**part of speech**

%{

#include <stdio.h>

%}

%%

[\t]+ ; /\* Ignore tabs \*/

Dread|Run|arrives {printf("%s, is a verb", yytext);}

it {printf("%s, is a pronoun", yytext);}

Destiny {printf("%s, is a noun", yytext);}

same {printf("%s, is an adjective", yytext);}

all {printf("%s, is an adverb", yytext);}

from|the {printf("%s, is a preposition", yytext);}

[a-zA-Z]+ {printf("%s, is an invalid token", yytext);}

%%

int yywrap(void) {

return 1;

}

int main(void) {

yylex();

yywrap();

return 0;

}

**\*\*(D)9.Write a program using LEX Tool, to implement a lexical analyzer for given C programming language without Symbol table.**

**INPUT**

**{**

**int a=3;**

**int b=4;**

**float c;**

**c = (a\*a + b\*b) \*2**

**}**

%{

int COMMENT = 0;

%}

%%

[\t]+;

; ;

{ ;

} ;

#.\* {printf("\n%s is a preprocessor directive", yytext);}

int |

float |

char |

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

if |

break |

continue |

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

\+ |

\- |

\\* |

\\ |

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

[a-zA-Z][a-zA-Z0-9]\* {printf("\n%s is an identifier", yytext);}

%%

int yywrap(){

return (1);

}

void main(){

yylex();

yywrap();

}

**(32)**

**(A)10.Write a program to evaluate a given arithmetic expression using YACC specification.**

**INPUT**

**0.33\*12-4-4+(3\*2)**

.l file

%{

#include "y.tab.h" // Include Bison-generated header for YYSTYPE and token types

%}

%%

[0-9]+(\.[0-9]+)? {

yylval.val = atof(yytext); // Assign the atof result to the union member 'val'

return NUMBER;

}

[ \t]+ ; // Ignore spaces and tabs

\n { return 0; } // Return 0 on a newline to signal the end of input

. { return yytext[0]; } // Return any other character as itself

%%

int yywrap() {

return 1;

}

.y file

%{

#include <stdio.h>

#include <stdlib.h>

void yyerror(const char \*s);

int yylex(void);

%}

%union {

double val; // Define a union member for double values

}

%token <val> NUMBER // Specify that NUMBER carries a double

%type <val> E // The non-terminal E also returns a double

%left '+' '-'

%left '\*' '/'

%left '(' ')'

%%

ArithmeticExpression:

E { printf("\nResult = %.2f\n", $1); return 0; }

;

E:

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

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

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

| E '/' E { if ($3 == 0.0) { yyerror("Division by zero"); $$ = 0.0; } else { $$ = $1 / $3; } }

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

| NUMBER { $$ = $1; }

;

%%

void yyerror(const char \*s) {

fprintf(stderr, "Error: %s\n", s);

}

int main() {

printf("\nEnter an Arithmetic Expression (e.g., 2.5 + 3.2 \* 5):\n");

if (!yyparse())

printf("\nThe arithmetic expression is valid.\n");

else

printf("\nInvalid expression.\n");

return 0;

}

%{

#include "y.tab.h"

extern void yyerror(const char \*s); // Declaration of yyerror

%}

**(A)11.Write a program to evaluate a given variable name using YACC specification.**

**SAMPLE INPUT**

**1) pune**

**2) PUNE**

**3) Pune1**

**4) pUNE\_2**

.l file

%%

[a-zA-Z\_][a-zA-Z\_0-9]\* { return IDENTIFIER; }

. { yyerror("Invalid input"); }

%%

int yywrap(void) {

return 1;

}  
  
.y file

%{

#include <stdio.h>

#include <stdlib.h>

// Declare yytext for use in the parser

extern char\* yytext;

void yyerror(const char \*s);

int yylex(void);

%}

%token IDENTIFIER

%%

input:

identifiers

;

identifiers:

| identifiers IDENTIFIER { printf("Valid variable name: %s\n", yytext); }

;

%%

void yyerror(const char \*s) {

fprintf(stderr, "Error: %s\n", s);

exit(1);

}

int main(void) {

printf("Enter variable names (Ctrl+D to end):\n");

yyparse();

return 0;

}

(B)11.Write a program to convert small case letters to UPPER case or vise versa using YACC specification.

