**LPCC ASSIGNMENT-4**

****

**SUBMITTED TO:**

**PROF. PALLAVI REGE**

**VISHWAKARMA INSTITUTE OF INFORMATION TECHNOLOGY, PUNE**

**COMPUTER ENGINEERING DEPARTMENT**

**BY:**

**NAME: ABHISHEK MORE**

**G.R No.: 21810033**

**ROLL NO.: 323036**

**CLASS: T.Y COMP**

**BATCH: COMP C2**

**ASSIGNMENT-4**

**AIM:**

Write a program to evaluate arithmetic expression, built-in functions and variables using Yacc specification.

**THEORY:**

A parser generator is a program that takes as input a specification of a syntax, and produces as output a procedure for recognizing that language. Historically, they are also called compiler-compilers.

YACC (yet another compiler-compiler) is an LALR (LookAhead, Left-to-right, Rightmost derivation producer with 1 lookahead token) parser generator. YACC was originally designed for being complemented by Lex.

**Syntax:**

/\* definitions \*/

....

%%

/\* rules \*/

....

%%

/\* auxiliary routines \*/

....

**Defination part:**

The definition part includes information about the tokens used in the syntax definition:

%token NUMBER

%token ID

• Yacc automatically assigns numbers for tokens, but it can be overridden by

%token NUMBER 621

• Yacc also recognizes single characters as tokens. Therefore, assigned token numbers should

no overlap ASCII codes.

• The definition part can include C code external to the definition of the parser and variable

declarations, within %{ and %} in the first column.

• It can also include the specification of the starting symbol in the grammar:

%start nonterminal

**Rule Part:**

• The rules part contains grammar definition in a modified BNF form.

• Actions is C code in { } and can be embedded inside (Translation schemes).

**Auxiliary Routines Part:**

• The auxiliary routines part is only C code.

• It includes function definitions for every function needed in rules part.

• It can also contain the main() function definition if the parser is going to be run as a program.

• The main() function must call the function yyparse().

**SOURCE CODE:**

1. **To evaluate an arithmetic expression using YACC tool:**

**4a.l**

%{

#include<stdio.h>

#include "y.tab.h"

extern int yylval;

%}

/\* defined section in which we defined the rule and regulation of regular expression which will be going to accept or not \*/

%%

[0-9]+ {

yylval=atoi(yytext);

return NUMBER;

}

[\t] ;

[\n] return 0;

. return yytext[0];

%%

int yywrap()

{

return 1;

}

**4a.y**

%{

#include<stdio.h>

int flag=0;

%}

%token NUMBER

%left '+' '-'

%left '\*' '/' '%'

%left '(' ')'

/\* defined section in which we defined the rule and regulation of regular expression which will going to accept or not \*/

%%

ArithmeticExpression: E{

printf("\nResult=%d\n",$$);

return 0;

}

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

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

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

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

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

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

| NUMBER {$$=$1;}

;

%%

void main()

{

printf("\nEnter Any Arithmetic Expression including (+,-,\*,/ and round bracktes):\n");

yyparse();

if(flag==0)

printf("\nEntered arithmetic expression is Valid\n\n");

