**COMPILER DESIGN LAB**

1. Implement Token Separation for a given expression.

2.a. Use LEX and YACC to implement parser on ambiguous grammar.

2.b. Use LEX and YACC to implement parser on unambiguous grammar using LEX.

3. Use LEX and YACC tool to implement Desktop Calculator.

4. Implement Operator precedence parsing.

5. Implement Recursive Descent Parser algorithm.

6. Implement Shift Reduce Parser algorithm.   
7. Implement the back end of the compiler to produce three address code.

8. Implement Symbol Table management using LEX and YACC.

9. Construct NFA and DFA from a regular expression and verify weather the given string accepted or not using JFLAP.

10. Construct NFA and DFA from a regular expression and generate the optimistic regular expression using JFLAP.

**1. Implement Token Separation for a given expression using LEX.**

File.l :

letter [A-Za-z]

digit [0-9]

operator [-+\*]

%%

void |

main |

if |

do |

printf |

int |

float |

char |

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

%s |

%c |

%d |

%f {printf("%s is a Format Specifier\n",yytext);}

({letter}|\_)({letter}|{digit})\* {printf("%s is a identifier\n",yytext);}

{operator}({operator})\* {printf("%s is an operator\n",yytext);}

{digit}+ {printf("%s is a number\n",yytext);}

"(" {printf("%s is an open parenthesis\n",yytext);}

\) {printf("%s is an close parenthesis\n",yytext);}

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

\{

\.

\=

\/

\,

\'

\" {printf("%s is a double quote\n",yytext);}

. {printf("\nSyntax Error!!\n");}

%%

int main(intargc, char \*argv[])

{

FILE \*fp=fopen(argv[1],"r");

printf("\n%s %s %s\n",argv[1],argv[2],argv[3]);

yyin=fp;

yylex();

return 0;

}

Input.c

int x;

charch;

void main()

{

getch();

}

Output :

**2.a. Use LEX and YACC to implement parser on ambiguous grammar**.

File.l :

%{

#include<stdlib.h>

#include"y.tab.h"

voidyyerror(char \*s);

externintyylval;

%}

%%

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

[-+\*/\n] {return \*yytext;}

"(" {return \*yytext;}

\) {return \*yytext;}

[\t];

. {yyerror("Syntax Error");}

%%

intyywrap()

{

return 1;

}

File.y :

%{

#include<stdio.h>

externintyylex(void);

voidyyerror(char \*);

%}

%token INT

%%

program:

programexpr '\n { printf("%d\n",$2);}

|

;

expr:

expr '+' expr {$$=$1+$3;}

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

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

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

|INT {$$=$1;}

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

;

%%

voidyyerror(char \*s)

{

printf("%s\n",s);

}

int main()

{

yyparse();

return 0;

}

INPUT:

OUTPUT:

**2.b. Use LEX and YACC to implement parser on unambiguous grammar.**

File.l :

%{

#include<stdlib.h>

#include"y.tab.h"

voidyyerror(char \*s);

externintyylval;

%}

%%

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

[-+\*/\n] {return \*yytext;}

"(" {return \*yytext;}

\) {return \*yytext;}

[\t];

. {yyerror("Syntax Error");}

%%

intyywrap()

{

return 1;

}

File.y :

%{

#include<stdio.h>

externintyylex(void);

voidyyerror(char \*);

%}

%token INT

%%

program:

programexpr '\n' { printf("%d\n",$2);}

|

;

expr:

T {$$=$1;}

|expr '+' T {$$=$1+$3;}

|expr '-' T {$$=$1-$3;}

;

T:

F {$$=$1;}

|T '\*' F {$$=$1\*$3;}

|T '/' F {$$=$1/$3;}

;

F:

INT {$$=$1;}

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

;

%%

voidyyerror(char \*s)

{

printf("%s\n",s);

}

int main()

{

yyparse();

return 0;

}

INPUT:

OUTPUT:

