**VISVESVARAYA TECHNOLOGICAL UNIVERSITY**

**“JnanaSangama”, Belgaum -590014, Karnataka.**

****

**LAB REPORT**

**on**

**Compiler Design**

***Submitted by***

**SUBHASH D (1BM21CS221)**

***in partial fulfillment for the award of the degree of***

**BACHELOR OF ENGINEERING**

***in***

**COMPUTER SCIENCE AND ENGINEERING**



**B.M.S. COLLEGE OF ENGINEERING**

**(Autonomous Institution under VTU)**

**BENGALURU-560019**

**Nov-2023 to Feb-2024**

**B. M. S. College of Engineering,**

**Bull Temple Road, Bangalore 560019**

(Affiliated To Visvesvaraya Technological University, Belgaum)

**Department of Computer Science and Engineering**



**CERTIFICATE**

This is to certify that the Lab work entitled “**Compiler Design**” carried out by **SUBHASH D (1BM21CS221),** who is bonafide student of **B.M.S. College of Engineering.** It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the academic semester June-2023 to Sep-2023. The Lab report has been approved as it satisfies the academic requirements in respect of a **Compiler Design(22CS5PCCPD)** work prescribed for the said degree.

**Sonika Sharma D** **Dr. Jyothi S Nayak**

Assistant Professor Professor and Head

Department of CSE Department of CSE

BMSCE, Bengaluru BMSCE, Bengaluru

**Index**

**Part-A: Implementation of Lexical Analyzer, By using C/C++/Java/Python language and using LEX tool.**

|  |  |  |
| --- | --- | --- |
| **S. No.** | **Program Details** | **Page No.** |
| 1 | Write a program to design Lexical Analyzer in C/C++/Java/Python Language (to recognize any five keywords, identifiers, numbers, operators and punctuations) | 1 - 2 |
| 2 | Write a program in LEX to recognize Floating Point Numbers. | 3 - 4 |
| 3 | Write a program in LEX to recognize different tokens: Keywords, Identifiers, Constants, Operators and Punctuation symbols. | 5 - 6 |
| 4 | Write a LEX program that copies a file, replacing each nonempty sequence of white spaces by a single blank. | 7 |
| 5 | Write a LEX program to recognize the following tokens over the alphabets {0,1, ... ,9}  a) The set of all string ending in 00.  b) The set of all strings with three consecutive 222’s.  c) The set of all string such that every block of five consecutive symbols contains at least two 5’s.  d) The set of all strings beginning with a 1 which, interpreted as the binary representation of an integer, is congruent to zero modulo 5.  e) The set of all strings such that the 10th symbol from the right end is 1.  f) The set of all four digits numbers whose sum is 9  g) The set of all four digital numbers, whose individual digits are in ascending order from left to right. | 8 - 10 |

**Part-B: Part-B: Implementation of Parsers (Syntax Analyzers) Using C/C++/Java/Python language)**

|  |  |  |
| --- | --- | --- |
| **S. No.** | **Program Details** | **Page No.** |
| 1 | Write a program to implement  (a) Recursive Descent Parsing with back tracking (Brute Force Method). S→ cAd , A →ab /a  (b) Recursive Descent Parsing with back tracking (Brute Force Method). S→ cAd , A → a / ab | 11 - 16 |
| 2 | 2. Write a program to implement: Recursive Descent Parsing with back tracking (Brute Force Method).  (a) S→ aaSaa | aa  (b) S → aaaSaaa | aa  (c) S → aaaaSaaaa | aa  (d) S → aaaSaaa |aSa | aa | 17 - 24 |

**Part-C: Syntax Directed Translation using YACC tool**

|  |  |  |
| --- | --- | --- |
| **S. No.** | **Program Details** | **Page No.** |
| 1 | Write a program to design LALR parsing using YACC. | 25 - 26 |
| 2 | Use YACC to Convert Binary to Decimal (including fractional numbers) | 27 - 29 |
| 3 | Use YACC to implement, evaluator for arithmetic expressions (Desktop calculator) | 30 -32 |
| 4 | Use YACC to convert: Infix expression to Postfix expression. | 33 - 35 |
| 5 | Use YACC to generate Syntax tree for a given expression | 36 - 38 |
| 6 | Use YACC to generate 3-Address code for a given expression | 39 – 41 |
| 7 | Use YACC to generate the 3-Address code which contains Arrays. | 42 - 44 |

