SYSTEM SOFTWARE LAB MANUAL

(LEX PROGRAMS)

**1. Program to count the number of identifiers.**

%{

#include<stdio.h>

int c=0;

%}

digit [0-9]

letter [a-zA-Z]

id {letter}({letter}|{digit})\*

%%

{id} {c++;}

%%

main()

{

yylex();

printf(“\nNo of identifiers = %d\n”, c);

}

int yywrap()

{

return 1;

}

2. Program to count the number of vowels and consonants in a given string.

%{

#include<stdio.h>

int v=0, c=0;

%}

%%

[aeiouAEIOU] {v++ ;}

[a-zA-Z] {c++ ;}

%%

main()

{

printf(“Enter the string :\n”);

yylex();

printf(“\nNo of vowels = %d\nNo of consonants = %d\n”, v, c);

}

int yywrap()

{

return 1;

}

3. Program to count the number of characters, words, spaces and lines in a given input file.

%{

#include<stdio.h>

Int c=0, w=0, s=0, l=0;

%}

word [^ \t\n,\.:]+

eol [\n]

blank [ ]

%%

{word} {w++; c=c+yyleng;}

{blank} {s++;}

{eol} {l++;}

**.** {c++;}

%%

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

{

if(argc!=2)

{

printf(“Enter the input file\n”);

exit(0);

}

yyin=fopen(argv[1],”r”);

yylex();

printf(“No of characters = %d\nNo of words = %d\nNo of spaces = %d\nNo of lines = %d”, c, w, s, l);

}

int yywrap()

{

return 1;

}

**4. Program to count no of:**

**a) +ve and –ve integers**

**b) +ve and –ve fractions**

%{

#include<stdio.h>

int pi=0, ni=0, pf=0, nf=0;

%}

%%

[-][0-9]+ {ni++;}

[+]?[0-9]+ {pi++;}

[+]?[0-9]\***\.**[0-9]+ {pf++;}

[-][0-9]\* **\.**[0-9]+ {nf++;}

%%

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

{

if(argc!=2)

{

printf(“Enter the input file\n”);

exit(0);

}

yyin=fopen(argv[1],”r”);

yylex();

printf(“No of +ve integers = %d\nNo of –ve integers = %d\nNo of +ve fractions = %d\n No of –ve fractions = %d\n”, pi, ni, pf, nf);

}

int yywrap()

{

return 1;

}

**5. Program to count the no of comment line in a given C program. Also eliminate them and copy that program into separate file.**

%{

#include<stdio.h>

int c=0;

%}

%S COMMENT

%%

\/\/.\*\n {c++;}

“/\*”[**.**]\*”\*/” {c++;}

“/\*” {BEGIN COMMENT ;}

<COMMENT>\n {com++ ;}

<COMMENT>”\*/” {BEGIN 0; c++ ;}

<COMMENT>**.|\**n {;}

**.**|\n {fprintf(yyout,”%s”,yytext);

%%

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

{

if(argc!=3)

{

printf(“Enter the files\n”);

exit(0);

}

yyin=fopen(argv[1],”r”);

yyout=fopen(argv[2],”w”);

yylex();

printf(“No. of comment lines = %d\n”, c);

}

int yywrap()

{

return 1;

}

6. Program to count the no of ‘scanf’ and ‘printf’ statements in a C program. Replace them with ‘readf’ and ‘writef’ statements respectively.

%{

#include<stdio.h>

int pc=0, sc=0;

%}

%%

“printf” { fprintf(yyout, ”writef”); pc++;}

“scanf” { fprintf(yyout, ”readf”); sc++;}

%%

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

{

if(argc!=3)

{

printf(“Enter the files\n”);

exit(0);

}

yyin=fopen(argv[1],”r”);

yyout=fopen(argv[2],”w”);

yylex();

printf(“No of printf statements = %d\n No of scanf

statements = %d\n”, pc, sc);

}

int yywrap()

{

return 1;

}

7. Program to recognize a valid arithmetic expression and identify the identifiers and operators present. Print them separately.

%{

#include<stdio.h>

#include<string.h>

int noprt=0, nopnd=0, valid=1, top=-1, k=0, l=0, j=0;

char opnd[10][10], oprt[10][10], a[100];

%}

%%

“(“ {top++; a[top]=’(‘ ; }

“{“ {top++; a[top]=’{‘ ; }

“[“ {top++; a[top]=’[‘ ; }

“)” {if(a[top]!=’(‘)

{

valid=0;

return=0;

}

else

{

top--;

}

}

“}” { if(a[top]!=’{‘)

{

valid=0;

return=0;

}

else

{

top--;

}

}

“]”{ if(a[top]!=’[‘)

{

valid=0;

return=0;

}

else

{

top--;

