Compiler Construction Lab Assignment 1

1. Write a FLEX program to identify identifier, keywords, relational operators, arithmetic operators (the list will be created by you in the rule section, according to input specifications and also print individual count of them.

aoper[+ - / \*]

roper[< > = ]

%{

#include <stdio.h>

%}

%%

if|else|while|int|switch|for|char {printf("keyword");}

[a-z]([a-z]|[0-9])\* {printf("identifier");}

{aoper}|- {printf("arithmetic operator");}

{roper}|{roper}{roper} {printf("relational operator");}

.\* {printf("invalid");}

%%

int main()

{

yylex();

return 0;

}

int yywrap()

{

return 1;

}

Output :

int

keyword

func

identifier

+

arithmetic operator

<=

relational operator

<

relational operator

<==

invalid

1. Write a FLEX program to
2. Request input of an even and an odd number
3. indicate input characteristic : Even/Odd
4. check for input's correctness and print result

letter [a-zA-Z]

digit [0-9]

%{

#include <stdio.h>

int e=0;

int o=0;

%}

%%

{digit}\*("0"|"2"|"4"|"6"|"8") {e++;printf("even");}

{digit}\*("1"|"3"|"5"|"7"|"9") {o++;printf("odd");}

.\* {printf("invalid");}

%%

int main()

{

yylex();

if(o==1 && e==1)

printf("successful");

else

printf("failure");

return 0;

}

int yywrap()

{

return 1;

}

Output :

23

odd

66

even

successful

1. Write a FLEX program to count the number of characters, words, and lines in a file, where the file name will be provided by the user. Add line numbers to each line in the file.

%{

#include<stdio.h>

int lines=0, words=0,s\_letters=0,c\_letters=0, num=0, spl\_char=0,total=0;

%}

%%

\n { lines++; words++;}

[\t ' '] words++;

[A-Z] c\_letters++;

[a-z] s\_letters++;

[0-9] num++;

. spl\_char++;

%%

main(void)

{

char a[50];

scanf("%s",a);

yyin= fopen(a,"r");

yylex();

total=s\_letters+c\_letters+num+spl\_char;

printf(" This File contains ...");

printf("\n\t%d lines", lines);

printf("\n\t%d words",words);

printf("\n\t%d small letters", s\_letters);

printf("\n\t%d capital letters",c\_letters);

printf("\n\t%d digits", num);

printf("\n\t%d special characters",spl\_char);

printf("\n\tIn total %d characters.\n",total);

}

int yywrap()

{

return(1);

}

Output :

input

This File contains ...

2 lines

6 words

17 small letters

0 capital letters

0 digits

0 special characters

In total 17 characters.

1. Write a FLEX program to find the length of the longest word in a text.

letter [a-zA-Z]

digit [0-9]

%{

#include <stdio.h>

int max=0;

char a[50];

%}

%%

({letter})+ {

if(yyleng > max)

{

strcpy(a,yytext);

max=yyleng;

}

}

.\*;

%%

int main()

{

yylex();

printf("%d %s",max,a);

return 0;

}

int yywrap()

{

return 1;

}

Output:

hello hello can you hear my name

5 hello

1. Write a BISON program to count the number of digits in a binary number.

%{

#include <stdio.h>

void yyerror();

int yylex();

int yywrap();

int f=0;

%}

%token X Y NL

%%

P:S NL {};

S: X S { f++;}|{};

S: Y S { f++;};

%%

int yywrap()

{

return 1;

}

void yyerror()

{

}

int main()

{

yyparse();

printf("%d",f);

return 0;

}

%{

#include <stdio.h>

#include "y.tab.h"

%}

%%

"0" return X;

"1" return Y;

"\n" return NL;

%%

Output

1001010100101

13

1. Write a BISON program to convert a binary number into decimal number.

|  |
| --- |
| %{  #include<stdio.h>  #include<stdlib.h>  void yyerror(char \*s);  %}  %token ZERO ONE  %%  N: L {printf("\n%d",$$);}  L: L B {$$=$1\*2+$2;}  | B {$$=$1;}  B:ZERO {$$=$1;}  |ONE {$$=$1;};  %%  int main()  {  while(yyparse());  }  yyerror(char \*s)  {  fprintf(stdout,"\n%s",s);  }    // (Lex File : save as .l)    %{  #include<stdio.h>  #include<stdlib.h>  #include"y.tab.h"  extern int yylval;  %}  %%  0 {yylval=0;return ZERO;}  1 {yylval=1;return ONE;}    [ \t] {;}  \n return 0;  . return yytext[0];  %% |

