**Ex.No:1** **Develop a lexical analyzer to recognize a few patterns in C. (Ex.identifiers, constants, comments, operators etc.).**

**PROGRAM:**

%{

#include <stdio.h>

#include <ctype.h>

#include <string.h>

int identifiers = 0, constants = 0, operators = 0, comments = 0;

%}

%option noyywrap

%%

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

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

"/\*"([^\*]|[\r\n]|"\*"[^/])\*"\*/" { printf("Comment detected\n"); comments++; }

"+"|"-"|"\*"|"/" { printf("Operator: %s\n", yytext); operators++; }

\n { /\* Ignore newlines \*/ }

[ \t] { /\* Ignore spaces \*/ }

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

%%

int main() {

printf("Enter C code:\n");

yylex();

printf("\nSummary:\nIdentifiers: %d\nConstants: %d\nOperators: %d\nComments: %d\n", identifiers, constants, operators, comments);

return 0;

}

**OUTPUT:**

Identifier: int

Identifier: a

Constant: 5

Comment detected

Identifier: b

Identifier: a

Operator: +

Constant: 10

Summary:

Identifiers: 5

Constants: 2

Operators: 1

Comments: 1

**Ex.No:2** **Implement a Lexical Analyzer using Lex Tool**

**PROGRAM:**

%{

int COMMENT=0;

%}

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

%%

#.\* {printf("\n %s is a preprocessor directive",yytext);}

int |

float |

void |

main |

if |

else |

printf |

scanf |

for |

char |

getch |

while {printf("\n %s is a keyword",yytext);}

"/\*" {COMMENT=1;}

"\*/" {COMMENT=0;}

{identifier}\( {if(!COMMENT)printf("\n Function:\t %s",yytext);}

\{ {if(!COMMENT)printf("\n Block begins");}

\} {if(!COMMENT)printf("\n Block ends");}

{identifier}(\[[0-9]\*\])? {if(!COMMENT)printf("\n %s is an Identifier",yytext);}

\".\*\" {if(!COMMENT)printf("\n %s is a string",yytext);}

[0-9]+ {if(!COMMENT)printf("\n %s is a number",yytext);}

\)(\;)? {if(!COMMENT)printf("\t");ECHO;printf("\n");}

\(ECHO;

= {if(!COMMENT)printf("\n %s is an Assignment operator",yytext);}

\<= |

\>= |

\< |

== {if(!COMMENT)printf("\n %s is a relational operator",yytext);}

.|\n

%%

int main(int argc, char \*\*argv)

{

if(argc>1)

{

FILE\*file;

file=fopen(argv[1],"r");

if(!file)

{

printf("\n Could not open the file:%s ",argv[1]);

exit(0);

}

yyin=file;

}

yylex();

printf("\n\n");

return 0;

}

int yywrap()

{

return 0;

}

**OUTPUT:**

is a preprocessor directive

void is a keyword

Function: main( )

Block begins

int is a keyword

a is an Identifier

= is an Assignment operator

10 is a number

int is a keyword

array[10] is an Identifier

**Ex.No:3a Program to recognize a valid arithmetic expression that uses operator + , -, \* and /.**

**PROGRAM:**

**LEX PART: ex3a.l**

%{

#include "ex3a.tab.h"

%}

%%

"=" {printf("\n Operator is EQUAL");}

"+" {printf("\n Operator is PLUS");}

"-" {printf("\n Operator is MINUS");}

"/" {printf("\n Operator is DIVISION");}

"\*" {printf("\n Operator is MULTIPLICATION");}

[a-zA-Z]\*[0-9]\* {printf("\n Identifier is %s",yytext);return ID;}

. return yytext[0];

\n return 0;

%%

int yywrap()

{

return 1;

}

**YACC PART: ex3a.y**

%{

#include<stdio.h>

%}

%token A

%token ID

%%

statement:A'='E

| E{

printf("\n Valid arithmetic expression");

$$=$1;

};

E:E'+'ID

|E'-'ID

|E'\*'ID

|E'/'ID

|ID

;

