Universidad Nacional Autónoma de México
Semestre 2021-1
Compiladores
Gramática
Profesor: Adrián Ulises Mercado Martínez
Programa 1

Para la gramática de la sección de gramática elaborar un analizador léxico que reconozca cada uno de los tokens que puede forman parte del alfabeto para el lenguaje generado por la gramática.

Es altamente recomendable que el analizador léxico solo retorne un entero por cada token encontrado. En el caso de los operadores y de las palabras reservadas se recomienda devolver un entero por cada uno de los componentes dentro de este grupo.

Usar la siguiente descripción de algunos de los tokens:

1. Identificadores: son los nombres de las variables y de los tipos o funciones. Un identificador es la secuencia de una o mas letras y dígitos. Los identificadores deben comenzar con una letra

```
a
_x9
ThisVariableIsExported
```

2. Enteros: Los enteros son secuencias de dígitos como los ejemplos a continuación

```
  42
  4_2  // lo mismo que 42
  170141183460469231731687303715884105727
  170_141183_460469_231731_687303_715884_105727

  _42         // Es un identificador no un entero
  42_         // invalido: debe separar dígitos susecivos
  4__2        // invalido: sólo se permite un guión bajo a la vez
```

3. Los decimales: son números que incluyen un punto decimal y además pueden incluir la notación científica.

```
0.
72.40
072.40       // == 72.40
2.71828
1.e+0
6.67428e-11
1E6
.25
.12345E+5
1_5.         // == 15.0
0.15e+0_2    // == 15.0
```

4. Imaginarios: es parte de un número complejo que representa la parte imaginaria. Consisten de un entero o un flotante seguidos de la letra minúscula i

```
0.i
2.71828i
1.e+0i
6.67428e-11i
1E6i
.25i
.12345E+5i
```

5. Cadenas: las cadenas se forman por secuencias de caracteres.

```
`abc`                // lo mismo que "abc"
`\n
\n`                  // lo mismo que "\\n\n\\n"
"\n"
"\""                 // lo mismo que `"`
"Hello, world!\n"
"[U+65E5][U+672C][U+8A9E]"
"\u65e5[U+672C]\U00008a9e"
"\xff\u00FF"
```

6. Constantes booleanas: true y false

Gramática

PRODUCCIÓN
Program → PackageClause Declarations FunctionDecls
PackageClause → **package identifier**
Declarations → Declarations Declaration | ε
Declaration → ConstDecl | VarDecl
VarDecl → **var** VarSpec | **var (** VarSpecs **)**
VarSpecs → VarSpecs VarSpec | VarSpec
VarSpec → Type IdentifierList
ConstDecl →**const** ConstSpec | **const (** ConstSpecs **)**
ConstSpecs → ConstSpecs ConstSpec | ConstSpec
ConstSpec → Type IdentifierList **=** ExpressionList
IdentifierList → **identifier ,** IdentifierList | **identifier**
ExpressionList → Expression **,** ExpressionList | Expression
Type → TypeName | TypeLit | **(** Type **)**
TypeName → **uint8** | **uint16** | **int8** | **int16** | **int32** | **float32** | **float64** | **complex64** | **byte** | **string** | **bool**
TypeLit → ArrayType | StructType | PointerType| SliceType
SliceType → **()** Type
ArrayType → **(** Expression **)** Type
StructType → **struct** { FieldDecl }
FieldDecl → Type IdentifierList | EmbeddedField IdentifierList
EmbeddedField → ***** TypeName | TypeName
PointerType → ***** Type
FunctionDecl → **func identifier** ParameterList Type FunctionBody
Parameters → **(** ParameterList **)**
ParameterList → ParameterDecls | ε
ParameterDecls → ParameterDecls ParameterDecl | ParameterDecl
ParameterDecl → Type IdentifierList
FunctionBody → Block
Block → { StatementList }
StatementList → StatementList Statement | Statement
Statement → Declaration | SimpleStmt | ReturnStmt | BreakStmt | ContinueStmt
| Block | IfStmt | SwitchStmt | ForStmt |DeferStmt
SimpleStmt → ε | Expression | IncDecStmt | Assignment
Assignment → Expression assign_op Expression
DeferStmt → **defer** Expression
IncDecStmt → Expression **++** | Expression −−
assign_op → add_op**=** | mul_op**=** | **=**
IfStmt → **if** SimpleStmt Expression Block | **if** SimpleStmt Expression Block **else** Block
SwitchStmt → **switch** SimpleStmt { ExprCaseClauses }
ExprCaseClauses → ExprCaseClauses ExprCaseClause | ExprCaseClause

ExprCaseClause → ExprSwitchCase ":" StatementList
ExprSwitchCase → **case** ExpressionList | **default**
ForStmt → **for** ForClause Block
Condition → Expression
ForClause → InitStmt **;** Condition **;** PostStmt
InitStmt → SimpleStmt
PostStmt → SimpleStmt
ReturnStmt → **return** ExpressionList | **return**
BreakStmt → **break**
ContinueStmt → **continue**
Expression → UnaryExpr | Expression binary_op Expression
UnaryExpr → PrimaryExpr | unary_op UnaryExpr
PrimaryExpr → Operand | Conversion | PrimaryExpr Selector | PrimaryExpr Index
| PrimaryExpr Slice | CallFunction
Selector → **.** **identifier**
Index → **(** Expression **)**
Slice → **(** Expression **:** Expression **)** | **(** Expression **:** Expression **:** Expression **)**
CallFunction → **identifier** Arguments
Arguments → **(** ExpressionList **)** | **()**
ExpressionList → Expression , ExpressionList | Expression
binary_op → || | && | rel_op | add_op | mul_op rel_op → **==** | **!=** | < | <= | > | >=
add_op → **+** | **-**
mul_op → ***** | **/** | **%**
unary_op → **+** | **-** | **!** | ***** | **&**
Conversion → Type **(** Expression **)**
Operand → Literal | **identifier** | **(** Expression **)**
Literal → **int_lit** | **float_lit** | **imaginary_lit** | **string_lit** | **bool_lit**