**Output :-**  
101  
5

1. Write a BISON program to recognize the strings belonging to the language L=anbmcp, where m=n+p, Σ={a,b,c}. Output will be as follows.
2. If the string belongs to the language, it will print “valid”.
3. If the string contains any character out of the alphabet set, it has to print “out of alphabet”.

%{

#include <stdio.h>

void yyerror();

int yylex();

int yywrap();

int f=0;

%}

%token X Y Z NL

%%

S:S1 S2 NL {printf("accepted"),f++;};

S1: X S1 Y {}|{};

S2: Y S2 Z {}|{};

%%

int yywrap()

{

return 1;

}

void yyerror()

{

}

int main()

{

yyparse();

if(f==0)

printf("not accepted");

return 0;

}

%{

#include <stdio.h>

#include "y.tab.h"

%}

%%

"a" return X;

"b" return Y;

"c" return Z;

"\n" return NL;

%%

Output :

abbcc

not accepted

1. Write a BISON program to check balanced parentheses in an expression.

%{

#include <stdio.h>

void yyerror();

int yylex();

int yywrap();

int f=0;

%}

%token X Y NL

%%

P:S NL {printf("accepted"),f++;};

S: X S Y S{}|{};

%%

int yywrap()

{

return 1;

}

void yyerror()

{

}

int main()

{

yyparse();

if(f==0)

printf("not accepted");

return 0;

}

%{

#include <stdio.h>

#include "y.tab.h"

%}

%%

"(" return X;

")" return Y;

"\n" return NL;

%%

Output:

()))

not accepted

()

accepted

1. Write a BISON program to convert an infix expression to its corresponding postfix.

%{

#include"y.tab.h"

extern int yylval;

%}

%%

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

\n return 0;

. return \*yytext;

%%

int yywrap(){

return 1;

}

%{

#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);

}

2+6\*2-5/3

262\*+53/-

1. Write a BISON program that will read the C program from the input file to check and identify all valid C variables and functions declarations. A variable can be defined as an alphabet followed by zero or more combination of alpha-numeric letters, i.e. this program will identify all variables declared after any one data type int, char, float, long, double, signed, and unsigned. More precisely, a valid variable comes after a data type, followed by a comma, and a variable followed by a comma, and a variable and so on at the end of the semicolon.
2. A function name could be any valid identifier
3. The syntax of a function declaration is, *return data type, followed by ‘(‘ ,then list of the arguments passed as parameters separated by comma, followed by ‘)’,and again a semicolon.*
4. Write a BISON program that will implement the scientific desktop calculator. The program will take the infix expression as input and evaluate the same.

%{

#include<stdio.h>

#include<math.h>

#include "y.tab.h"

%}

%%

[0-9]+ {

yylval.dval=atoi(yytext);

return NUMBER;

}

[t];

n return 0;

. {return yytext[0];}

%%

void yyerror(char \*str)

{

printf("n Invalid Character...");

}

int main()

{

printf("Enter Expression => ");

yyparse();

return(0);

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* calci.y \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

%{

#include<stdio.h>

int yylex(void);

%}

%union

{

float dval;

}

%token <dval> NUMBER

%left '+' '-'

%left '\*' '/'

%nonassoc UMINUS

%type <dval> exp

%%

state : exp {printf("Answer = %fn",$1);}

;

exp : NUMBER

| exp '+' exp {$$=$1+$3;}

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

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

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

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

| '-' exp %prec UMINUS {$$=-$2;}

;

%%

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Output \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

[a40@localhost ~]$ lex calci.l

[a40@localhost ~]$ yacc -d calci.y

[a40@localhost ~]$ cc lex.yy.c y.tab.c -ll

[a40@localhost ~]$ ./a.out

Enter Expression => 10+7

Answer = 17