**Assignment No 01**

**Write a program using LEX specifications to implement lexical analysis phase of compiler to generate tokens of subset of ‘C’ program**

**assignmentOne.l**

%{

#include<stdio.h>

%}

%%

[/][/].\* {printf("\n Single line comment1 : %s",yytext);}

"/\*"[^\*]\*"\*/" {printf("\n Multi line comment : %s",yytext);}

# {printf("\n Processing Directives : %s",yytext);}

include|printf|int|void|main {printf("\n Keywords : %s",yytext);}

"<"|">"|"("|")"|";"|","|"{"|"}" {printf("\n Punctuation : %s",yytext);}

[a-z]+[.][h] {printf("\n Header files : %s",yytext);}

["].\*["] {printf("\n Litrels : %s",yytext);}

[a-zA-Z][a-zA-Z0-9\_] {printf("\n Identifier : %s",yytext);}

[0-9]+ {printf("\n Integer Number : %s",yytext);}

[0-9]+(\.[0-9]+) {printf("\n Decimal Number : %s",yytext);}

"+"|"-"|"=" {printf("\n Operators : %s",yytext);}

%%

int yywrap()

{

return 1;

}

int main()

{

yyin=fopen("pro.c","r");

yylex();

return 0;

}

**pro.c**

#include<stdio.h>

void fun(){

printf("Hello this is user defined function");

}

int main(){

//variable

int rno=77;

float marks=8.88;

printf("Akshada Phopse");

fun();

return 0;

/\*hello this

is multiline comment\*/

}

**Commands to Run Program**

1. lex assignmentOne.l
2. gcc lex.yy.c
3. ./a.out

**Assignment No 02**

**Write a LEX program to display word, character and line counts for a sample input text file**

**assignmentTwo.l**

%{

#include <stdio.h>

int wc = 0, lc = 0, cc = 0, dc = 0, vc = 0;

%}

%%

[aeiouAEIOU] { vc++; cc++; wc++;}

[0-9] { dc++; cc++; }

\n { lc++; cc++; }

[ \t]+ { cc += yyleng; }

[^ \t\n]+ { wc++; cc += yyleng; }

%%

int yywrap()

{

return 1;

}

int main()

{

yyin = fopen("atwo.txt", "r");

yylex(); // Perform lexical analysis

printf("Number of Lines : %d\n", lc);

printf("Number of Words : %d\n", wc);

printf("Number of Characters : %d\n", cc);

printf("Number of Digits : %d\n", dc);

printf("Number of Vowels : %d\n", vc);

return 0;

}

**atwo.txt**

This is 2 nd assignment a e i o u

**Commands to Run Program**

1. lex assignmentTwo.l
2. gcc lex.yy.c
3. ./a.out

**Assignment No 03**

**Write a program using YACC specifications to implement syntax analysis phase of compiler to validate type and syntax of variable declaration in C program.**

**assignThree.l**

%{

#include "y.tab.h"

%}

%%

"int" { return INT; }

"float" { return FLOAT; }

"char" { return CHAR; }

[a-zA-Z\_][a-zA-Z0-9\_]\* { return ID; }

"," { return COMMA; }

";" { return SEMICOLON; }

[ \t\n] { /\* Ignore whitespace \*/ }

. { return yytext[0]; }

%%

int yywrap() {

return 1;

}

**assignThree.y**

%{

#include <stdio.h>

#include <stdlib.h>

int yyerror(char \*str);

int yywrap();

%}

%token INT FLOAT CHAR ID COMMA SEMICOLON

%%

Stmt: Type VarList SEMICOLON { printf("Valid Declaration\n"); }

| error SEMICOLON { printf("Invalid Declaration\n"); }

;

Type: INT | FLOAT | CHAR;

VarList: ID

| ID COMMA VarList;

%%

int yyerror(char \*str) {

printf("Syntax Error: %s\n", str);

return 0;

}

int main() {

printf("Enter a variable declaration:\n");

yyparse();

return 0;

}**Commands to Run Program**

1. lex assignThree.l
2. yacc –d assignThree.y
3. gcc lex.yy.c y.tab.c
4. ./a.out

**Assignment No 04**

**Write a program using YACC specifications to implement calulator to perform various arithmetic operations**

**assignFour.l**

%{

#include "y.tab.h"

%}

%%

[0-9] { yylval = atoi(yytext); return N; }

[ \t ]