%%

extern FILE \*yyin;

main()

{

do

{

yyparse();

}while(!feof(yyin));

}

yyerror(char\*s)

{

}

**Output:**

[root@localhost]# lex ex3a.1

[root@localhost]# yacc –d ex3a.y

[root@localhost]# gcc lex.yy.c ex3a.tab.c

[root@localhost]# ./a.out

x=a+b;

Identifier is x

Operator is EQUAL

Identifier is a

Operator is PLUS

Identifier is b

**Ex.No:3b Program to recognize a valid variable which starts with a letter followed by any number of letters or digits.**

**PROGRAM:**

%{

#include <stdio.h>

%}

%%

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

[0-9]+ { printf("Invalid variable (starts with digit): %s\n", yytext); }

\n { return 0; }

. { printf("Invalid token: %s\n", yytext); }

%%

int main() {

printf("Enter variable names (CTRL+D to stop):\n");

yylex();

return 0;

}

**OUTPUT:**

Enter variable names (CTRL+D to stop):

\_hello

Valid variable: \_hello

Enter variable names (CTRL+D to stop):

\_hello

Valid variable: \_hello

Enter variable names (CTRL+D to stop):

1hello

Invalid variable (starts with digit): 1

Valid variable: hello

**Ex.No:3c** **Program to recognize a valid control structures syntax of c language(For loop,**

**While loop, if-else, if-else-if, switch case, etc.,**

**PROGRAM:**

%{

#include <stdio.h>

%}

%%

"for" { printf("Control structure: for\n"); }

"while" { printf("Control structure: while\n"); }

"if" { printf("Control structure: if\n"); }

"else" { printf("Control structure: else\n"); }

"switch" { printf("Control structure: switch\n"); }

"case" { printf("Control structure: case\n"); }

"default" { printf("Control structure: default\n"); }

\n { return 0; }

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

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

%%

int main() {

printf("Enter control structures:\n");

yylex();

return 0;

}

**OUTPUT:**

if (x > 0) {

while (y < 10) {

for (int i = 0; i < 5; i++) {

// loop

}

}

} else {

switch (z) {

case 1: break;

default: break;

}

}

Control structure: if

Control structure: while

Control structure: for

Control structure: else

Control structure: switch

Control structure: case

Control structure: default

**Ex.No:3d Implement an Arithmetic Calculator using LEX and YACC**

**PROGRAM:**

**LEX PART: ex3d.l**

%{

/\* Definition section \*/

#include<stdlib.h>

#include "y.tab.h"

extern int yylval;

%}

/\* Rule Section \*/

%%

[0-9]+ {

yylval=atoi(yytext);

return NUMBER;

}

'<=' return LE;

'>=' return GE;

'!=' return NE;

'==' return EQ;

[\t] ;

[\n] return 0;

. return yytext[0];

%%

**YACC PART:ex3d.y**

%{

/\* Definition section \*/

#include<stdio.h>

int flag=0;

%}

%token NAME NUMBER

%left GE LE EQ NE EE '<' '>'

%left '+' '-'

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

%left '(' ')'

%nonassoc UMINUS

/\* Rule Section \*/

%%

ArithmeticExpression: E{

printf("Result=%d", $$);

return 0;

};

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

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

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

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

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

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

| NUMBER {$$=$1;}

|E GE E {$$=$1 >= $3 ;}

|E LE E {$$=$1 <= $3 ;}

|E NE E {$$=$1 != $3 ;}

|E EE E {$$=$1 == $3 ;}

|UMINUS E {$$=-$1 ;}

;

%%

//driver code

int main()

{

//printf("\nEnter the Expression:\n");

yyparse();

//if(flag==0)

//printf("\nEntered arithmetic expression is Valid\n\n");

// return 0;

}

int yyerror()

{

// printf("\nEntered arithmetic expression is Invalid\n\n");

// flag=1;

}

int yywrap(){

return 1;

}

**OUTPUT:**

6\*(2+3)

