**Compiler Design Lab (TCS-610)**

**Command for installation of Lex and Yacc tools in Ubuntu** (when connected to internet)

## $sudo apt -get install flex

**Program1:** Lex code for counting number of lines, spaces, tabs and rest of characters.

**int n,m,t,c;**

**%%**

**\n n++;**

**\t m++; [ ] t++;**

**. c++;**

**%%**

**main()**

**{ yylex();**

**printf("Total number of\nlines=%d \ntabs=%d\nspaces=%d\nchars=%d \n",n,m,t,c); }**

/\* To run the code $ lex count.l

$ gcc lex.yy.c -lfl

$ ./a.out

mai ghar jana chahta hu.

//Press <Enter> then <CTRL><d> to stop giving input.

Total number of lines=2

tabs=1 spaces=3 char=19 \*/

**Program2:** Design a LEX Code to identify and print valid Identifier of C/C++ in given Input pattern.

**%{**

**int c=0;**

**%}**

**%%**

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

**. ;**

**%%**

**main()**

**{**

**yylex();**

**printf("\nTotal number of valid Identifier = %d \n",c);**

**}**

/\*

$ lex valid\_Identifier.l

$ cc lex.yy.c -lfl

$ ./a.out

count ad\_samsung w12

valid Identifier = count valid Identifier = ad\_samsung valid Identifier = w12 123 3\_er gh\_

valid Identifier = gh\_

//Press <Enter> then <CTRL><d> to stop giving input.

Total number of valid Identifier = 4

$

\*/

**Program3:** Design a LEX Code to identify and print integer and float value in given Input pattern.

**%{**

**int m=0,n=0;**

**%}**

**%%**

**-?[0-9]+ {m++; printf("\t Integer = %s",yytext);}**

**-?[0-9]+"."[0-9]+ {n++; printf("\t Float = %s",yytext);}**

**. ;**

**%%**

**main()**

**{**

**yylex();**

**printf("\nTotal number of Integer = %d & Float = %d \n",m,n);**

**}**

/\*

$ lex int\_float.l

$ cc lex.yy.c -lfl

$ ./a.out

123 4.9 acd 2567 .32

Integer = 123 Float = 4.9 Integer = 2567 Float = .32

2 3 3\_er gh\_ t4.5

Integer = 2 Integer = 3 Integer = 3 Float = 4.5

//Press <Enter> then <CTRL><d> to stop giving input.

Total number of Integer = 5 & Float = 3

$

\*/

**Program4:** Lex code for tokenizing C-code

**%{**

**int n=0;**

**%}**

**%%**

**"while"|"if"|"else" {n++; printf("\t keywords: %s",yytext);} "int"|"float" {n++; printf("\t keywords: %s",yytext);}**

**[a-zA-Z\_][a-zA-Z0-9\_]\* {n++; printf("\t Identifier: %s",yytext);} "<="|"=="|"="|"++"|"-"|"\*"|"+""("|")"|"," {n++; printf("\t operator: %s",yytext);} "{"|"}"|";" {n++; printf("\t Seperators: %s",yytext);}**

**-?[0-9]+"."[0-9]+ {n++; printf("\t Float %s",yytext);}**

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

**. ;**

**%%**

**main()**

**{ yylex();**

**printf("\nTotal number of token = %d \n",n); }**

**/\*** $ lex token.l

$ gcc lex.yy.c -lfl

$ ./a.out

int p=1,d=0,r=4;

float m=0.0, n=200.0; while (p <= 3)

{ if(d==0)

{ m= m+n\*r+4.5; d++; }

else

{ r++; m=m+r+1000.0; }

p++;

}

//Press <Enter> then <CTRL><d> to get output. **\*/**

**Program 5:** Design a LEX Code to count the number of total characters, words, white space  .

**%{**

**int n,w,c;**

**%}**

**%%**

**[ \n\t] {n++;}**

**[^ \n\t]+ {w++;c=c+yyleng;}**

**%%**

**main()**

**{**

**extern FILE \*yyin; yyin=fopen("Input.txt","r"); yylex();**

**printf("whitespace=%d word=%d total char=%d \n",n,w,n+c);**

**}**

/\* // before running the prog. create "Input.txt" file using vi editor

// write senences in it like "i am looking

// for you" and save it.

$ lex word-count.l

$ cc lex.yy.c -lfl

$ ./a.out

line=2 word=5 total char=19

$

\*/

**Program 6:** Lex code for replacing multiple whitespaces by single space

**%{**

**%}**

**%%**

**[ \n\**



**. fprintf(yyout,"%s",yytext);**

**%%**

**main()**

**{ extern FILE \*yyin,\*yyout; yyin=fopen("Input.txt","r"); yyout=fopen("Output.txt","w"); yylex(); }**

/\*

// before running the prog. create "Input.txt" file using vi editor

// write senences in it like "i am looking

// for you" and save it.

