**Experiment 9**

**Aim:** Yacc code for Project number 3.

**Input of program**- Grammar and input string to be parsed in our project's language

**Output of program**- appropriate action.

**Tools:** gcc compiler, Text editor, flex, bison, make.

**Procedure:**

* Flex code (laymanCalculator.l)

%{

#include<stdio.h>

#include "y.tab.h"

#include<stdlib.h>

extern int yylineno;

extern int yyerror(const char \*s);

%}

letter [A-Z]

digit [0-9]

whitespace [ \t\r]+

keyword #START|#END|DECLARE|ASSIGN|ANSWER|RESULT

id {letter}{digit}

operation ADD|SUB|MUL|DIV

word {letter}+

number {digit}+(\.{digit}+)?

eol [\.?]?{whitespace}?[\n]+

comment "\*\*"[^\n]\*

%%

{keyword} {

if(!strcmp(yytext, "#START")){

return START;

}else if(!strcmp(yytext, "#END")){

return END;

}else if(!strcmp(yytext, "DECLARE")){

return DECLARE;

}else if(!strcmp(yytext, "ASSIGN")){

return ASSIGN;

}else if(!strcmp(yytext, "ANSWER")){

return ANSWER;

}else if(!strcmp(yytext, "RESULT")){

return RESULT;

}

}

{operation} {

if(!strcmp(yytext, "ADD")){

return ADD;

}else if(!strcmp(yytext, "SUB")){

return SUB;

}else if(!strcmp(yytext, "MUL")){

return MUL;

}else if(!strcmp(yytext, "DIV")){

return DIV;

}

}

{id} {

strcpy(yylval.id, yytext);

return ID;

}

{word} {

strcpy(yylval.word, yytext);

return WORD;

}

{number} {

yylval.number=atof(yytext);

return NUMBER;

}

{eol} {

if(yytext[0]=='.'){

return DOT;

}else if(yytext[0]=='?'){

return QMARK;

}

}

{comment} { }

{whitespace} { }

. {

char msg[256];

sprintf(msg, "Lexical Error : <Invalid symbol : \'%c\'>", yytext[0]);

yyerror(msg);

}

%%

int yywrap(void){

return 1;

}

* Yacc Code (laymanCalculator.y)

%{

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

extern int yylex();

extern int yychar, yylineno;

int yyerror(const char \*s);

float answer[50];

int ans\_index=0;

%}

%start program

%token <number> NUMBER

%token <word> WORD

%token <id> ID

%token START END DECLARE ASSIGN ANSWER RESULT ADD SUB MUL DIV DOT QMARK

%define parse.error verbose

%union{

float number;

char word[30];

char id[2];

}

%%

program:

START statements END {

printf("#Successfully Compiled.\n");

}

| /\* nothing \*/

;

statements:

statement statements

| /\* nothing \*/

;

statement:

declarestmt

| assignstmt

| operationstmt

| operationonanswerstmt

| resultstmt

;

declarestmt:

DECLARE ID DOT {

printf("ID %s Declared.\n", $2);

}

;

assignstmt:

ASSIGN NUMBER WORD ID DOT { if(strcmp($3, "TO")){

char msg[256];

sprintf(msg, "Semantic Error : <Invalid token : \'%s\'>", $3);

yyerror(msg);

}

printf("Value %f is assigned to ID %s.\n", $2, $4);

}

;

operationstmt:

ADD NUMBER WORD NUMBER DOT {

if(strcmp($3, "TO") && strcmp($3, "WITH")){

char msg[256];

sprintf(msg, "Semantic Error : <Invalid token : \'%s\'>", $3);

yyerror(msg);

}

answer[ans\_index++] = $2 + $4;

printf("%f is added in %f and answer is %f.\n", $4, $2, answer[ans\_index-1]);

}

| SUB NUMBER WORD NUMBER DOT {

if(strcmp($3, "FROM")){

char msg[256];

sprintf(msg, "Semantic Error : <Invalid token : \'%s\'>", $3);

yyerror(msg);

}

answer[ans\_index++] = $4 - $2;

printf("%f is subtracted from %f and answer is %f.\n", $2, $4, answer[ans\_index-1]);

}

| MUL NUMBER WORD NUMBER DOT {

if(strcmp($3, "TO") && strcmp($3, "WITH")){

char msg[256];

sprintf(msg, "Semantic Error : <Invalid token : \'%s\'>", $3);

yyerror(msg);