}

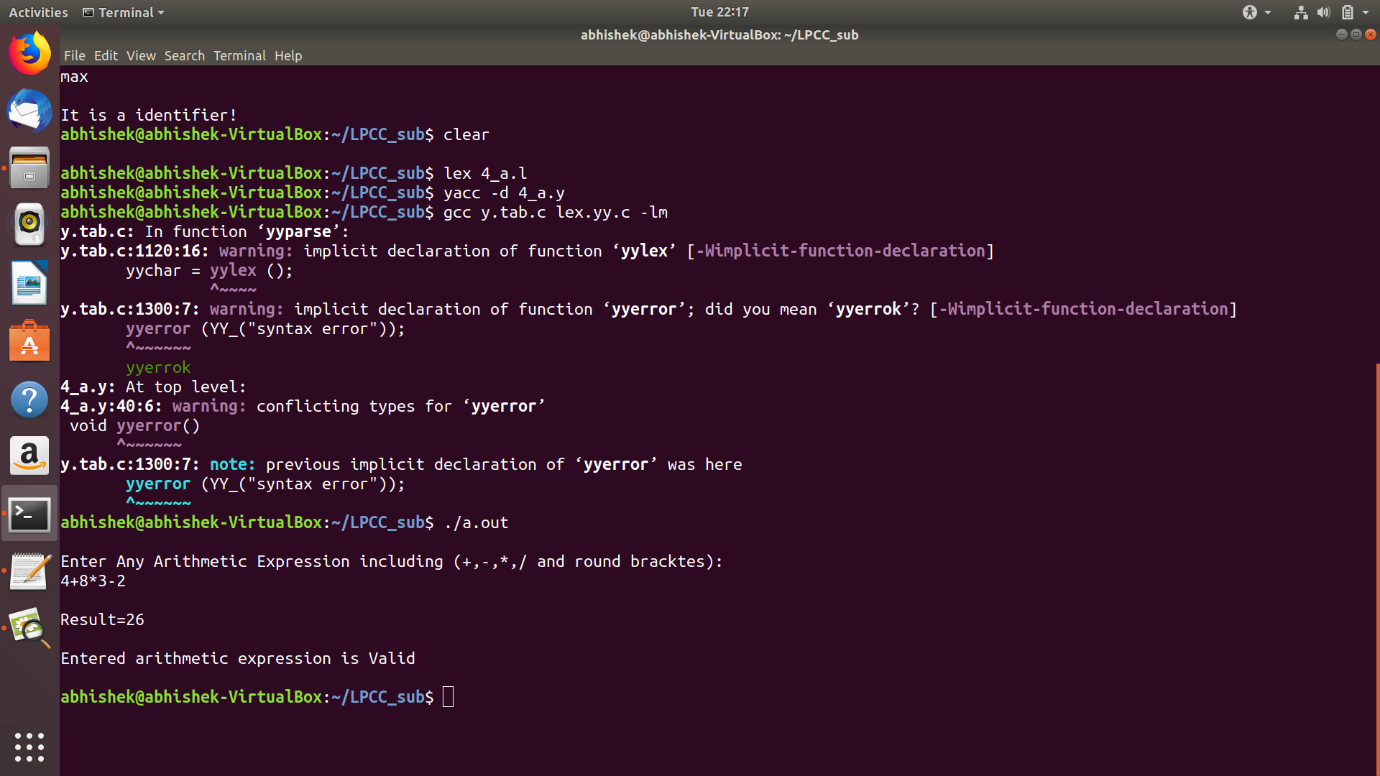
void yyerror()

{

printf("\nEntered arithmetic expression is Invalid\n\n");

flag=1;

}

**OUTPUT:**

1. **To evaluate or check built-in functions using YACC tool:**

**4b.l**

%{

#include"y.tab.h"

#include<stdio.h>

#include<math.h>

#include<string.h>

void yyerror(char \*);

%}

%%

(([0-9]+)|([0-9]\*\.[0-9]+)) {

yylval.dval=atof(yytext);

return NUM;

}

[+-/\*\n,~()] {return (\*yytext);}

sin return SIN;

cos return COS;

tan return TAN;

sqrt return SQRT;

strlen return STRLEN;

exit exit(0);

([a-zA-Z]+) {

yylval.vblno = strlen(yytext);

return ID;

}

[ \t] /\*ignore\*/

%%

int yywrap(void)

{

return 1;

}

**4b.y**

%{

#include<stdio.h>

#include<math.h>

#include<string.h>

void yyerror(char \*);

%}

%union

{

double dval;

int vblno;

char\* strval;

}

%token <dval> NUM

%token <vblno> ID

%token SIN COS TAN SQRT STRLEN

%right '~'

%left '+' '-'

%left '\*' '/'

%type <dval> expression

%type <vblno> str1

%%

program: program statement'\n'

|

;

statement:

expression { printf("%lf\n", $1); }

|str1 { printf("%d\n", $1); }

;

expression:

NUM

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

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

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

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

| '~' expression {$$ = -(1) \* $2;}

| SQRT'('expression')' {$$ = sqrt( $3 );}

| SIN'('expression')' {$$ = sin ($3\*3.142/180);}

| COS'('expression')' {$$ = cos ($3\*3.142/180);}

| TAN'('expression')' {$$ = tan ($3\*3.142/180);}

;

str1:

ID

| STRLEN'('ID')' {$$ = $3;}

%%

main()

{

yyparse();

}

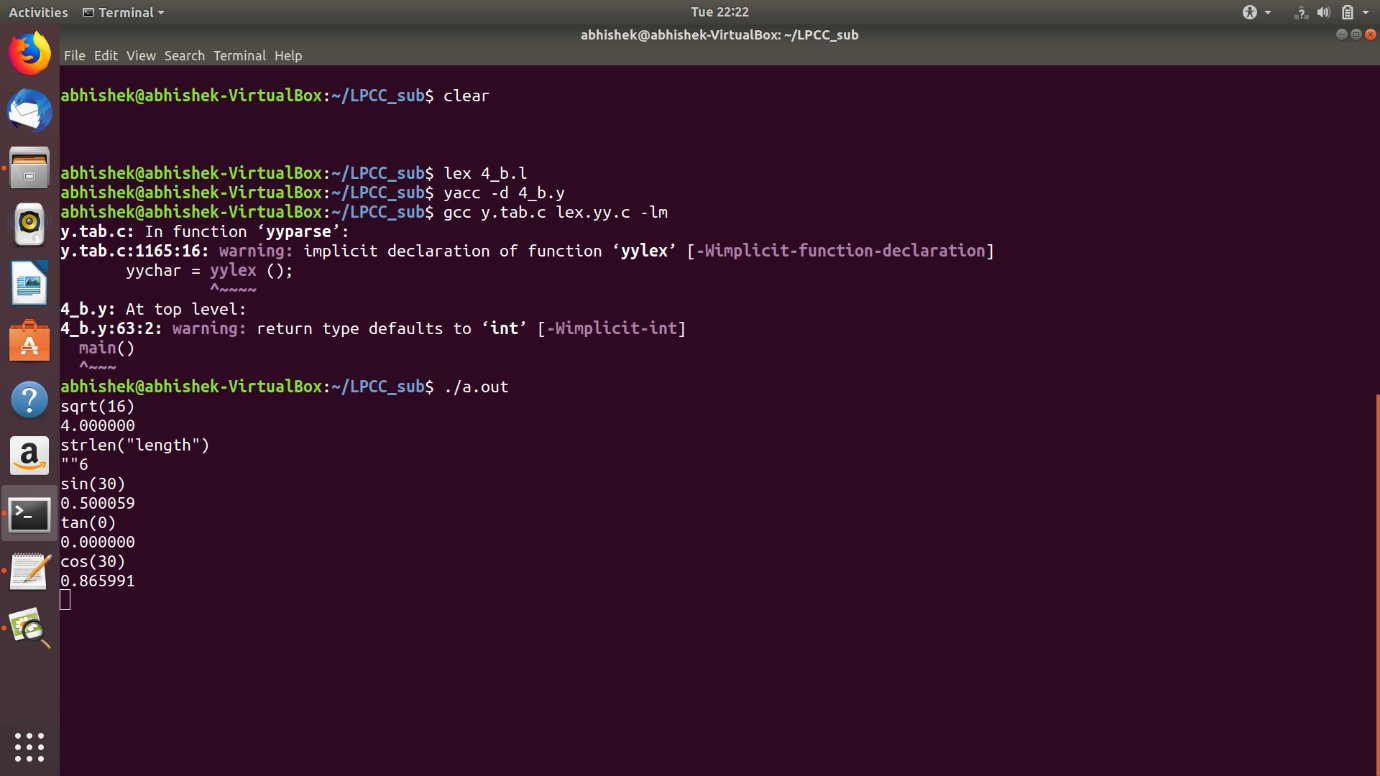
void yyerror(char \*s)

{

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

}

**OUTPUT:**

****

1. **To recognize valid variable name using YACC tool:**

**Assg4c.l**

%{

#include "y.tab.h"

#include<stdio.h>

%}

%%

[a-zA-Z\_][a-zA-Z\_0-9]\* return letter;

[0-9] return digit;

. return yytext[0];

\n return 0;

%%

int yywrap()

{

return 1;

}

**Assg4c.y**

%{

#include<stdio.h>

int valid=1;

%}

%token digit letter

%%

start : letter s

s : letter s

| digit s

|

;