SAMPLE INPUT

1. Pune – pUNE

2. PUNE –pune

.l file

%{

#include "y.tab.h"

%}

%%

[a-z] { yylval.c = yytext[0] - 32; return LETTER; }

[A-Z] { yylval.c = yytext[0] + 32; return LETTER; }

\n { return NEWLINE; }

. { return yytext[0]; }

%%

int yywrap() {

return 1;

}

.y file

%{

#include <stdio.h>

int yylex();

void yyerror(const char \*);

%}

%union {

char c;

}

%token <c> LETTER

%token NEWLINE

%%

input: /\* empty \*/

| input line

line: letters NEWLINE { printf("\n"); }

;

letters: LETTER { printf("%c", $1); }

| letters LETTER { printf("%c", $2); }

;

%%

int main() {

yyparse();

return 0;

}

void yyerror(const char \*s) {

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

}

**(C)11.Write a program to evaluate a given built-in functions using YACC specification.**

**INPUT**

**1.u= sqrt(36)**

**2. v = strlen(“pune”)**

.l file

%{

#include "y.tab.h"

#include <stdlib.h>

#include <string.h>

%}

%option noyywrap

%%

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

\"[^\"]\*\" { yylval.sval = strdup(yytext); return STRING; }

"sqrt" { return SQRT; }

"strlen" { return STRLEN; }

"=" { return '='; }

[ \t\n]+ { /\* ignore whitespace \*/ }

. { return yytext[0]; }

%%

.y file

%{

#include <stdio.h>

#include <math.h>

#include <string.h>

extern int yylex();

void yyerror(const char \*s) { fprintf(stderr, "%s\n", s); }

%}

%union {

int ival;

char \*sval;

}

%token <sval> STRING

%token <ival> NUMBER SQRT STRLEN

%type <ival> expr function

%type <sval> var

%%

program:

program statement

|

;

statement:

var '=' expr { printf("%s = %d\n", $1, $3); }

;

var:

'u' { $$ = strdup("u"); }

| 'v' { $$ = strdup("v"); }

;

expr:

function

;

function:

SQRT '(' NUMBER ')' { $$ = (int) sqrt($3); }

| STRLEN '(' STRING ')' { $$ = strlen($3) - 2; /\* Subtract 2 for the quotes \*/ }

;

%%

int main(void) {

printf("Enter expressions like 'u = sqrt(36)' or 'v = strlen(\"pune\")':\n");

return yyparse();

}

**(D)11.Write a program to evaluate a given built-in functions using YACC specification.**

**INPUT**

**u= sin(12)+cos(12)**

.l file

%{

#include "y.tab.h" // make sure this matches your Bison-generated header

#include <math.h>

#include <string.h>

%}

%option noyywrap

%option yylineno

%%

[uU][a-zA-Z\_]\* { yylval.var = strdup(yytext); return VARIABLE; }

[0-9]+(\.[0-9]+)? { yylval.num = atof(yytext); return NUMBER; }

"sin" { return SIN; }

"cos" { return COS; }

"+" { return PLUS; }

"-" { return MINUS; }

"\*" { return TIMES; }

"/" { return DIVIDE; }

"(" { return LPAREN; }

")" { return RPAREN; }

"=" { return EQUALS; }

";" { return SEMICOLON; }

[ \t]+ ; // Ignore whitespace

. { return yytext[0]; } // Handle other characters

%%

.y file

%{

#include <stdio.h>

#include <stdlib.h>

#include <math.h> // Include for math functions

#include "y.tab.h" // Ensure this is the header generated by Flex

void yyerror(const char \*s);

int yylex(void);