"\n" { return 0;}

. {return yytext[0]; }

%%

int yywrap() {

return 1;

}

**assignFour.y**

%{

#include <stdio.h>

#include <stdlib.h>

int yyerror(char \*str);

int yywrap();

%}

%token N

%left '+' '-'

%left '\*' '/' '%'

%left '(' ')'

%%

A: E { printf("Result is = %d\n", $$); };

E: E '+' E { $$ = $1 + $3; }

| E '-' E { $$ = $1 - $3; }

| E '\*' E { $$ = $1 \* $3; }

| E '/' E { $$ = $1 / $3; }

| E '%' E { $$ = $1 % $3; }

| '(' E ')' { $$ = $2; }

| N { $$ = $1; }

%%

int yyerror(char \*str) {

printf("Invalid Expression\n");

return 0;

}

int main() {

printf("Enter Arithmetic Expression: ");

yyparse();

return 0;

}

**Commands to Run Program**

1. lex assignFour.l
2. yacc –d assignFour.y
3. gcc lex.yy.c y.tab.c
4. ./a.out

**Write a program using YACC specifications to implement syntax analysis phase of compiler to recognize simple and compound sentences given in input file.**

Ass5.l

%{

#include "y.tab.h"

#include <string.h>

#include <stdlib.h>

char\* strdup(const char\* str);

%}

%%

he|she|they|we|i|john|mary { yylval.str = strdup(yytext); return SUBJECT; }

eats|likes|loves|hates|knows|sees { yylval.str = strdup(yytext); return VERB; }

apple|banana|pizza|book|movie|car { yylval.str = strdup(yytext); return OBJECT; }

and|but|or|so { yylval.str = strdup(yytext); return CONJUNCTION; }

[ \t\n]+ ;

. { printf("Unknown character: %s\n", yytext); }

%%

int yywrap(void) {

return 1;

}

Ass5.y

%{

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

void yyerror(const char \*s);

int yylex(void);

%}

%union {

char\* str;

}

%token <str> SUBJECT VERB OBJECT CONJUNCTION

%type <str> simple\_sentence compound\_sentence

%start sentence

%%

sentence:

simple\_sentence { printf("Valid simple sentence\n"); free($1); }

| compound\_sentence { printf("Valid compound sentence\n"); free($1); }

;

simple\_sentence:

SUBJECT VERB OBJECT {

printf("Simple sentence: %s %s %s\n", $1, $2, $3);

$$ = strdup("simple\_sentence");

free($1); free($2); free($3);

}

;

compound\_sentence:

simple\_sentence CONJUNCTION simple\_sentence

{

printf("Compound sentence with conjunction '%s'\n", $2);

$$ = strdup("compound\_sentence");

free($2);

}

;

%%

int main() {

printf("Enter a sentence:\n");

yyparse(); // Start the parser

return 0;

}

void yyerror(const char \*s) {

fprintf(stderr, "Error: %s\n", s);

}

Sentence.txt  
he eats apple

mary loves pizza

john sees car

she likes book and he eats banana

we hate movie but they love car

i know book or she sees movie

they love pizza so we eat apple

john likes banana and mary hates book

they loves pizza so we eats apple

**Write a program to implement recursive descent parser(RDP) for sample language.**

#include <iostream>

#include <cctype>

#include <string>

using namespace std;

string input; // Input string

int index = 0; // Pointer to the current character in the input

// Function prototypes

bool parseE();

bool parseEPrime();

bool parseT();

bool parseTPrime();

bool parseF();

int main() {

cout << "Enter an expression: ";

getline(cin, input); // Read the entire line as input

// Start parsing from the E (Expression) non-terminal

if (parseE() && index == input.length()) {

cout << "Sentence successfully parsed." << endl;

} else {

cout << "Sentence failed to parse." << endl;

}

return 0;

}

// E -> T E'

bool parseE() {

if (parseT()) {

return parseEPrime();

}

return false;

}

// E' -> + T E' | ε

bool parseEPrime() {

if (index < input.length() && input[index] == '+') {

index++; // consume '+'

if (parseT()) {

return parseEPrime();

}

return false;

}

return true; // epsilon (empty production)

}

// T -> F T'

bool parseT() {

if (parseF()) {

return parseTPrime();

}

return false;

}

// T' -> \* F T' | ε

bool parseTPrime() {

if (index < input.length() && input[index] == '\*') {

index++; // consume '\*'

if (parseF()) {

return parseTPrime();