}

}

[+-\*/] {noprt++; strcpy(oprt[l], yytext); l++;}

[0-9]+|[a-zA-Z][a-zA-Z0-9\_]\* {nopnd++; strcpy(opnd[j],yytext); j++;}

%%

int main()

{

printf(“Enter the expression :\n”);

yylex();

if(valid==1 && top==-1 && (nopnd-noprt)==1)

{

printf(“Expression is valid\n”);

printf(“\nOperators :\n”);

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

{

printf(“%s\t”, oprt[k]);

}

printf(“\nOperands :\n”);

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

{

printf(“%s\t”, opnd[k]);

}

}

else

{

printf(“\nExpression is invalid\n”);

}

}

int yywrap()

{

return 1;

}

8. Program to recognize whether a given sentence is simple or compound.

%{

#include<stdio.h>

Int s=1;

%}

%%

[ \t\n]+[aA][nN][dD][ \t\n]+ {s=0;}

[ \t\n]+[oO][rR][ \t\n]+ {s=0;}

[ \t\n]+[bB][uU][tT][ \t\n]+ {s=0;}

**.** {;}

%%

int main()

{

printf(“Enter the statement :\n”);

yylex();

if(s==1)

{

printf(“\nSimple statement\n”);

}

else

{

printf(“\nCompound statement\n”);

}

return 0;

}

int yywrap()

{

return 1;

}

9. Program to recognize and count the number of identifiers in a given input file.

%{

#include<stdio.h>

int c=0;

%}

%%

[a-zA-Z][a-zA-Z0-9\_]\* {c++ ; ECHO; printf(“\n”);}

**.**+ { ;}

\n { ;}

%%

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

{

if(argc!=2)

{

printf(“Enter input file\n”);

exit(0);

}

yyin=fopen(argv[1],”r”);

yylex();

printf(“No of identifiers = %d\n”,id);

return 0;

}

int yywrap()

{

return 1;

}

10. Program to count the number of words in a string.

%{

#include<stdio.h>

int wc=0;

%}

%%

char [A-Za-z\\_\’\.\”]

num ([0-9])+

delim [“ “\n\t]

whitespace {delim}+

words {char}+

%%

{words} {wc++;}

{whitespace} {;}

{num} { }

%%

void main()

{

yylex();

printf(“\nNo of words : %d\n”, wc);

}

int yywrap()

{

return 1;

}

# YACC PROGRAMS

1. Program to recognize a valid declaration statement.

Yacc Part

%{

#include<stdio.h>

#include<stdlib.h>

%}

%token SP SC CM NL ID INT CHAR FLOAT

%%

S : type SP list SC NL{printf(“Valid\n”);exit(0);}

;

type : INT

| CHAR

| FLOAT

;

list : list SP CM SP ID

| list CM ID

| ID

;

%%

int yyerror(char \*msg)

{

printf(“Invalid Expression\n”);

exit(0);

}

main ()

{

printf(“Enter the expression\n”);

yyparse();

}

Lex Part

%{

#include “y.tab.h”

%}

%%

[ ] {return SP;}

[\n] {return NL;}

[;] {return SC;}

[,] {return CM;}

“int” {return INT;}  
“char” {return CHAR;}

“float” {return FLOAT;}

[a-zA-Z]+ {return ID;}

. {;}

%%

**2. Program to recognize nested IF control statements and display the**

**levels of nesting.**

Yacc Part

%{

#include<stdio.h>

#include<stdlib.h>

int count=0;

%}

%token IF RELOP S NUMBER ID

%%

stmt : if\_stmt { printf(“No of nested if statements = %d\n”, count); exit(0);}

;

if\_stmt : IF ‘(‘ cond ‘)’ if\_stmt {count++;}

| S;

;

cond : x RELOP x

;

x : ID

| NUMBER

;

%%

int yyerror(char \*msg)

{

printf(“Invalid Expression\n”);

exit(0);

}

void main ()

{

printf(“Enter the statement”);

yyparse();

}

Lex Part

%{

#include “y.tab.h”

%}

%%

“if” { return IF; }

[sS][0-9]\* {return S;}

“<”|”>”|”==”|”!=”|”<=”|”>=” { return RELOP; }

[0-9]+ { return NUMBER; }

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

\n { ; }

**.** { return yytext[0]; }

%%

**3. Program to check the syntax of a simple expression involving operators +, -, \* and /**

Yacc Part

%{

#include<stdio.h>

#include<stdlib.h>

%}

%token NUMBER ID NL

%left ‘+’ ‘-‘

%left ‘\*’ ‘/’

%%

stmt : exp NL { printf(“Valid Expression”); exit(0);}

;

exp : exp ‘+’ exp

| exp ‘-‘ exp

| exp ‘\*’ exp

| exp ‘/’ exp

| ‘(‘ exp ‘)’

| ‘[‘ exp ‘]’

| ‘{‘ exp ‘}’

| ID

| NUMBER

;

%%

int yyerror(char \*msg)