extern int yylineno; // External declaration if using %option yylineno in Flex

double vars[256]; // Simple variable storage based on ASCII index

%}

%union {

double num; // For numerical values

char\* var; // For variable names

}

%token <var> VARIABLE

%token <num> NUMBER

%token SIN COS

%token PLUS MINUS TIMES DIVIDE

%token LPAREN RPAREN

%token EQUALS SEMICOLON

%type <num> expression term factor

%type <var> assignment

%%

input:

| input line

;

line:

assignment SEMICOLON { printf("%s = %f\n", $1, vars[$1[0]]); }

| error SEMICOLON { yyerror("syntax error"); }

;

assignment:

VARIABLE EQUALS expression { vars[$1[0]] = $3; $$ = $1; }

;

expression:

expression PLUS term { $$ = $1 + $3; }

| expression MINUS term { $$ = $1 - $3; }

| term { $$ = $1; }

;

term:

term TIMES factor { $$ = $1 \* $3; }

| term DIVIDE factor { $$ = $1 / $3; }

| factor { $$ = $1; }

;

factor:

NUMBER { $$ = $1; }

| VARIABLE { $$ = vars[$1[0]]; }

| LPAREN expression RPAREN { $$ = $2; }

| SIN LPAREN expression RPAREN { $$ = sin($3); }

| COS LPAREN expression RPAREN { $$ = cos($3); }

;

%%

void yyerror(const char \*s) {

fprintf(stderr, "Error near line %d: %s\n", yylineno, s);

}

int main(void) {

printf("Enter expressions (e.g., 'u = sin(12) + cos(12);'):\n");

yyparse();

return 0;

}

**(E)11.Write a program to evaluate a given built-in functions using YACC specification.**

**INPUT**

**p= pow(3,2) / log (24)**

.l file

%{

#include "y.tab.h"

#include <math.h>

#include <string.h>

extern void yyerror(const char \*);

%}

%option noyywrap

%%

[ \t\n]+ { /\* Ignore whitespace \*/ }

[a-zA-Z\_][a-zA-Z0-9\_]\* {

if (strcmp(yytext, "pow") == 0) return POW;

if (strcmp(yytext, "log") == 0) return LOG;

yylval.str = strdup(yytext);

return IDENTIFIER;

}

[0-9]+(\.[0-9]+)? { yylval.num = atof(yytext); return NUMBER; }

"=" { return '='; }

";" { return ';'; }

"(" { return '('; }

")" { return ')'; }

"," { return ','; }

"+" { return '+'; }

"-" { return '-'; }

"/" { return '/'; }

. { yyerror("Invalid character"); }

%%

.y file

%{

#include <stdio.h>

#include <math.h>

#include <stdlib.h>

#include <string.h>

void yyerror(const char \*s);

extern int yylex();

%}

%union {

double num; // For numerical values

char\* str; // For string values (e.g., identifiers)

}

%token <str> IDENTIFIER

%token <num> NUMBER

%token POW LOG

%type <num> expr // Expressions return a numeric value

%type <num> statement // Statements return a numeric value (for expression statements)

%left '+' '-'

%left '\*' '/'

%right NEG

%%

program:

| program statement

;

statement:

expr ';' { printf("Result = %lf\n", $1); }

| IDENTIFIER '=' expr ';' {

printf("%s = %lf\n", $1, $3);

free($1);

}

;

expr:

NUMBER { $$ = $1; }

| IDENTIFIER { printf("Variable [%s] used, but not defined in this scope.\n", $1); free($1); $$ = 0; }

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

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

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

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

| '-' expr %prec NEG { $$ = -$2; }

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

| POW '(' expr ',' expr ')' { $$ = pow($3, $5); }

| LOG '(' expr ')' { $$ = log($3); }

;

%%

void yyerror(const char \*s) {

fprintf(stderr, "Error: %s\n", s);

}

int main(void) {

printf("Enter an expression:\n");

yyparse();

return 0;

}