Kompilatory - Go2LLVM Paweł Płatek

Informatyka, 3 rok, 2018

1 Gramatyka

```
LEXER

string_tok = '"' {'\"' | unicode_value | byte_value} '"'

unicode_value    = unicode_char | escaped_char .
unicode_char = /* an arbitrary Unicode code point except newline */ .

escaped_char     = `\` ( "a" | "b" | "f" | "n" | "r" | "t" | "v" | `\` | "'" | `"` ) .

letter           = unicode_letter | '_'
unicode_letter = /* long list of unicode letters */;

byte_value       = octal_byte_value | hex_byte_value
octal_byte_value = '\' octal_digit octal_digit octal_digit;
hex_byte_valye = '\x' hex_digit hex_digit

bool_tok = true | false
int_tok = (1..9) {(1..9)} | 0 {(0..7)} | 0 (x | X) {(0..9 | a..f | A..F)}
float_tok = {(1..9)} '.' [{(1..9)}] | '.' [{(1..9)}]

unary_op_tok = ('+' | '-' | '!' | '*' | '&')
binary_op_tok = ('||' | '&&' | '==' | '!=' | '<' | '<=' | '>' | '>=' | '+' | '-' | '|' | '^'

//keywords
package_tok = 'package'
var_tok = 'var'
func_tok = 'func'
return_tok = 'return'
if_tok = 'if'
ekse_tok = 'else'
ident_tok =  (a..z | _) {(a..z | _) | (0..9)}

// Whitespace and comments
line_comment = '/' [{.*}]
comment = '/*' [{.*}] '*/'
whitespace = '\t '
statement_termination = (';' | '\n' | comment)


PARSER

// top level stuff
```

```
SourceFile = PackageClause Eos { TopLevelDecl Eos }
PackageClause = "package" ident_tok

TopLevelDecl = Declaration | FunctionDecl

Block = "{" StatementList "}"
Type = [*] IDENT_TOK
EOS = statement_termination

// Statements
StatementList = { Statement EOS }
Statement = Declaration | SimpleStmt | Block | ReturnStmt | IfStmt

IfStmt = 'if' [SimpleStmt ';'] Expression Block ['else' (IfStmt | Block)]
ReturnStmt = "return" [ExpressionList]

SimpleStmt = EmptyStmt | Expression | Assignment
Assignment = identifierList [('+' | '-' | '|' | '^' | '*' | '/' | '%' | '<<' | '>>' | '&')]
EmptyStmt = ";"


// Declarations
Declaration = "var" IdentifierList Type ["=" ExpressionList]
        IdentifierList = ident_tok { "," ident_tok }


// Expressions
Expression = UnaryExpr | Expression binary_op_tok Expression
binary_op_tok=('*' | '/' | '%' | '<<' | '>>' | '&' | '&^' | '+' | '-' | '|' | '^' | '==' | '

UnaryExpr = unary_op_tok UnaryExpr | operand
unary_op_tok=('*' | '&' | '+' | '-' | '!' | '^')

ExpressionList = Expression { "," Expression }


// Operands
Operand = basicLit | ident_tok | ident_tok Arguments | "(" Expression ")"

Arguments '(' [ExpressionList [',']] ')'

BasicLit = int_tok | float_tok | imag_tok  | string_tok | bool_tok


// Functions
Function = func_tok ident_tok signature [block]
Signature = parameters [result]
Result = type
```

```
Parameters = '(' [parameterList [',']] [...] ')'
ParameterList = parameterDecl { COMMA parameterDecl }
ParameterDecl = identifielList type
```

2 Język

1. packages 3. exported names 4. functions - declaration - definition - arguments - return values - externs 5. variables - declaration - types - initializers - constants 6. types 7. casting 8. control flow - if/the/else 9. pointers

3 Build'n'Run

4 LLVM

```
millis() - dispTime[i]*codeChangeCounter[i] - totalTimePaused >= dispTime[i]
```

5 Listing kodu

```
1 *
2  ~Paweł Płatek
3 /
```