$ lex remove-whitspace.l

$ cc lex.yy.c -lfl

$ ./a.out

// output will be shown in file "Output.txt"

// content of file will look like "i am looking for you"

$

**\*/**

**Program 7:** Lex code for removing C-comment from C-program.

**%{**

**%}**

**%%**

**"//"[^\n]\* ; "/\*"([^\*]|[\*]+[^/])\*[\*]+"/" ;**

**. fprintf(yyout,"%s",yytext);**

**%%**

**main()**

**{ extern FILE \*yyin,\*yyout; yyin=fopen("Input.c","r");**

**yyout=fopen("Out.c","w"); yylex();**

**}**

**/\* Input.c**

int p=1,d=0,r=4;

float m=0.0, n=200.0; // hello while (p <= 3)

{ if(d==0) /\*this is wrong Method \*/

{ m= m+n\*r+4.5; d++; }

// haha p++; }

**Out.c**

int p=1,d=0,r=4;

float m=0.0, n=200.0; while (p <= 3)

{ if(d==0)

{ m= m+n\*r+4.5; d++; } p++; }

**\*/**

**Program 8:** Design a LEX Code to extract all html tags in the given HTML file at run time and store into Text file given at run time

**%{**

**%}**

**%% "<"[^>]\*">" ;**

**. fprintf(yyout,"%s",yytext);**

**%%**

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

**{ extern FILE \*yyin,\*yyout; yyin=fopen(argv[1],"r");**

**yyout=fopen(argv[2],"w"); yylex();**

**}**

**/\* Input file:**

<html> heloo </html>

<html> whatever </html>

**Output file:**

<html> </html>

<html> </html>

\*/

**Program 9:** Implementation of DFA accepting even number of a and b over input {a, b} with dead state.

**%s A B C F**

**%%**

**<INITIAL>\n printf(" accepted\n");BEGIN INITIAL;**

**<INITIAL>a BEGIN A;**

**<INITIAL>b BEGIN B;**

**<A>a BEGIN INITIAL;**

**<A>b BEGIN C;**

**<A>\n BEGIN INITIAL; printf(" not accepted\n");**

**<B>a BEGIN C;**

**<B>b BEGIN INITIAL;**

**<B>\n BEGIN INITIAL; printf(" not accepted\n");**

**<C>a BEGIN B;**

**<C>b BEGIN A;**

**<C>\n BEGIN INITIAL; printf(" not accepted\n");**

**<A>[^ab\n] BEGIN F;**

**<B>[^ab\n] BEGIN F;**

**<C>[^ab\n] BEGIN F;**

**<INITIAL>[^ab\n] BEGIN F;**

**<F>[^\n] BEGIN F;**

**<F>[\n] BEGIN INITIAL;printf("Invalid Input\n");**

**. ;**

**%%**

**main()**

**{ printf( Enter the String of a and b only:\n ); yylex();**

**}**

**/\*** $ lex even.l

$ gcc lex.yy.c -lfl

$ ./a.out

Enter the String of a and b only: abbbba

accepted abbaa

not accepted abcab

Invalid Input **\*/**

**Program 10:** Design a DFA in LEX Code which accepts string containing third last over input alphabet {a, b}.

|  |  |
| --- | --- |
| **%s A B C D** | **E F G H** |
| **%%** |  |
| **<INITIAL>a** | **BEGIN A;** |
| **<INITIAL>b** | **BEGIN INITIAL;** |
| **<A>a BEGIN** | **D;** |
| **<A>b BEGIN** | **B;** |
| **<B>a BEGIN** | **E;** |
| **<B>b BEGIN** | **C;** |
| **<C>a BEGIN** | **A;** |
| **<C>b BEGIN** | **INITIAL;** |
| **<D>a BEGIN** | **G;** |
| **<D>b BEGIN** | **F;** |
| **<E>a BEGIN** | **A;** |
| **<E>b BEGIN** | **B;** |
| **<F>a BEGIN** | **E;** |
| **<F>b BEGIN** | **C;** |
| **<G>a BEGIN** | **G;** |
| **<G>b BEGIN** | **F;** |

