Custom Programming Language – Lexical Analysis and Parsing

Lexical analysis is the process of converting a sequence of characters from source program into a sequence of tokens. A program which performs lexical analysis is termed as a lexical analyzer (lexer), tokenizer or scanner. In this project, a custom programming is designed and lexically analysed and parsed.

**Lexical Analysis**

The lexical analysis consists of the following two steps:

* Scanning
* Tokenization

The working depiction of the lexical analyzer is given below

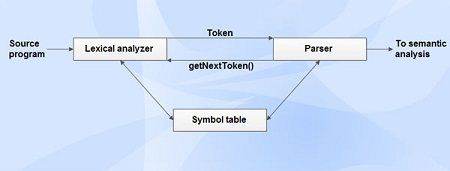


Figure : Workflow of Lexical Analysis

**Custom Programming Language**

The programming language consists of the grammar, identifiers, operators, etc. mainly. The overview of the custom programming language developed in this project is as under:

**Tokens**

toktyp["$"] = "Integer type declarator";

toktyp["@"] = "String type declarator";

toktyp["whiley"] = "Loop construct - while";

toktyp["iff--"] = "If - Conditional construct";

toktyp["eelse--"] = "Else Conditional construct";

toktyp["(("] = "Parentheses begin";

toktyp["))"] = "Parentheses end";

toktyp["{{"] = "Braces begin";

toktyp["}}"] = "Braces end";

toktyp["++"] = "Arithmetic operator - addition OR String Concatenation";

toktyp["--"] = "Arithmetic operator - subtraction";

toktyp["\*"] = "Arithmetic operator - division";

toktyp["/"] = "Arithmetic operator - multiplication";

toktyp["%%"] = "Arithmetic operator - modulo";

toktyp[">>"] = "Relational operator - greater than";

toktyp["<<"] = "Relational operator - less than";

toktyp["==="] = "Relational operator - equality";

toktyp["=="] = "Assignment operator";

toktyp["+\*\*"] = "Increment operator";

toktyp["-\*\*"] = "Decrement operator";

toktyp["nomain"] = "Main function declarator";

toktyp["returnn"] = "Function return statement";

toktyp["empty"] = "Void function declarator";

toktyp[";;"] = "End\_of\_statement representor";

toktyp["#"] = "Character type declarator";

**Operators**

Binary Operator:

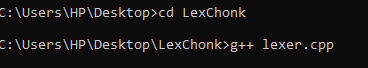
str=="++" || str=="--" || str=="\*\*" || str=="//" || str=="%%")

Unary Operator:

str=="+\*\*" || str=="-\*\*"

**Code Implementation**

**Compiling:**



**Input File:**



**Output:**

out.txt file in the directory

**Executions:**

**Input 1**

$ nomain(){

^ Code to find the sum of first 10 natural numbers

$ i == 1;

$ sum == 0

while(i<<11){

sum = sum ++ i;

+\*\*i;

}

returnn# 0;

}

**Output**

<Integer type declarator, $> on line number 1

<Identifier, nomain> on line number 1

<Parentheses begin, (> on line number 1

<Parentheses end, )> on line number 1

<Braces begin, {> on line number 1

<Integer type declarator, $> on line number 3

<Identifier, i> on line number 3

<Relational operator - equality, ==> on line number 3

<int\_literal, 1> on line number 3

<End\_of\_statement representor, ;> on line number 3

<Integer type declarator, $> on line number 4

<Identifier, sum> on line number 4

<Relational operator - equality, ==> on line number 4

<int\_literal, 0> on line number 4

<Loop construct - while, while> on line number 5

<Parentheses begin, (> on line number 5

<Identifier, i> on line number 5

<Relational operator - less than, <> on line number 5

<Relational operator - less than, <> on line number 5

<int\_literal, 11> on line number 5

<Parentheses end, )> on line number 5

<Braces begin, {> on line number 5

<Identifier, sum> on line number 6

<Assignment operator, => on line number 6

<Identifier, sum> on line number 6

<Increment operator, ++> on line number 6

<Identifier, i> on line number 6

**.. up to so on.**

**Input 2**

$ nomain(()){{

^Code to check if user specified number is odd or even

@ result == "odd";;