**3. Use LEX and YACC tool to implement Desktop Calculator.**

File.l

%{

#include<stdlib.h>

#include"y.tab.h"

voidyyerror(char \*s);

externintyylval;

%}

%%

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

[a-z] {yylval=toascii(\*yytext)-97; return ID;}

[A-Z] {yylval=toascii(\*yytext)-65; return ID;}

[-+\*=/\n] {return \*yytext;}

\( {return \*yytext;}

")" {return \*yytext;}

[\t] ;

. {yyerror("Invalid Token!!");}

%%

intyywrap()

{

return 1;

}

File.y

%{

#include<stdio.h>

externintyylex(void);

voidyyerror(char \*);

int x=0;

intval[26];

%}

%token INT ID

%%

nithish:

nithishexpr '\n' {x=$2; printf("%d\n",$2);}

|nithish ID '=' expr '\n' {val[$2]=$4;}

|

;

expr:

expr '+' T {$$=$1+$3;}

|expr '-' T {$$=$1-$3;}

|T {$$=$1;}

|'+' T {$$=x+$2;}

|'-' T {$$=x-$2;}

;

T:

F {$$=$1;}

|T '\*' F {$$=$1\*$3;}

|T '/' F {$$=$1/$3;}

|'\*' F {$$=x\*$2;}

|'/' F {$$=x/$2;}

;

F:

INT {$$=$1;}

|ID {$$=val[$1];}

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

;

%%

voidyyerror(char\* s)

{

printf("%s",s);

}

int main()

{

yyparse();

return 0;

}

Input

Output

**4. Implement Operator precedence parsing**

#include<stdio.h>

#include<conio.h>

#include<string.h>

#include<ctype.h>

#include<stdlib.h>

charbuf[50],stk[50]="$",choice;

intibuf,istk=0,i,j;

char q[9][9]={

//{'+','-','\*','/','^','a','(',')','$'}

{'>','>','<','<','<','<','<','>','>'},//+

{'>','>','<','<','<','<','<','>','>'},//-

{'>','>','>','>','<','<','<','>','>'},//\*

{'>','>','>','>','<','<','<','>','>'},// \

{'>','>','>','>','<','>','<','>','>'},//^

{'<','<','<','<','<','E','E','>','>'},// a

{'<','<','<','<','<','<','<','=','E'},//(

{'>','>','>','>','>','E','E','>','>'},//)

{'<','<','<','<','<','<','<','E','A'}, //$

};

char c[9]={'+','-','\*','/','^','a','(',')','$'};

intcheckindex(char ch)

{

if(isalpha(ch)) ch='a';

for(i=0;i<9;i++)

if(ch==c[i]) // q[\*][/]

returni; // 2 3

printf("\nInvalid Token");

getch();

exit(0);

}

main()

{

clrscr();

printf("Enter a string:");

scanf("%s",buf);

printf("\n + - \* / ^ a ( ) $\n");

for(i=0;i<9;i++)

{

printf("\n%c ",c[i]);

for(j=0;j<9;j++)

{

printf(" %c",q[i][j]);

}

}

printf("\n\nPARSING TABLE:");

printf("\nSTACK\tBUFFER\tACTION");

strcat(buf,"$");

strrev(buf);

ibuf=strlen(buf)-1;

while(1)

{

choice=q[checkindex(stk[istk])][checkindex(buf[ibuf])];

printf("\n%s\t %s\t %c\t ",stk,buf,choice);

getch();

switch(choice)

{

case '<':

printf(" PUSH");

istk++; stk[istk]=buf[ibuf];

stk[istk+1]='\0';

buf[ibuf]='\0'; ibuf--;

break;

case '>':

printf(" POP");

stk[istk]='\0'; istk--;

break;

case '=':

printf("POP BOTH");

stk[istk]='\0'; istk--;

buf[ibuf]='\0'; ibuf--;

break;

case 'A':

printf(" Accepted");

getch();

exit(0);

default:

printf(" ERROR");

getch();

exit(0);

}

}

getch();

}

Input:

Output: