**Keywords**

|  |  |
| --- | --- |
| **Keyword** | **Function** |
| def | Declares the definition of a function. |
| return | End the execution of a function and returns the control to the calling function. It can also return a value to the calling function. |
| var | Marks the declaration of a variable. |
| if | Marks an if-statement. |
| else | Marks an else-statement. |
| elif | Marks an if-else-statement. |
| and | A keyword used as a logical operator. It checks 2 expressions and returns a binary value, i.e. false if one of the expressions is false and true if both expressions are true. |
| or | A keyword used as a logical operator. It checks 2 expressions and returns a binary value, i.e. true if one of the expressions is true and returns false otherwise. |
| null | It is used as a placeholder that represents “empty” or absence of a value. |
| true | Holds a binary value of 1 or true. |
| false | Holds a binary value of 0 or false. |

**Identifier Rules**

* Identifiers can be a combination of lowercase letters [a,b,c, … ,x,y,z], uppercase letters [A,B,C, … X,Y,Z] and digits [0,1,2, … ,7,8, 9].
* Identifiers cannot start with digits.
* All keywords of Turtle cannot be used as identifiers by the user.
* No special characters such as !, %, &, #, @, etc. are allowed.
* Identifiers are case sensitive.
* Turtle recommends Camel Casing for identifiers, but Pascal Casing can also be used.
* (As of now) Identifiers can be of any length.
* “def” should be used before a function-identifier, when defining a function.
* “var” should be used before variable-identifier, when declaring a variable.

**Partial List of Tokens**

* Identifier
* Keyword
* Operator
* Delimiter
* Literal (Number)
* Newline

**Code (.l file of Flex)**

%{

#include <stdio.h>

#undef yywrap

#define yywrap() 1

void showError();

%}

%%

(def|return|var|if|else|elif|and|or|null|true|false) {

    printf("KEYWORD\n");

}

[a-zA-Z][a-zA-Z0-9]\* {

    printf("IDENTIFIER\n");

}

[0-9][0-9]\* {

    printf("NUMBER LITERAL\n");

}

[ \t\n] ;

(;|\,|[|]) {

    printf("DELIMITER\n");

}

(=) {

    printf("ASSIGNMENT OPERATOR\n");

}

. {

    showError();

}

%%

int main()

{

    yylex();

}

void showError(){

    printf("INVALID CHARACTER\n");

}

**Commands to Run the .l Code**

|  |  |
| --- | --- |
| **Generate lexer** | flex lex\_generator.l |
| **Compile the generated C code** | gcc lex.yy.c |
| **Run the lexer** | ./a.exe |

**Lexical Analyzer’s** **Output Examples**

|  |  |
| --- | --- |
|  | The lexer generates tokens for the first line “var myAge = 20” |
|  | Lexer returns the type of token for “90” |
|  | The lexer return “INVALID CHARACTER” as I haven’t given rules for operators. |
|  | Lexer identifying the identifiers correctly. |

**Context Free Grammar (CFG)**

statements -> eps | statement newline statement

statement -> keyword identifier assignment expression

| func\_def delimiter statements delimiter

| identifier assignment expression

| keyword delimiter expression operator expression delimiter

| identifier assignment func\_def delimiter stmts delimiter

| keyword identifier assignment func

func\_def -> keyword identifier delimiter [identifier]\* delimiter

func -> identifier delimiter delimiter

identifier -> [a-zA-Z][a-zA-Z0-9]\*

expression -> literal operator literal | literal

literal -> num\_literal

num\_literal -> [0-9][0-9]\*

assignment -> =

delimiter -> ; | , | [ | ] | { | } | ( | )

operator -> + | - | \* | / | %

keyword -> def | return | if | else | elif | var | true | false | null | and | or

**Code Examples for CFG’s**

|  |  |
| --- | --- |
| statement -> keyword identifier assignment expression | var myAge = 20  or  var sum = 20 + 2 |

|  |  |
| --- | --- |
| statement -> func\_def delimiter statements delimiter | def foo() {  var a = 20  } |

|  |  |
| --- | --- |
| statement -> identifier assignment expression | age = 30 |

|  |  |
| --- | --- |
| statement -> keyword delimiter expression operator expression delimiter | if ( 20 + 2 ) |

|  |  |
| --- | --- |
| statement -> identifier assignment func\_def delimiter stmts delimiter | var myFunc = def foo() {  var a = 20  } |

|  |  |
| --- | --- |
| statement -> keyword identifier assignment func | var myFunction = foo() |

|  |  |
| --- | --- |
| func\_def -> keyword identifier delimiter [identifier]\* delimiter | def foo()  or  def foo(a) |

|  |  |
| --- | --- |
| expression -> literal operator literal | literal | 20 + 100  or  10 |

|  |  |
| --- | --- |
| func -> identifier delimiter delimiter | foo() |

|  |  |
| --- | --- |
| identifier -> [a-zA-Z][a-zA-Z0-9]\* | Variable  or  var100  or  myAge |

|  |  |
| --- | --- |
| func\_def -> keyword identifier delimiter [identifier]\* delimiter | def funct() |