Result: 30

**Ex.No: 4 Generate three address code for a simple program using LEX and YACC**

**PROGRAM:**

**LEX PART:ex4.l**

%{

#include "y.tab.h"

%}

%%

"print" {return print;}

"True" {return \_true\_;}

"False" {return \_false\_;}

"<" {return lt;}

">" {return gt;}

"==" {return eq;}

"<=" {return lteq;}

">=" {return gteq;}

"!=" {return nteq;}

[a-zA-Z] {yylval.id = yytext[0]; return identifier;}

[0-9]+ {yylval.num = atoi(yytext); return number;}

[ \t\n] ;

[-+=;\*/(){}%] {return yytext[0];}

. {ECHO; yyerror ("unexpected character"); exit(1);}

%%

int yywrap (void)

{

return 1;

}

**YACC PART:ex4.b**

%{

#include<stdio.h>

#include<stdlib.h>

int temp\_count=0;

void yyerror(const char\*s){

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

}

%}

%token NUM EOL

%left '+' '-'

%left '\*' '/'

%%

program:lines

;

lines:lines line

| line

;

line:expr EOL

{

printf("Result:t%d\n",$1);

}

;

expr:NUM{

$$=$1;

}

| '(' expr ')'

{

$$=$2;

}

| expr '+' expr

{

printf("t%d=%d+%d\n",++temp\_count,$1,$3);

$$=temp\_count;

}

| expr '-' expr

{

printf("t%d=%d-%d\n",++temp\_count,$1,$3);

$$=temp\_count;

}

| expr '\*' expr

{

printf("t%d=%d\*%d\n",++temp\_count,$1,$3);

$$=temp\_count;

}

| expr '/' expr

{

if($3==0)

{yyerror("Division by zero");

$$=0;}

else{

printf("t%d=%d/%d\n",++temp\_count,$1,$3);

$$=temp\_count;

}

}

;

%%

int main()

{

yyparse();

return 0;

}

**OUTPUT:**

Input: x = 5 \* 2 + 3;

t1 = 5 \* 2

t2 = t1 + 3

x = t2

**Ex.No:5 Implement type checking using Lex and Yacc.**

**PROGRAM:**

**LEX PART:ex5.l**

%{

#include "y.tab.h"

%}

%%

[0-9]+ {

yylval = atoi(yytext);

return INTEGER;

}

[0-9]+"."[0-9]\* {

yylval = atof(yytext);

return FLOAT;

}

[a-zA-Z]+ {

yylval= yytext;

return CHAR;

}

[ \t] ; // Ignore whitespace and tabs

\n { return EOL; } // Newline character

. { return yytext[0]; } // Return other characters as is

%%

int yywrap() {

return 1;

}

**YACC PART:ex5.y**

%{

#include <stdio.h>

void yyerror(const char\* s) {

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

}

int yylex(); // Declare the lexer function

%}

%token INTEGER FLOAT CHAR EOL

%%

program:

/\* empty \*/

| program line

;

line:

statement EOL {

if ($1 == INTEGER) {

printf("Type: INTEGER\n");

} else if ($1 == FLOAT) {

printf("Type: FLOAT\n");

}

else if ($1 == CHAR) {

printf("Type: CHAR/STRING\n");

} else {

printf("Invalid type\n");

}

}

;

statement:

expression {

$$ = $1;

}

;

expression:

INTEGER {

$$ = INTEGER;

}

| FLOAT {

$$ = FLOAT;

}

| CHAR {

$$ = CHAR;

}

;

%%

int main() {

yyparse();

return 0;

}

**OUTPUT:**

123

Type:INTEGER

123.897

Type:FLOAT

God

Type:CHAR/STRING

df24

Invalid type or Parse error

**Ex.No:6 Implement simple code optimization techniques (Constant folding, Strength**

**reduction and Algebraic transformation)**

**PROGRAM:**

#include<stdio.h>

#include<string.h>

struct op

{

char l;

char r[20];

}

op[10],pr[10];

void main()

{

int a,i,k,j,n,z=0,m,q;

char \*p,\*l;

char temp,t;

char \*tem;

printf("Enter the Number of Values:");

scanf("%d",&n);

for(i=0;i<n;i++)