**Practice Programs (Lex) –** 45 - 60

**Practice Programs (YACC) –** 61 - 64

**Course Outcome**

|  |  |
| --- | --- |
| CO1 | Apply the fundamental concepts for the various phases of compiler design. |
| CO2 | Analyse the syntax and semantic concepts of a compiler. |
| CO3 | Design various types of parsers and Address code generation |
| CO4 | Implement compiler principles, methodologies using lex, yacc tools |

**Part-A: Implementation of Lexical Analyzer, By using C/C++/Java/Python language and using LEX tool.**

1. **Write a program to design Lexical Analyzer in C/C++/Java/Python Language (to recognize any five keywords, identifiers, numbers, operators and punctuations)**

import re

def is\_operator(char):

return char in ['+', '-', '\*', '/', '>', '<', '=']

def is\_valid\_identifier(token):

return token[0].isalpha() and not token.isdigit()

def get\_keywords():

return ["auto", "break", "case", "char", "const", "continue", "default", "do",

"double", "else", "enum", "extern", "float", "for", "goto", "if",

"int", "long", "register", "return", "short", "signed", "sizeof", "static",

"struct", "switch", "typedef", "union", "unsigned", "void", "volatile", "while"]

def is\_integer(token):

try:

int(token)

return True

except ValueError:

return False

def lexical\_analyzer(input\_str):

tokens = re.findall(r'[a-zA-Z\_]\w\*|[-+\*/<>=]|[(),;]|[0-9]+', input\_str)

print('Tokens: ')

for token in tokens:

if token in ['+', '-', '\*', '/', '>', '<', '=']:

print(f"Operator -> {token}")

elif token in [',', ';', '(', ')']:

print(f"Delimiter -> {token}")

elif token in get\_keywords():

print(f"Keyword -> {token}")

elif is\_integer(token):

print(f"Integer -> {token}")

elif is\_valid\_identifier(token):

print(f"Identifier -> {token}")

else:

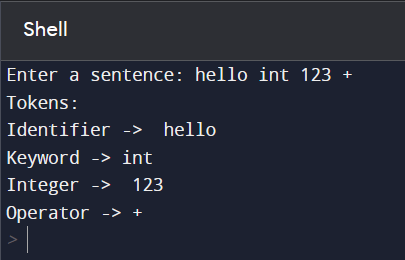
print(f"Unidentified -> {token}")

if \_\_name\_\_ == "\_\_main\_\_":

input\_string = input("Enter a C program code: ")

lexical\_analyzer(input\_string)

OUTPUT:



1. **Write a program in LEX to recognize Floating Point Numbers.**

%{

#include<stdio.h>

int cnt=0;

%}

sign [+|-]

num [0-9]

dot [.]

%%

{sign}?{num}\*{dot}{num}\* {printf("Floating point no.");cnt=1;}

{sign}?{num}\* {printf("Not Floating point no.");cnt=1;}

%%

int yywrap()

{

}

int main()

{

yylex();

if(cnt==0){

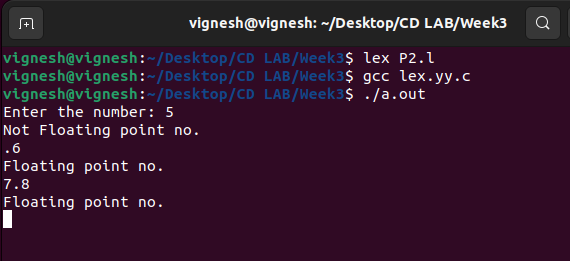
printf("Not floating pnt no.");

}

return 0;

}

OUTPUT:



1. **Write a program in LEX to recognize different tokens: Keywords, Identifiers, Constants, Operators and Punctuation symbols.**

%{

#include<stdio.h>

int cnt=0;

%}

letter [a-zA-Z]

digit [0-9]

punc [!|,|.]

oper [+|\*|-|/|%]

boole [true|false]

%%

{digit}+|{digit}\*.{digit}+ {printf("Constants");}

int|float {printf("Keyword");}

{letter}({digit}|{letter})\* {printf("Identifiers");}

{oper} {printf("Operator");}

{punc} {printf("Punctuator");}

%%

int yywrap()

{

}

int main()

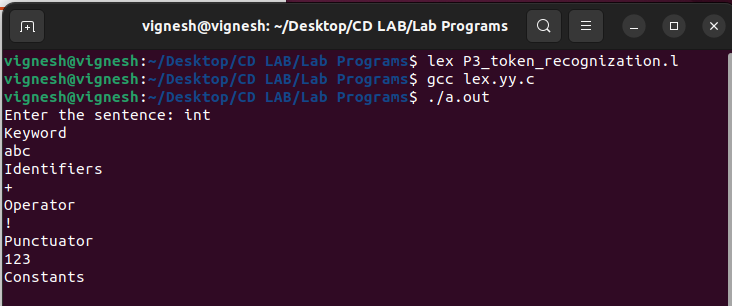
{

yylex();

return 0;

}

OUTPUT:



1. **Write a LEX program that copies a file, replacing each nonempty sequence of white spaces by a**

**single blank.**

%{

#include<stdio.h>

%}

%%

[\t" "]+ fprintf(yyout," ");

.|\n fprintf(yyout,"%s",yytext);

%%

int yywrap()

{

return 1;

}

int main(void)

{

yyin=fopen("input1.txt","r");

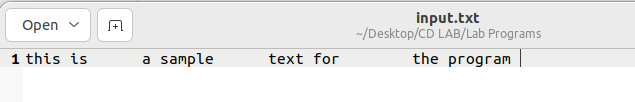
yyout=fopen("output.txt","w");

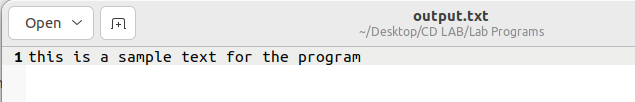
yylex();

return 0;

}

OUTPUT:





1. **Write a LEX program to recognize the following tokens over the alphabets {0,1,..,9}**
2. **The set of all string ending in 00.**
3. **The set of all strings with three consecutive 222’s.**
4. **The set of all string such that every block of five consecutive symbols contains at least two 5’s.**
5. **The set of all strings beginning with a 1 which, interpreted as the binary representation of an integer, is congruent to zero modulo 5.**
6. **The set of all strings such that the 10th symbol from the right end is 1.**
7. **The set of all four digits numbers whose sum is 9**
8. **The set of all four digital numbers, whose individual digits are in ascending order from left to right.**

d[0-9]

%{

/\* d is for recognising digits \*/

int c1=0,c2=0,c3=0,c4=0,c5=0,c6=0,c7=0;

/\* c1 to c7 are counters for rules a1 to a7 \*/

%}

%%

({d})\*00 { c1++; printf("%s -> string ending in 00\n",yytext);}

({d})\*222({d})\* { c2++; printf("%s -> string with three consecutive 222’s \n",yytext);}

(1(0)\*(11|01)(01\*01|00\*10(0)\*(11|1))\*0)(1|10(0)\*(11|01)(01\*01|00\*10(0)\*(11|1))\*10)\* {

c4++;

printf("%s -> string beginning with a 1 which, interpreted as the binary representation of an integer, is congruent to zero modulo 5 \n",yytext);

}

({d})\*1{d}{9} {

c5++; printf("%s -> string such that the 10th symbol from the right end is 1 \n",yytext);

}

({d})\* {

int i,c=0;

if(yyleng<5)

{

printf("%s doesn't match any rule\n",yytext);