%%

int yyerror()

{

printf("\nIt is not a identifier!\n");

valid=0;

return 0;

}

int main()

{

printf("\nEnter a variable\n");

yyparse();

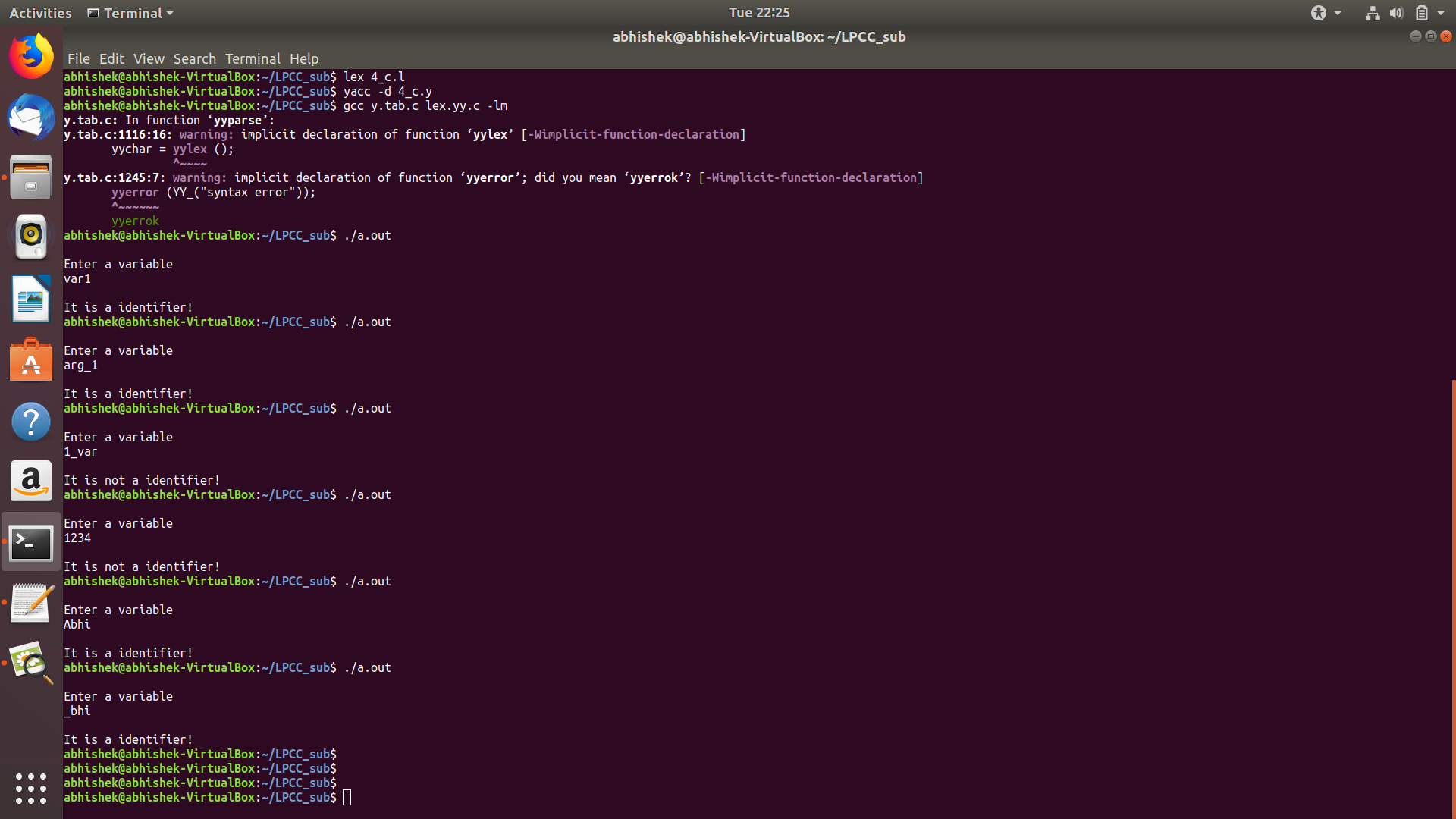
if(valid)

{

printf("\nIt is a identifier!\n");

}

}

**OUTPUT:**