkIntLiteral(5),
tkNewLine, tkPrint, tkIdent(x)}
```

Tokens

Lexical Analyzer splits the initial code into tokens.

```
var x := 5
print x
```

```
{tkVar, tkIdent(x), tkAssign, tkIntLiteral(5),
tkNewLine, tkPrint, tkIdent(x)}
```

Token

```
Span span;
// {pos=0, len=3}, {pos=6, len=2}, ...

Type type;
// tkVar, tkAssign, ...

vector<pair<string, Type>> typeChars;
// <"var", "tkVar">, <":=", tkAssign>, ...
```

Tokens

Lexical Analyzer splits the initial code into tokens.

```
var x := 5
print x
```

```
{tkVar, tkIdent(x), tkAssign, tkIntLiteral(5),
tkNewLine, tkPrint, tkIdent(x)}
```

Token

```
Span span;
// {pos=0, len=3}, {pos=6, len=2}, ...

Type type;
// tkVar, tkAssign, ...

vector<pair<string, Type>> typeChars;
// <"var", "tkVar">, <":=", tkAssign>, ...
```

StringLiteral	`string value`
Identifier	`string identifier`
Integer	`string value`
Real	`string value`

Lexer

Token

```
Span span;
// {pos=0, len=3}, {pos=6, len=2}, ...

Type type;
// tkVar, tkAssign, ...

vector<pair<string, Type>> typeChars;
// <"var", "tkVar">, <":=", tkAssign>, ...
```

Lexer

```
vector<Token> tokenize(file, log);

checkComments();
checkStringLiterals();
checkNumbers();
checkToken();
checkIdentifierToken();
```

Examples

```
var x := 5
print x
```

```
tkVar tkIdent(”x”) tkAssign tkIntLiteral(5) tkNewLine
tkPrint tkIdent tkNewLine
```

```
var t := {x:=1}
t := t + {y:=2}
```

```
tkVar tkIdent(t) tkAssign tkOpenCurlyBrace tkIdent(“x”)
tkAssign tkIntLiteral(1) tkClosedCurlyBrace tkNewLine
tkIdent(“t”) tkAssign tkIdent(“t”) tkPlus tkOpenCurlyBrace
tkIdent(“y”) tkAssign tkIntLiteral(2) tkClosedCurlyBrace
```

```
var x := 3
if x < 10 then
    print "small"
else
    print "big"
end
```

```
tkVar tkIdent(“x”) tkAssign tkIntLiteral(3) tkNewLine tkIf
tkIdent(“x”) tkLess tkIntLiteral(10) tkThen tkNewLine
tkPrint tkStringLiteral(“small”) tkNewLine tkElse tkNewLine
tkPrint tkStringLiteral(“big”) tkNewLine tkEnd tkNewLine
```