}

else

{

for(i=0;i<5;i++) { if(yytext[i]=='5') {

c++; } }

if(c>=2)

{

for(;i<yyleng;i++)

{

if(yytext[i-5]=='5') {

c--; }

if(yytext[i]=='5') { c++;

}

if(c<2) { printf("%s doesn't match any rule\n",yytext);

break; }

}

if(yyleng==i)

{

printf("%s -> string such that every block of five consecutive symbols contains at least two 5’s\n",yytext); c3++; }

}

else

{

printf("%s doesn't match any rule\n",yytext);

}

}

}

%%

int yywrap()

{

}

int main()

{

printf("Enter text\n");

yylex();

printf("Total number of tokens matching rules are : \n");

printf("Rule A : %d \n",c1);

printf("Rule B : %d \n",c2);

printf("Rule C : %d \n",c3);

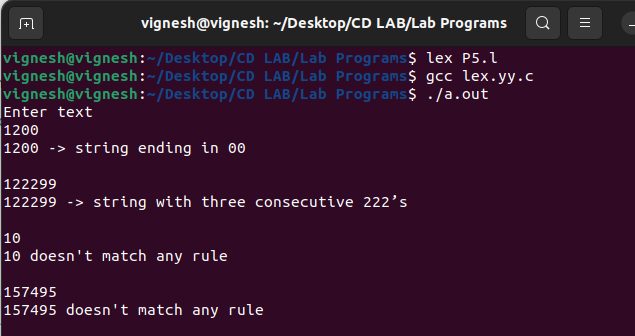
printf("Rule D : %d \n",c4);

printf("Rule E : %d \n",c5);

return 0;

}

OUTPUT:



**Part-B: Part-B: Implementation of Parsers (Syntax Analyzers) Using C/C++/Java/Python language)**

1. **Write a program to implement**
2. **Recursive Descent Parsing with back tracking (Brute Force Method). S→ cAd , A →ab /a**

#include<stdio.h>

#include<string.h>

int S();

int A();

char input[100];

int currentIndex = 0;

int match(char symbol) {

if (input[currentIndex] == symbol) {

currentIndex++;

return 1;

} else {

return 0;

}

}

int S() {

if (match('c')) {

if (A()) {

if (match('d')) {

return 1;

}

}

}

return 0;

}

int A() {

int tempIndex = currentIndex;

if (match('a')) {

if (match('b')) {

return 1;

}

}

currentIndex = tempIndex;

if (match('a')) {

return 1;

}

return 0;

}

int main() {

printf("Enter the input string: ");

scanf("%s", input);

currentIndex = 0;

if (S() && currentIndex == strlen(input)) {

printf("Parsing successful! Input belongs to the given grammar.\n");

} else {

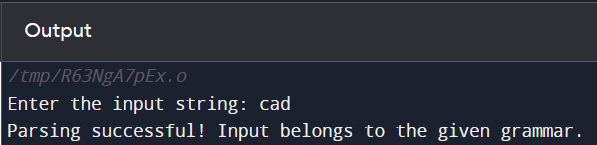
printf("Parsing failed! Input does not belong to the given grammar.\n");

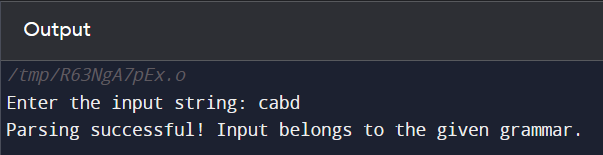
}

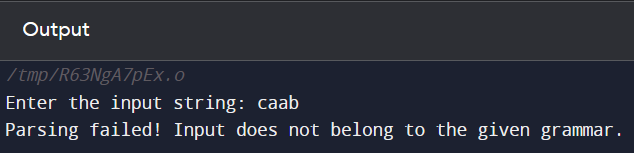
return 0;

}

OUTPUT:







**(b) Recursive Descent Parsing with back tracking (Brute Force Method). S→ cAd , A → a / ab**

#include<stdio.h>

#include<string.h>

int S();

int A();

char input[100];

int currentIndex = 0;

int match(char symbol) {

if (input[currentIndex] == symbol) {

currentIndex++;

return 1;

} else {

return 0;

}

}

int S() {

if (match('c')) {

if (A()) {

if (match('d')) {

return 1;

}

}

}

return 0;

}

int A() {

int tempIndex = currentIndex;

if (match('a')) {

return 1;

}

currentIndex = tempIndex;

if (match('a')) {

if (match('b')) {

return 1;

}

}

currentIndex = tempIndex;

return 0;

}

int main() {

printf("Enter the input string: ");

scanf("%s", input);

currentIndex = 0;

if (S() && currentIndex == strlen(input)) {

printf("Parsing successful! Input belongs to the given grammar.\n");

} else {

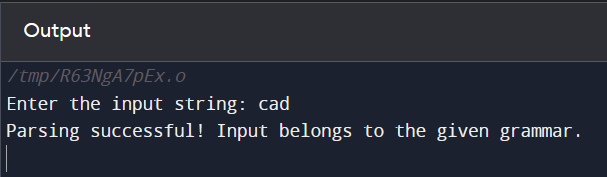
printf("Parsing failed! Input does not belong to the given grammar.\n");

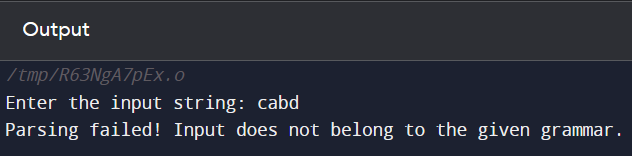
}

return 0;

}

OUTPUT:



****

1. **Write a program to implement: Recursive Descent Parsing with back tracking (Brute Force Method).**

**(a) S→ aaSaa | aa**

#include <stdio.h>

#include <string.h>

int S();

char input[100];

int currentIndex = 0;

int match(char symbol) {

if (input[currentIndex] == symbol) {

currentIndex++;

return 1;

} else {

return 0;

}

}

int S() {

int tempIndex = currentIndex;

if (match('a') && match('a')) {

if (S() && match('a') && match('a')) {

return 1;

}

}

currentIndex = tempIndex;

if (match('a') && match('a')) {

return 1;

}

return 0;

}

int main() {

printf("Enter the input string: ");

scanf("%s", input);

currentIndex = 0;

if (S() && currentIndex == strlen(input)) {

printf("Parsing successful! Input belongs to the given grammar.\n");

} else {

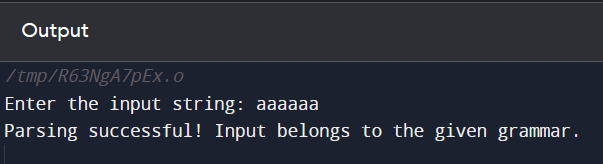
printf("Parsing failed! Input does not belong to the given grammar.\n");

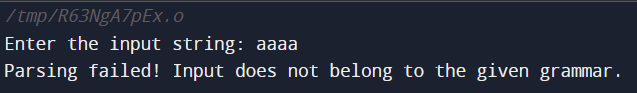
}

return 0;

}

OUTPUT:





1. **S → aaaSaaa | aa**

#include <stdio.h>

#include <string.h>

int S();

char input[100];

int currentIndex = 0;

int match(char symbol) {

if (input[currentIndex] == symbol) {

currentIndex++;

return 1;

} else {

return 0;

}

}

int S() {

int tempIndex = currentIndex;

if (match('a') && match('a') && match('a')) {

if (S() && match('a') && match('a') && match('a')) {

return 1;

}

}

currentIndex = tempIndex;

if (match('a') && match('a')) {

return 1;

}

return 0;

}

int main() {

printf("Enter the input string: ");

scanf("%s", input);

currentIndex = 0;

if (S() && currentIndex == strlen(input)) {

printf("Parsing successful! Input belongs to the given grammar.\n");

