**Objectives:**

* Develop a tokenization program using Flex.
* To design and implement a lexical analyzer using Flex.
* To create a set of regular expressions that can recognize different tokens in a given programming language.
* To create a syntax for a unique programming language
* To verify the correctness and accuracy of the lexical analyzer's output against expected results.

**Introduction:**

FLEX (Fast Lexical analyzer generator) is a tool for generating scanners. Lexical analysis is the first phase of a compiler. It takes the modified source code from language preprocessors that are written in the form of sentences. The lexical analyzer breaks these syntaxes into a series of tokens, by removing any whitespace or comments in the source code. Scanners perform lexical analysis by dividing the input into meaningful units. For a C program the units are variables, constants, keywords, operators, punctua5on etc. These units also called as tokens. . These tokens are then passed to the parser for further processing.

**Flex (.l) file:**

%{

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

int identifier = 0,keyword = 0,user\_func = 0,datatype = 0,loop = 0,cndstatement = 0,statement = 0;

struct Symbol {

char name[256];

};

struct Symbol symbol\_table[1000];

int symbol\_count = 0;

int isSymbolDeclared(char\* name) {

for (int i = 0; i < symbol\_count; i++) {

if (strcmp(symbol\_table[i].name, name) == 0) {

return 1;

}

}

return 0;

}

%}

D\_type "purno"|"vogno"|"string"

identifier [a-zA-Z\_][0-9a-zA-Z\_]\*

digit [0-9]

arith\_op "+"|"-"|"\*"|"/"|"^"

un\_op "++"|"--"

lg\_op "and"|"or"|"not"

rel\_op "<"|">"|"<="|">="|"=="|"!="

assign\_op "="|"+="|"-="|"/="|"\*="

lb\_func "cin>>"|"cout<<"|"cin >>"|"cout <<"

us\_func "func"

con\_state "if"|"else"|"else if"

lop "loop "[0-9]+".."[0-9]+

purno (-|\+)?[0-9]+

vogno [+-]?[0-9]+[.][0-9]+|[+-]?[.][0-9]+|[1-9]+e[+-][0-9]+

key "break"|"continue"|"return"|"void"|"main"

par "("|")"|"{"|"}"|"["|"]"

%%

[/][/](.\*|\n)\*[/][/] printf("Multi Line Comment ");

[#][#](.\*|" ")\* printf("Single Line Comment ");

[;] {

printf(";");

statement++;

}

["].\*["] {

printf("Literal");

}

{D\_type} {

printf("Datatype ");

datatype++;

}

{key} {

printf("keyword ");

keyword++;

}

{par} {

printf("Parenthesis ");

}

{arith\_op} {

printf("arithmatic\_operator ");

}

{un\_op} {

printf("unary\_operator ");

}

{lg\_op} {

printf("logical\_operator ");

}

{rel\_op} {

printf("relational\_operator ");

}

{assign\_op} {

printf("assignment\_operator ");

}

{lb\_func} {

printf("library\_function : ");

}

{us\_func} {

printf("user\_function ");

user\_func++;

}

{lop} {

printf("Loop ");

loop++;

}

{con\_state} {

printf("conditional\_statement ");

cndstatement++;

}

{purno} {

printf("Integer ");

}

{vogno} {

printf("Float ");

}

{identifier} {

printf("identifier ");

identifier++;

}

%%

int yywrap()

{

return 1;

}

int main()

{

yyin=freopen("input.txt","r",stdin);

//yyout=freopen("output.txt","w",stdout);

yylex();

printf("\nNumber of tokens:\n");

printf("\nIdentfiers = %d\nStatements = %d\nUser function = %d\nDatatype = %d\nLoop = %d\nConditional statement = %d\nKeywords = %d\n",identifier,statement,user\_func,datatype,loop,cndstatement,keyword);

}

**Input.txt:**

//multi line

comment

//

## this is single line comment

purno main(){

purno a,b, c;

vogno d;

string x ;

purno x;

a=56 , b= 45;

d = 4.76 ;

if ( a <= b and c!= a)

{

cout<< a;

}else

{

cin >> c;

cout << c;

}

}

func test(){

loop 1..5

{

cout << "hello";

}

}

**Output.txt:**

Multi Line Comment

Single Line Comment

identifier keyword Parenthesis Parenthesis Parenthesis

Datatype identifier ,identifier , identifier ;

Datatype identifier ;

Datatype identifier , identifier ;

identifier assignment\_operator Integer , identifier assignment\_operator Integer ;

identifier assignment\_operator Float ;

conditional\_statement Parenthesis identifier relational\_operator identifier logical\_operator identifier relational\_operator identifier Parenthesis

Parenthesis

library\_function : identifier ;

Parenthesis conditional\_statement

Parenthesis

library\_function : identifier ;

library\_function : identifier ;

Parenthesis

Parenthesis

user\_function identifier Parenthesis Parenthesis Parenthesis

Loop

Parenthesis

library\_function : Literal;

Parenthesis

Parenthesis

Number of tokens:

Identfiers = 18

Statements = 9

User function = 1

Datatype = 3

Loop = 1

Conditional statement = 2

Keywords = 1

**ID AND TYPE DESIGN:**

Identifier: [a-zA-Z\_][0-9a-zA-Z\_]\*

Data types: there are 3 data types.

Purno -> integer

Vogno -> float

String -> string

**OPERATOR HANDLING:**

Logical Operator:

and -> &&

or -> ||

not -> !

Relational Operator:

< | <= | > | >= | != | ==

Assignment Operator:

= | += | -= | \*= | /=

Arithmetic Operator:

+ | - | \* | / | ^

**LOOP AND DIRECTIVES:**

Loop format : "loop "[0-9]+".."[0-9]+

Example: loop 1..10

It will iterate from 1 to 10 based on conditions.

**Discussion:**

In this experiment, we successfully designed and implemented a unique programming language using Flex. During the implementation of the tokenization program in Flex, several challenges were encountered. One of the main challenges was defining regular expressions that correctly match and distinguish between different token types. These skills will be valuable for future compiler development projects. Overall, it was a successful lab.

**Conclusion:**

In conclusion, our experiment demonstrated the feasibility of creating a new programming language using Flex. We achieved the objective of implementing a tokenization program in Flex to break down a source code file into meaningful tokens. Tokenization is a critical step in the compilation process, and our program successfully recognized keywords, identifiers, operators, literals, and other language-specific elements. We also addressed challenges such as operator handling, comments, and whitespace to ensure accurate tokenization.