**<INITIAL>\n BEGIN INITIAL;printf("not accepted\n");**

**<A>\n BEGIN INITIAL; printf(" not accepted\n");**

**<B>\n BEGIN INITIAL; printf(" not accepted\n");**

**<C>\n BEGIN INITIAL; printf(" accepted\n");**

**<D>\n BEGIN INITIAL; printf(" not accepted\n");**

**<E>\n BEGIN INITIAL; printf(" accepted\n");**

**<F>\n BEGIN INITIAL; printf(" accepted\n");**

**<G>\n BEGIN INITIAL; printf(" accepted\n");**

**<INITIAL>[^ab\n] BEGIN H;**

**<A>[^ab\n] BEGIN H;**

**<B>[^ab\n] BEGIN H;**

**<C>[^ab\n] BEGIN H;**

**<D>[^ab\n] BEGIN H;**

**<E>[^ab\n] BEGIN H;**

**<F>[^ab\n] BEGIN H;**

**<G>[^ab\n] BEGIN H;**

**<H>[^\n] BEGIN H;**

**<H>[\n] BEGIN INITIAL; printf("Invalid Input\n");**

**%%**

**main()**

**{**

**printf( Enter the String of a and b only:\n ); yylex();**

**}**

**/\*** $ lex Program10.l

$ gcc lex.yy.c -lfl

$ ./a.out

Enter the String of a and b only: abbbba

not accepted abbaba accepted abcab

Invalid Input **\*/**

**Program 11:** Design a DFA in LEX Code to Identify and print Integer & Float Constants and Identifier.

**%s A B C D Y Z**

**%%**

**<INITIAL>[A-Za-z\_] BEGIN B;**

**<INITIAL>[0-9] BEGIN A;**

**<INITIAL>[.] BEGIN Y;**

**<INITIAL>[^A-Za-z0-9\_.\n] BEGIN Z;**

**<INITIAL>\n BEGIN INITIAL;printf( Not accepted\n );**

**<A>[.] BEGIN C;**

**<A>[0-9] BEGIN A;**

**<A>[A-Za-z\_] BEGIN Y;**

**<A>[^A-Za-z0-9\_.\n] BEGIN Z;**

**<A>\n BEGIN INITIAL; printf( Integer\n );**

**<B>[A-Za-z\_] BEGIN B;**

**<B>[0-9] BEGIN B;**

**<B>[.] BEGIN Y;**

**<B>[^A-Za-z0-9\_.\n] BEGIN Z;**

**<B>\n BEGIN INITIAL; printf( Identifier\n );**

**<C>[0-9] BEGIN D;**

**<C>[.] BEGIN Y;**

**<C>[A-Za-z\_] BEGIN Y;**

**<C>[^A-Za-z0-9\_.\n] BEGIN Z;**

**<C>\n BEGIN INITIAL; printf( Not Accepted\n );**

**<D>[0-9] BEGIN D;**

**<D>[.] BEGIN Y;**

**<D>[A-Za-z\_] BEGIN Y;**

**<D>[^A-Za-z0-9\_.\n] BEGIN Z;**

**<D>\n BEGIN INITIAL; printf( Float\n );**

**<Y>[A-Za-z0-9\_.] BEGIN Y;**

**<Y>[^A-Za-z0-9\_.\n] BEGIN Z;**

**<Y>[\n] BEGIN INITIAL; printf("Not Accepted\n");**

**<Z>[^\n] BEGIN Z;**

**<Z>[\n] BEGIN INITIAL; printf("Invalid Input\n");**

**%%**

**main()**

**{**

**printf( Enter the char [A-Za-z0-9\_.] only:\n ); yylex(); }**

/\*

$ lex Program11.l

$ gcc lex.yy.c -lfl

$ ./a.out

Enter the char [A-Za-z0-9\_.]only: 1568

Integer 0.32

Float Qw\_12 Identifier C.32

Not accepted 12A

Not accepted A2@cd&W Invalid Input

\*/

**Program 12:** Yacc-Lex code for +, -, \* and / of integers with precedence specification explicitly.

# a.y

**%{**

**#include<stdio.h> int yylex(void);**

**void yyerror(char \*);**

**%}**

**%token digit**

**%left + -**

**%left \* /**

**%%**

**S:S E \ {$$=$2; printf("output=%d\n",$$); }**



**| ;**

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

**- { $$=$1-$3; }**

**{ $$=$1\*$3; }**

**/ { $$=$1/$3; }**

**| digit { $$ = $1; }**

**;**

**%%**

**int main() { yyparse();**

**return 0; }**

**void yyerror(char \*msg)**

**{ printf("\n%s",msg);**

**printf("\narithematic expression is invalid"); }**

**b.l**

**%{**

**#include<stdlib.h> int yylval; #include"y.tab.h"**

**%}**

**%%**

**[0-9]+ { yylval = atoi(yytext); return digit; } [-+\*/\n] return \*yytext;**

**. ;**

**%%**

**int yywrap(void) { return 1;**

**}**

**/\*** $ yacc -d a.y

$ lex b.l

$ gcc lex.yy.c y.tab.c

$ ./a.out 2+3\*5

output=17

4\*6-3+5\*2-12/3

output=27 2++5

Syntax error **\*/**