} else {

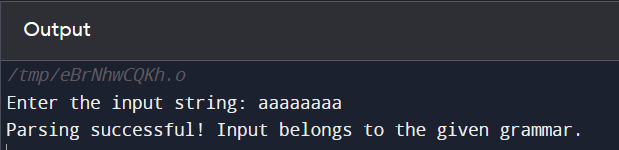
printf("Parsing failed! Input does not belong to the given grammar.\n");

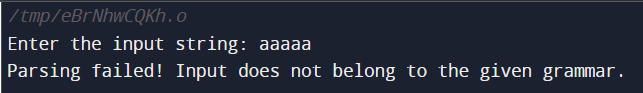
}

return 0;

}

OUTPUT:





1. **S → aaaaSaaaa | aa**

#include <stdio.h>

#include <string.h>

int S();

char input[100];

int currentIndex = 0;

int match(char symbol) {

if (input[currentIndex] == symbol) {

currentIndex++;

return 1;

} else {

return 0;

}

}

int S() {

int tempIndex = currentIndex;

if (match('a') && match('a') && match('a') && match('a')) {

if (S() && match('a') && match('a') && match('a') && match('a')) {

return 1;

}

}

currentIndex = tempIndex;

if (match('a') && match('a')) {

return 1;

}

return 0;

}

int main() {

printf("Enter the input string: ");

scanf("%s", input);

currentIndex = 0;

if (S() && currentIndex == strlen(input)) {

printf("Parsing successful! Input belongs to the given grammar.\n");

} else {

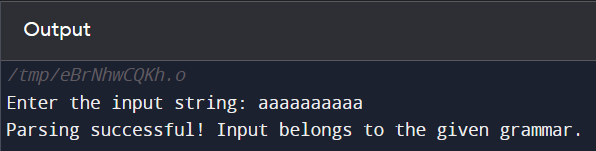
printf("Parsing failed! Input does not belong to the given grammar.\n");

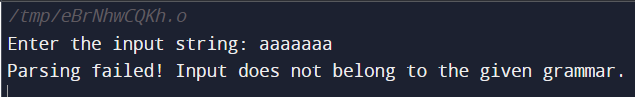
}

return 0;

}

OUTPUT:





1. **S → aaaSaaa |aSa | aa**

#include <stdio.h>

#include <string.h>

int S();

char input[100];

int currentIndex = 0;

int match(char symbol) {

if (input[currentIndex] == symbol) {

currentIndex++;

return 1;

} else {

return 0;

}

}

int S() {

int tempIndex = currentIndex;

if (match('a') && match('a') && match('a')) {

if (S() && match('a') && match('a') && match('a')) {

return 1;

}

}

currentIndex = tempIndex;

if (match('a') && S() && match('a')) {

return 1;

}

currentIndex = tempIndex;

if (match('a') && match('a')) {

return 1;

}

return 0;

}

int main() {

printf("Enter the input string: ");

scanf("%s", input);

currentIndex = 0;

if (S() && currentIndex == strlen(input)) {

printf("Parsing successful! Input belongs to the given grammar.\n");

} else {

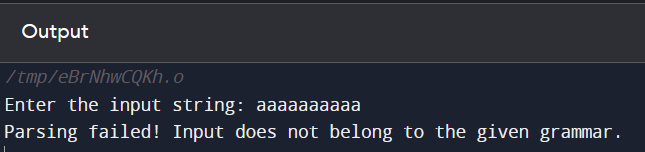
printf("Parsing failed! Input does not belong to the given grammar.\n");

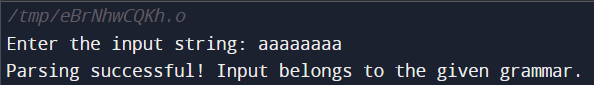
}

return 0;

}

OUTPUT:

****

****

**Part-C: Syntax Directed Translation using YACC tool**

1. **Write a program to design LALR parsing using YACC.**

**Lex:**

%{

#include "y.tab.h"

extern int yylval;

%}

%%

//If the token is an Integer number,then return it's value.