{

printf(“Invalid Expression\n”);

exit(0);

}

void main ()

{

printf(“Enter the expression\n”);

yyparse();

}

Lex Part

%{

#include “y.tab.h”

%}

%%

[0-9]+ {return NUMBER; }

\n {return NL ;}

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

**.** {return yytext[0]; }

%%

**4. Program to recognize a valid variable, which starts with a letter,**

**followed by any number of letters or digits.**

## Yacc Part

%{

#include<stdio.h>

#include<stdlib.h>

%}

%token DIGIT LETTER UND NL

%%

stmt : variable NL { printf(“Valid Identifiers\n”); exit(0);}

;

variable : LETTER alphanumeric

;  
alphanumeric: LETTER alphanumeric

| DIGIT alphanumeric

| UND alphanumeric

| LETTER

| DIGIT

| UND

;

%%

int yyerror(char \*msg)

{

printf(“Invalid Expression\n”);

exit(0);

}

void main ()

{

printf(“Enter the variable name\n”);

yyparse();

}

Lex Part

%{

#include “y.tab.h”

%}

%%

[a-zA-Z] { return LETTER ;}

[0-9] { return DIGIT ; }

[\_] { return UND; }

[\n] { return NL ;}

**.** { return yytext[0]; }

%%

5. Program to evaluate an arithmetic expression involving operating +, -,\* and /.

Yacc Part

%{

#include<stdio.h>

#include<stdlib.h>

%}

%token NUMBER ID NL

%left ‘+’ ‘-‘

%left ‘\*’ ‘/’

%%

stmt : exp NL { printf(“Value = %d\n”,$1); exit(0);}

;

exp : exp ‘+’ exp { $$=$1+$3; }

| exp ‘-‘ exp { $$=$1-$3; }

| exp ‘\*’ exp { $$=$1\*$3; }

| exp ‘/’ exp { if($3==0)

{

printf(“Cannot divide by 0”);

exit(0);

}

else

$$=$1/$3;

}

| ‘(‘ exp ‘)’ { $$=$2; }

| NUMBER { $$=$1; }

;

%%

int yyerror(char \*msg)

{

printf(“Invalid Expression\n”);

exit(0);

}

void main ()

{

printf(“Enter the expression\n”);

yyparse();

}

Lex Part

%{

#include “y.tab.h”

extern int yylval;

%}

%%

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

\n { return NL ;}

**.** { return yytext[0]; }

%%

**6. Program to recognize strings ‘aabb’ and ‘ab’ using grammar (anbn, n>=0)**

Yacc Part

%{

#include<stdio.h>

#include<stdlib.h>

%}

%token A B NL

%%

stmt : s NL { printf(“Valid String\n”); exit(0) ;}

;

s : A s B

|

;

%%

int yyerror(char \*msg)

{

printf(“Invalid String\n”);

exit(0);

}

void main ()

{

printf(“Enter the String\n”);

yyparse();

}

Lex Part

%{

#include “y.tab.h”

%}

%%

[aA] { return A; }

[bB] { return B; }

\n { return NL ;}

**.** { return yytext[0]; }

%%

**7. Program to recognize the grammar (anb, n>=10)**

### METHOD 1

## YACC PART

%{

#include<stdio.h>

#include<stdlib.h>

%}

%token A B NL

%%

stmt : A A A A A A A A A A s B NL

{

Printf(“Valid String\n”); exit(0);

}

;

s : A s

|

;

%%

int yyerror(char \*msg)

{

printf(“Invalid String\n”);

exit(0);

}

void main ()

{

printf(“Enter the String\n”);

yyparse();

}

Lex Part

%{

#include “y.tab.h”

%}

%%

[aA] { return A; }

[bB] { return B; }

\n { return NL ;}

**.** { return yytext[0]; }

%%

### METHOD 2

## YACC PART

%{

#include<stdio.h>

#include<stdlib.h>

%}

%token A B NL

%%

stmt : S NL

{

printf(“Valid string\n”);

exit(0);

}

;

S : A S B  
 |

;

%%

int yyerror(char \*msg)

{

printf(“Invalid String\n”);

exit(0);

}

void main ()

{

printf(“Enter the String\n”);

yyparse();

}

Lex Part

%{

#include “y.tab.h”

int c=0;

%}

%%

Aaaaaaaaaa[a]\*AAAAAAAAAA[A]\* { return A; }

[bB] { c++;

if(c==1)

{

return B;

}

else

{

printf(“Invalid string\n”);

exit(0);

}

}

\n { return NL ;}

**.** { return yytext[0]; }

%%

**Steps to Execute Lex Program:**

lex <pgm name>

cc lex.yy.c –ll

./a.out

**Steps to execute YACC program:**

yacc –d <yacc\_pgm name>

lex <lex\_pgm\_name>

cc y.tab.c lex.yy.c –ly –ll

./a.out