{

printf("left: ");

scanf(" %c",&op[i].l);

printf("right: ");

scanf(" %s",&op[i].r);

}

printf("Intermediate Code \n") ;

for(i=0;i<n;i++)

{

printf("%c=",op[i].l);

printf("%s \n",op[i].r);

}

for(i=0;i<n-1;i++)

{

temp=op[i].l;

for(j=0;j<n;j++)

{

p=strchr(op[j].r,temp);

if(p)

{

pr[z].l=op[i].l;

strcpy(pr[z].r,op[i].

r);

z++;

}

}

}

pr[z].l=op[n-1].l;

strcpy(pr[z].r,op[n-1].r);

z++;

printf(" After Dead Code Elimination\n");

for(k=0;k<z;k++)

{

printf("%c =",pr[k].l);

printf("%s \n",pr[k].r);

}

for(m=0;m<z;m++)

{

tem=pr[m].r;

for(j=m+1;j<z;j++)

{

p=strstr(tem,pr[j].r);

if(p)

{

t=pr[j].l;

pr[j].l=pr[m].l;

for(i=0;i<z;i++)

{

l=strchr(pr[i].r,t) ;

if(l)

{

a=l-pr[i].r;

printf("pos: %d\n",a);

pr[i].r[a]=pr[m].l;

}}}}}

printf("Eliminate Common Expression\n");

for(i=0;i<z;i++)

{

printf("%c =",pr[i].l);

printf("%s\n",pr[i].r);

}

for(i=0;i<z;i++)

{

for(j=i+1;j<z;j++)

{

q=strcmp(pr[i].r,pr[j].r);

if((pr[i].l==pr[j].l)&&!q)

{

pr[i].l="";

}

}

}

printf("Optimized Code\n");

for(i=0;i<z;i++)

{

if(pr[i].l!="")

{

printf("%c=",pr[i].l);

printf("%s\n",pr[i].r);

}

}

}

**OUTPUT:**

Enter the Number of Values:5

left: a

right: 9

left: b

right: c+d

left: e

right: c+d

left: f

right: b+e

left: r

right: :f

Intermediate Code

a=9

b=c+d

e=c+d

f=b+e

r=:

After Dead Code Elimination

b =c+d

e =c+d

r =:f

Eliminate Common Expression

b =c+d

b =c+d

r =:

Optimized Code

a=c+d

b=c+d

r=:f

**Ex.No:7 Implement back-end of the compiler for which the three address code is given as input and the 8086 assembly language code is produced as output.**

**PROGRAM:**

#include <stdio.h>

#include <stdio.h>

#include <string.h>

void main() {

char icode[10][30], str[20], opr[10];

int i = 0;

printf(" Enter the set of intermediate code (terminated by exit):");

do

{

scanf("%s", icode[i]);

} while (strcmp(icode[i++], "exit") != 0);

printf(" target code generation");

printf(" \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

i = 0;

do {

strcpy(str, icode[i]);

switch (str[3]) {

case '+':

strcpy(opr, "ADD ");

break;

case '-':

strcpy(opr, "SUB ");

break;

case '\*':

strcpy(opr, "MUL ");

break;

case '/':

strcpy(opr, "DIV ");

break;

}

printf(" Mov %c,R%d\n", str[2], i);

printf(" %s%c,R%d\n", opr, str[4], i);

printf(" Mov R%d,%c\n", i, str[0]);

}

while (strcmp(icode[++i], "exit") != 0);

}

**OUTPUT:**

Enter the set of intermediate code (terminated by exit):

a=a\*b

c=f\*h

g=a\*h

f=Q+W

t=q-j

exit

target code generation \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Mov a,R0

MUL b,R0

Mov R0,a

Mov f,R1

MUL h,R1

Mov R1,c

Mov a,R2

MUL h,R2

Mov R2,g

Mov Q,R3

ADD W,R3

Mov R3,f

Mov q,R4

SUB j,R4

Mov R4,t