**Program 13:** Yacc-Lex code for +, -, \* and / of integers with precedence specified within CFG.

# a.y

**%{**

**#include<stdio.h> int yylex(void);**

**void yyerror(char \*);**

**%}**

**%token digit**

**%%**

**S:S E '\n' {$$=$2; printf("output=%d\n",$$); }**

**| ;**

|  |  |
| --- | --- |
| **E:E '+' T** | **{$$=$1+$3;}** |
| **|E '-' T** | **{$$=$1-$3;}** |
| **|T** | **{$$ = $1;}** |
| **;** |  |
| **T:T '\*' F** | **{$$=$1\*$3;}** |
| **|T '/' F** | **{$$=$1/$3;}** |
| **|F** | **{$$=$1;}** |
| **;** |  |
| **F:digit** | **{$$=$1;}** |
| **%%** |  |

**int main() {**

**yyparse();**

**return 0; }**

**void yyerror(char \*msg)**

**{ printf("\n%s",msg);**

**printf("\narithematic expression is invalid"); }**

**b.l**

**%{**

**#include<stdlib.h> int yylval; #include"y.tab.h"**

**%}**

**%%**

**[0-9]+ {yylval = atoi(yytext);return digit;} [-+\*/\n] return \*yytext;**

**. ;**

**%%**

**int yywrap(void) { return 1;**

**}**

**/\*** $ yacc -d a.y

$ lex b.l

$ gcc lex.yy.c y.tab.c

$ ./a.out 5+6\*9-15/3

output=54 20++5

Syntax error **\*/**

**Program 14:** Yacc-Lex code for converting infix expression to postfix expression.

**itp.l**

**%{**

**#include"y.tab.h" extern int yylval;**

**%}**

**%%**

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

**\n return 0;**

**. return \*yytext;**

**%%**

**int yywrap(){**

**return 1;**

**}**

**itp.y**

**%{**

**#include<stdio.h>**

**%}**

**%token NUM**

**%left '+' '-'**

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

**%right NEGATIVE**

**%%**

**S: E {printf("\n");}**

**;**

**E: E '+' E {printf("+");}**

**| E '\*' E {printf("\*");}**

**| E '-' E {printf("-");}**

**| E '/' E {printf("/");}**

**| '(' E ')'**

**| '-' E %prec NEGATIVE {printf("-");}**

**| NUM {printf("%d", yylval);}**

**;**

**%%**

**int main(){**

**yyparse();**

**}**

**int yyerror (char \*msg) {**

**return printf ("error YACC: %s\n", msg);**

**}**

**/\*** $ yacc -d itp.y

$ lex itp.l

$ gcc lex.yy.c y.tab.c

$ ./a.out 5+6\*9-2

Output: 569\*+2- 3-9/5

Output: 395/- 20++5

Syntax error **\*/**

**Program 15:** Yacc-Lex code for Desk calculator.

**Cal.l**

**%{**

**#include<stdlib.h> int yylval; #include"y.tab.h"**

**%}**

%%

**[a-z] {yylval=\*yytext-'a'; return id;}**

**[0-9]+ {yylval=atoi(yytext); return digit;} [-+()=/\*\n] {return \*yytext;}**

**[ \t] ;**

**. yyerror("invalid character");**

%%

**int yywrap(void) { return 1;**

**}**

**Cal.y**

**%token id digit**

**%left '+' '-'**

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

**%{ #include<stdio.h> void yyerror(char \*); int yylex(void);**

**int sym[26];**

**%}**

**%%**

**P: P S '\n'**

**| ;**

**S: E {printf("Output: %d\n",$1);}**

**| id '=' E {sym[$1]=$3;}**

**;**

**E: digit {$$=$1;}**

**| id { $$ = sym[$1]; }**

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

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

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

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

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

**;**

**%%**

**void yyerror(char \*s)**

**{**

**fprintf(stderr, "%s\n", s); return 0;**

**}**

**int main(void)**

**{**

**yyparse(); return 0 }**

**/\*** $ yacc -d cal.y

$ lex cal.l

$ gcc lex.yy.c y.tab.c

$ ./a.out

7+6\* 9- 15 / 3

Output: 56 a=5

a

Output: 5 b=6

a\*b Output: 30 c=a+b

c

Output: 11 2@+3

Invalid character Output: 5

e

Output: 0 e=a+b\*c-b Output: 65

2++3

Syntax error

\*/