$ num == 0;; ^User needs to enter the number here

$ modval == num%2;;

iff--((modval===0)){{

result=="even\_number here";;

}}

eelse--{{

result=="odd\_number here";;

}}

returnn 0;;

}}

**Output**

<Integer type declarator, $> on line number 1

<Identifier, nomain> on line number 1

<Parentheses begin, (> on line number 1

<Parentheses begin, (> on line number 1

<Parentheses end, )> on line number 1

<Parentheses end, )> on line number 1

<Braces begin, {> on line number 1

<Braces begin, {> on line number 1

<String type declarator, @> on line number 3

<Identifier, result> on line number 3

<Relational operator - equality, ==> on line number 3

<String Literal, "odd"> on line number 3

<End\_of\_statement representor, ;> on line number 3

<End\_of\_statement representor, ;> on line number 3

<Integer type declarator, $> on line number 4

<Identifier, num> on line number 4

<Relational operator - equality, ==> on line number 4

<int\_literal, 0> on line number 4

<End\_of\_statement representor, ;> on line number 4

<End\_of\_statement representor, ;> on line number 4

<Integer type declarator, $> on line number 5

<Identifier, modval> on line number 5

<Relational operator - equality, ==> on line number 5

<Identifier, num> on line number 5

<Arithmetic operator - modulo, %> on line number 5

<int\_literal, 2> on line number 5

<End\_of\_statement representor, ;> on line number 5

<End\_of\_statement representor, ;> on line number 5

<Identifier, iff> on line number 6

<Decrement operator, --> on line number 6

<Parentheses begin, (> on line number 6

<Parentheses begin, (> on line number 6

<Identifier, modval> on line number 6

<Relational operator - equality, ==> on line number 6

<Assignment operator, => on line number 6

<int\_literal, 0> on line number 6

**.. up to so on.**

**Input 3**

$ fact( $ n) {

iff-- (n <<1){

returnn 1;;

}

elsee iff (n == 1){

returnn 1;;

}

elsee{

returnn n\*\*fact(n--1);;

}

}

$ main(){

$ x;

x == 1;

while (x << 11) {

write(x);

write(fact(x));

writeln();

x = x ++ 1;

}

returnn 0;

}

**Output**

<Integer type declarator, $> on line number 1

<Identifier, fact> on line number 1

<Parentheses begin, (> on line number 1

<Integer type declarator, $> on line number 1

<Identifier, n> on line number 1

<Parentheses end, )> on line number 1

<Braces begin, {> on line number 1

<Identifier, iff> on line number 2

<Decrement operator, --> on line number 2

<Parentheses begin, (> on line number 2

<Identifier, n> on line number 2

<Relational operator - less than, <> on line number 2

<Relational operator - less than, <> on line number 2

<int\_literal, 1> on line number 2

<Parentheses end, )> on line number 2

<Braces begin, {> on line number 2

<Identifier, returnn> on line number 3

<int\_literal, 1> on line number 3

<End\_of\_statement representor, ;> on line number 3

<End\_of\_statement representor, ;> on line number 3

<Braces end, }> on line number 4

<Identifier, elsee> on line number 5

<Identifier, iff> on line number 5

<Parentheses begin, (> on line number 5

<Identifier, n> on line number 5

<Relational operator - equality, ==> on line number 5

<int\_literal, 1> on line number 5

<Parentheses end, )> on line number 5

<Braces begin, {> on line number 5

<Identifier, returnn> on line number 6

<int\_literal, 1> on line number 6

<End\_of\_statement representor, ;> on line number 6

<End\_of\_statement representor, ;> on line number 6

<Braces end, }> on line number 7

<Identifier, elsee> on line number 8

**.. up to so on.**