}

return false;

}

return true; // epsilon (empty production)

}

// F -> ( E ) | id

bool parseF() {

if (index < input.length() && input[index] == '(') {

index++; // consume '('

if (parseE()) {

if (index < input.length() && input[index] == ')') {

index++; // consume ')'

return true;

}

}

return false;

} else if (index < input.length() && isalpha(input[index])) {

// Match an identifier (id)

index++; // consume the identifier

return true;

}

return false;

}

**Write a program using LEX and YACC to generate a symbol table**

*//Symbol.java*

*import* java**.**util**.***\****;**

*import* java**.**io**.***\****;**

class **Symbol** {

*int* entryNo**;**

    String symbol**;**

*int* address**;**

*int* length**;**

    Symbol(*int* **entryNo,** String **symbol,** *int* **address,** *int* **length**) {

*this***.***entryNo* **=** entryNo**;**

*this***.***symbol* **=** symbol**;**

*this***.***address* **=** address**;**

*this***.***length* **=** length**;**

    }

}

*public* class **symbol** {

*public* *static* *void* main(String[] **args**) {

        String fileName **=** "program.txt"**;**

        List**<**Symbol**>** symbolTable **=** **new** ArrayList**<>**()**;**

*int* locationCounter **=** 0**;**

*int* entryNo **=** 1**;**

**try** (BufferedReader br **=** **new** BufferedReader(**new** FileReader(fileName))) {

            String line**;**

**while** ((line **=** br**.**readLine()) **!=** null) {

                line **=** line**.**trim()**;**

**if** (line**.**isEmpty()) **continue;**

**if** (line**.**equalsIgnoreCase("END")) {

**continue;**

                }

                String[] parts **=** line**.**split("\\s+")**;**

**if** (parts[0]**.**equalsIgnoreCase("START")) {

                    locationCounter **=** Integer**.**parseInt(parts[1])**;**

**continue;**

                }

                String label **=** null**;**

                String instruction **=** null**;**

*int* length **=** 1**;**

**if** (parts**.***length* **==** 1) {

                    label **=** parts[0]**;**

                } **else** **if** (parts**.***length* **==** 2) {

**if** (**!**isOpcode(parts[0])) {

                        label **=** parts[0]**;**

                        instruction **=** parts[1]**;**

                    } **else** {

                        instruction **=** parts[0]**;**

                    }

                } **else** **if** (parts**.***length* **>=** 3) {

                    label **=** parts[0]**;**

                    instruction **=** parts[1]**;**

                }

*// Check if the label is not an opcode before adding it to the symbol table*

**if** (label **!=** null **&&** **!**isOpcode(label)) {

**if** ("DS"**.**equalsIgnoreCase(instruction)) {

                        length **=** Integer**.**parseInt(parts[2])**;**

                    }

                    symbolTable**.**add(**new** Symbol(entryNo**++,** label**,** locationCounter**,** length))**;**

                    locationCounter **+=** length**;**

                } **else** {

                    locationCounter**++;**

                }

            }

*// Print symbol table*

            System**.***out***.**printf("%-10s %-10s %-10s %-10s%n"**,** "Entry No."**,** "Symbol"**,** "Address"**,** "Length")**;**

            System**.***out***.**println("--------------------------------------------")**;**

**for** (Symbol sym **:** symbolTable) {

                System**.***out***.**printf("%-10d %-10s %-10d %-10d%n"**,** sym**.***entryNo***,** sym**.***symbol***,** sym**.***address***,** sym**.***length*)**;**

            }

        } **catch** (IOException **e**) {

            System**.***out***.**println("Error reading file: " **+** e**.**getMessage())**;**

        }

    }

*// This method checks whether the word is a valid opcode*

*static* *boolean* isOpcode(String **word**) {

        String[] opcodes **=** {"MOVER"**,** "MOVEM"**,** "ADD"**,** "SUB"**,** "MULT"**,** "DIV"**,** "READ"**,** "PRINT"}**;**

**for** (String opcode **:** opcodes) {

**if** (opcode**.**equalsIgnoreCase(word)) {

**return** true**;**

            }

        }

**return** false**;**

    }

}

Program.txt

START 0

LOOP MOVER AREG, A

ADD B

MOVEM AREG, C

END

A DS 1

B DS 1

C DS 1