}

answer[ans\_index++] = $2 \* $4;

printf("%f is multipled with %f and answer is %f.\n", $2, $4, answer[ans\_index-1]);

}

| DIV NUMBER WORD NUMBER DOT {

if(strcmp($3, "BY")){

char msg[256];

sprintf(msg, "Semantic Error : <Invalid token : \'%s\'>", $3);

yyerror(msg);

}

answer[ans\_index++] = $2 / $4;

printf("%f is divided by %f and answer is %f.\n", $2, $4, answer[ans\_index-1]);

}

;

resultstmt:

WORD WORD RESULT QMARK {

if(strcmp($1, "WHAT")){

char msg[256];

sprintf(msg, "Semantic Error : <Invalid token : \'%s\'>", $1);

yyerror(msg);

}

if(strcmp($2, "IS")){

char msg[256];

sprintf(msg, "Semantic Error : <Invalid token : \'%s\'>", $2);

yyerror(msg);

}

printf("The final result : %f.\n", answer[ans\_index]);

printf("Final result displayed.\n");

}

;

operationonanswerstmt:

WORD ADD WORD ANSWER DOT {

if(strcmp($1, "NOW")){

char msg[256];

sprintf(msg, "Semantic Error : <Invalid token : \'%s\'>", $1);

yyerror(msg);

}

float tmp=answer[0];

for(int i=1; i<ans\_index; i++)

tmp += answer[i];

answer[0]=tmp;

ans\_index=0;

printf("%s answer are added\n", $3);

}

| WORD SUB WORD ANSWER DOT {

if(strcmp($1, "NOW")){

char msg[256];

sprintf(msg, "Semantic Error : <Invalid token : \'%s\'>", $1);

yyerror(msg);

}

float tmp=answer[0];

for(int i=1; i<ans\_index; i++)

tmp -= answer[i];

answer[0]=tmp;

ans\_index=0;

printf("%s answer are subtracted\n", $3);

}

| WORD MUL WORD ANSWER DOT {

if(strcmp($1, "NOW")){

char msg[256];

sprintf(msg, "Semantic Error : <Invalid token : \'%s\'>", $1);

yyerror(msg);

}

float tmp=answer[0];

for(int i=1; i<ans\_index; i++)

tmp \*= answer[i];

answer[0]=tmp;

ans\_index=0;

printf("%s answer are multipled\n", $3);

}

| WORD DIV WORD ANSWER DOT {

if(strcmp($1, "NOW")){

char msg[256];

sprintf(msg, "Semantic Error : <Invalid token : \'%s\'>", $1);

yyerror(msg);

}

float tmp=answer[0];

for(int i=1; i<ans\_index; i++){

if(answer[i]==0.0 || tmp==0.0){

char msg[256];

sprintf(msg, "Math error : Divide by 0 due to answer number %d.\n", i+1);

yyerror(msg);

}

tmp /= answer[i];

}

answer[0]=tmp;

ans\_index=0;

printf("%s answer are divided\n", $3);

}

;

%%

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

if(argc != 2){

printf("Enter this way please : ./a.out filename\n");

exit(0);

}

extern FILE \*yyin;

yyin = fopen(argv[1], "r");

yyparse();

printf("\n");

}

int yyerror(const char \*errormsg)

{

fprintf(stderr, "at line %d :- %s\n", yylineno, errormsg);

exit(1);

}

* Make File

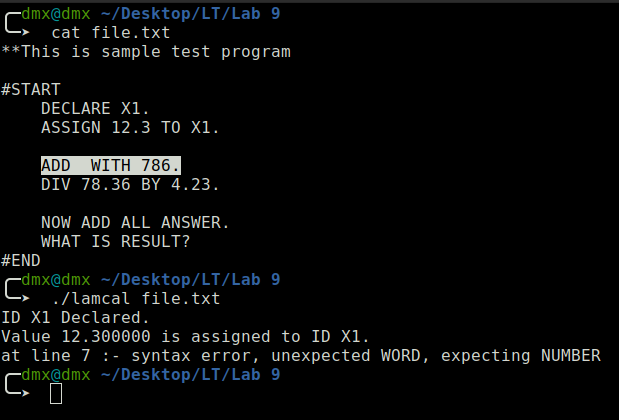
start:

