SYSTEM SOFTWARE LAB MANUAL

# (LEX PROGRAMS)

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

%{

#include<stdio.h> int vowels=0;

int cons=0;

%}

%%

[aeiouAEIOU] {vowels++;} [a-zA-Z] {cons++;}

%%

int yywrap()

{

return 1;

}

main()

{

printf(“Enter the string.. at end press ^d\n”); yylex();

printf(“No of vowels=%d\nNo of consonants=%d\n”,vowels,cons);

}

1. Program to count the number of characters, words, spaces, end of 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 yywrap()

{

return 1;

}

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

{

If(argc!=2)

{

printf(“Usage: <./a.out> <sourcefile>\n”); exit(0);

}

yyin=fopen(argv[1],”r”); yylex();

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

}

1. Program to count no of:

a) +ve and –ve integers

b) +ve and –ve fractions

%{

#include<stdio.h>

int posint=0, negint=0,posfraction=0, negfraction=0;

%}

%%

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

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

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

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

%%

int yywrap()

{

return 1;

}

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

{

If(argc!=2)

{

printf(“Usage: <./a.out> <sourcefile>\n”); exit(0);

}

+ve

}

yyin=fopen(argv[1],”r”); yylex();

printf(“No of +ve integers=%d\n No of –ve integers=%d\n No of

fractions=%d\n No of –ve fractions=%d\n”, posint, negint, posfraction, negfraction);

1. 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 com=0;

%}

%s COMMENT

%%

“/\*”[**.**]\*”\*/” {com++;}

“/\*” {BEGIN COMMENT ;}

<COMMENT>”\*/” {BEGIN 0; com++ ;}

<COMMENT>\n {com++ ;}

<COMMENT>**.** {;}

**.**|\n {fprintf(yyout,”%s”,yytext);

%%

int yywrap()

{

return 1;

}

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

{

If(argc!=2)

{

printf(“Usage: <./a.out> <sourcefile> <destn file>\n”); exit(0);

}

yyin=fopen(argv[1],”r”);

yyout=fopen(argv[2],”w”); yylex();

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

}

1. 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 yywrap()

{

return 1;

}

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

{

if(argc!=2)

{

printf(“Usage: <./a.out> <sourcefile> <destn file>\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);

}

1. 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, m, 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;

}

else

top--;

}

“}” { if(a[top]!=’{‘)

{

valid=0; return;

}

else

top--;

}

“]” { if(a[top]!=’[‘)

{

valid=0; return;

}

else

top--;

}

“+”|”-“|”\*”|”/” { noprt++;

strcpy(oprt[l], yytext); l++;

}

[0-9]+|[a-zA-Z][a-zA-Z0-9\_]\* {nopnd++;

%%

int yywrap()

{

strcpy(opnd[j],yytext); j++;

}

return 1;

}

main()

{

int k;

printf(“Enter the expression.. at end press ^d\n”); yylex();

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

{

}

else

}

printf(“The expression is valid\n”); printf(“The operators are\n”); for(k=0;k<l;k++)

Printf(“%s\n”,oprt[k]); for(k=0;k<l;k++)

Printf(“%s\n”,opnd[k]);

Printf(“The expression is invalid”);

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

%{

#include<stdio.h> Int is\_simple=1;

%}

%%

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

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

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

**.** {;}

%%

int yywrap()

{

return 1;

}

main()

{

int k;

printf(“Enter the sentence.. at end press ^d”); yylex();

if(is\_simple==1)

{

}

else

{

Printf(“The given sentence is simple”);

Printf(“The given sentence is compound”);

}

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

%{

#include<stdio.h> int id=0;

%}

%%

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

**.**+ { ;}

\n { ;}

%%

int yywrap()

{

return 1;

}

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

{

if(argc!=2)

{

printf(“Usage: <./a.out> <sourcefile>\n”); exit(0);

}

yyin=fopen(argv[1],”r”); printf(“Valid identifires are\n”); yylex();

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

}

# YACC PROGRAMS

1. Program to test the validity of a simple expression involving operators

+, -, \* and / Yacc Part

%token NUMBER ID NL

%left ‘+’ ‘-‘

%left ‘\*’ ‘/’

%%

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

;

exp : exp ‘+’ exp

| exp ‘-‘ exp

| exp ‘\*’ exp

| exp ‘/’ exp

| ‘(‘ exp ‘)’

| ID

| NUMBER

;

%%

int yyerror(char \*msg)

{

printf(“Invalid Expression\n”); exit(0);

}

main ()

{

printf(“Enter the expression\n”); yyparse();

}

Lex Part

%{

#include “y.tab.h”

%}

%%

[0-9]+ { return DIGIT; }

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

\n { return NL ;}

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

%%

1. Program to recognize nested IF control statements and display the levels of nesting.

Yacc Part

%token IF RELOP S NUMBER ID

%{

int count=0;

%}

%%

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);

}

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]; }

%%

1. Program to check the syntax of a simple expression involving operators

+, -, \* and / Yacc Part

%token NUMBER ID NL

%left ‘+’ ‘-‘

%left ‘\*’ ‘/’

%%

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

;

exp : exp ‘+’ exp

| exp ‘-‘ exp

| exp ‘\*’ exp

| exp ‘/’ exp

| ‘(‘ exp ‘)’

| ID

| NUMBER

;

%%

int yyerror(char \*msg)

{

printf(“Invalid Expression\n”); exit(0);

}

main ()

{

printf(“Enter the expression\n”); yyparse();

}

Lex Part

%{

#include “y.tab.h”

%}

%%

[0-9]+ { return NUMBER; }

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

\n { return NL ;}

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

%%

1. Program to recognize a valid variable, which starts with a letter, followed by any number of letters or digits.

Yacc Part

%token DIGIT LETTER NL UND

%%

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);

}

main ()

{

printf(“Enter the variable name\n”); yyparse();

}

Lex Part

%{

#include “y.tab.h”

%}

%%

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

[0-9] { return DIGIT ; }

[\n] { return NL ;}

[\_] { return UND; }

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

%%

1. Program to evaluate an arithmetic expression involving operating +, -,

\* and /. Yacc Part

%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)

{

}

else

}

printf(“Cannot divide by 0”); exit(0);

$$=$1/$3;

| ‘(‘ exp ‘)’ { $$=$2; }

| ID { $$=$1; }

| NUMBER { $$=$1; }

;

%%

int yyerror(char \*msg)

{

printf(“Invalid Expression\n”); exit(0);

}

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]; }

%%

1. Program to recognize strings ‘aaab’, ‘abbb’, ‘ab’ and ‘a’ using grammar (anbn, n>=0)

Yacc Part

%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);

}

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]; }

%%

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

%token A B NL

%%

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

{

Printf(“Valid”); exit(0);

}

;

s : s A

|

;

int yyerror(char \*msg)

{

printf(“Invalid String\n”); exit(0);

}

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]; }

%%

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