flex -l laymanCalculator.l

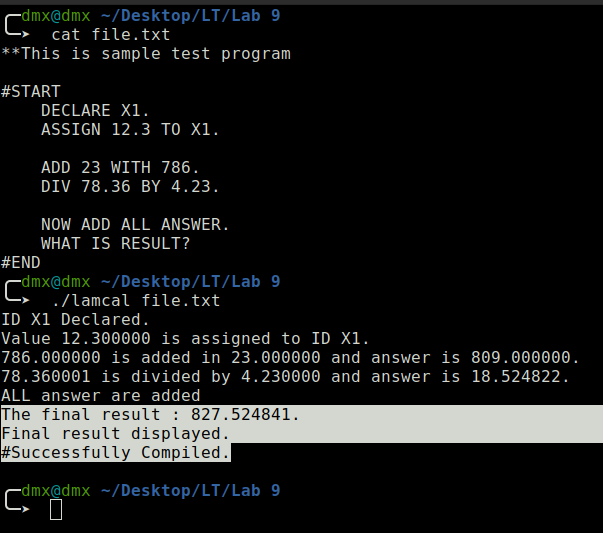
bison -dy laymanCalculator.y

gcc lex.yy.c y.tab.c -o lamcal

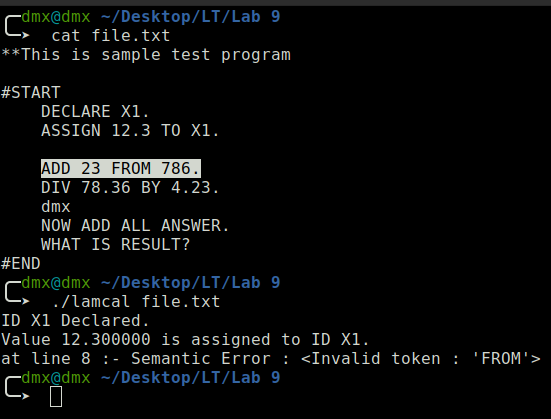
* **Input/Output**



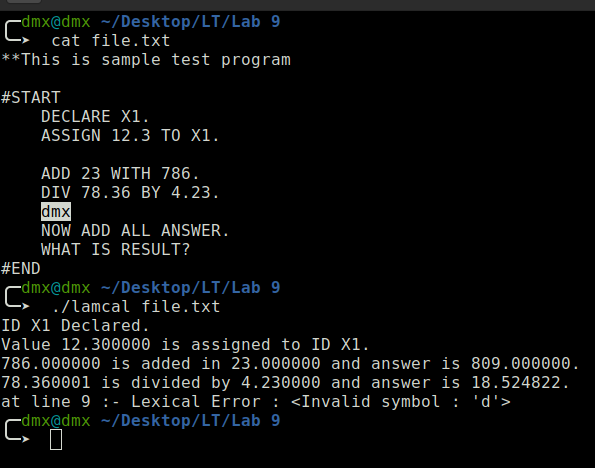
*Fig 9.1 Syntax Error*



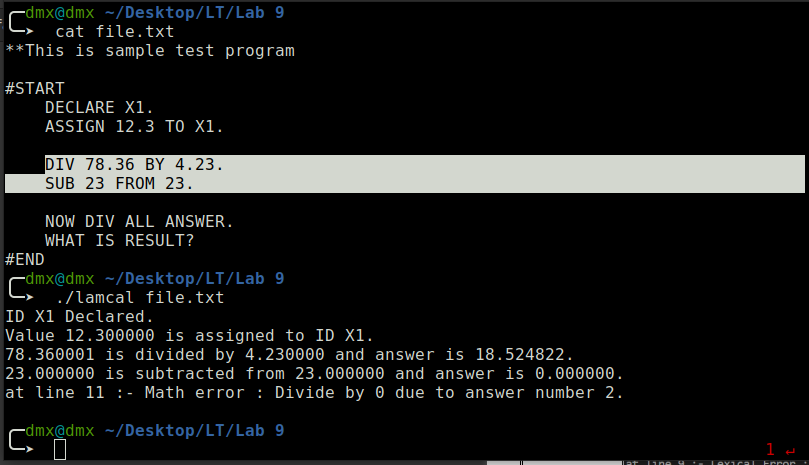
*Fig 9.2 No Error in input file*



*Fig 9.3 Semantic Error*



*Fig 9.4 Lexical Error*



*Fig 9.5 Math Error*