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

//If the token is space or tab,then just ignore it.

[\t] ;

//If the token is new line,return 0.

[\n] return 0;

//For any other token, return the first character read since the last

match.

. return yytext[0];

%%

**Yacc:**

%{

#include <math.h>

#include<ctype.h>

#include<stdio.h>

int var\_cnt=0;

char iden[20];

%}

%token id

%token digit

%%

S:id '=' E { printf("%s=t%d\n",iden,var\_cnt-1); }

E:E '+' T { $$=var\_cnt; var\_cnt++; printf("t%d = t%d + t%d;\n", $$, $1, $3 ); }

|E '-' T { $$=var\_cnt; var\_cnt++; printf("t%d = t%d - t%d;\n", $$, $1, $3 ); }

|T { $$=$1; }

;

T:T '\*' F { $$=var\_cnt; var\_cnt++; printf("t%d = t%d \* t%d;\n", $$, $1, $3 ); } |T '/' F { $$=var\_cnt; var\_cnt++; printf("t%d = t%d / t%d;\n", $$, $1, $3 ); } |F {$$=$1 ; }

F:P '^' F { $$=var\_cnt; var\_cnt++; printf("t%d = t%d ^ t%d;\n", $$, $1, $3 );} | P { $$ = $1;}

;

P: '(' E ')' { $$=$2; }

|digit { $$=var\_cnt; var\_cnt++; printf("t%d = %d;\n",$$,$1); } ;

%%

int main()

{

var\_cnt=0;

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

yyparse();

return 0;

}

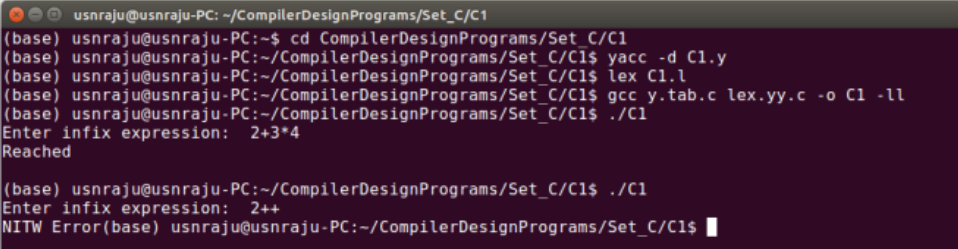
yyerror()

{

printf("error");

}

OUTPUT:



1. **Use YACC to Convert Binary to Decimal (including fractional numbers)**

**Lex:**

/\* definitions \*/

%{

// including required header files

#include<stdio.h>

#include<stdlib.h>

#include"y.tab.h"

// declaring a external variable yylval

extern int yylval;

%}

/\* rules

if 0 is matched ,make yylval to 0 and return ZERO which is

variable in Yacc program

if 1 is matched ,make yylval to 1 and return ONE which is

variable in Yacc program

if . is matched ,return POINT which is variable in Yacc program

if line change , return 0

otherwise ,ignore\*/

%%

0 {yylval=0;return ZERO;}

1 {yylval=1;return ONE;}

"." {return POINT;}

[ \t] {;}

\n return 0;

%%

**Yacc:**

/\* definition section\*/

%{

#include<stdio.h>

#include<stdlib.h>

#include<math.h>

//#define YYSTYPE double

void yyerror(char \*s);

float x = 0;

%}

// creating tokens whose values are given by lex

%token ZERO ONE POINT

// following a grammer rule which is converting binary number to

decimal number (float value)

%%

L: X POINT Y {printf("%f",$1+x);}

| X {printf("%d", $$);}

X: X B {$$=$1\*2+$2;}

| B {$$=$1;}

Y: B Y {x=$1\*0.5+x\*0.5;}

| {;}

B:ZERO {$$=$1;}

|ONE {$$=$1;};

%%

// main function

int main()

{

printf("Enter the binary number : ");

// calling yyparse function which execute grammer rules and

lex

while(yyparse());

printf("\n");

}

// if any error

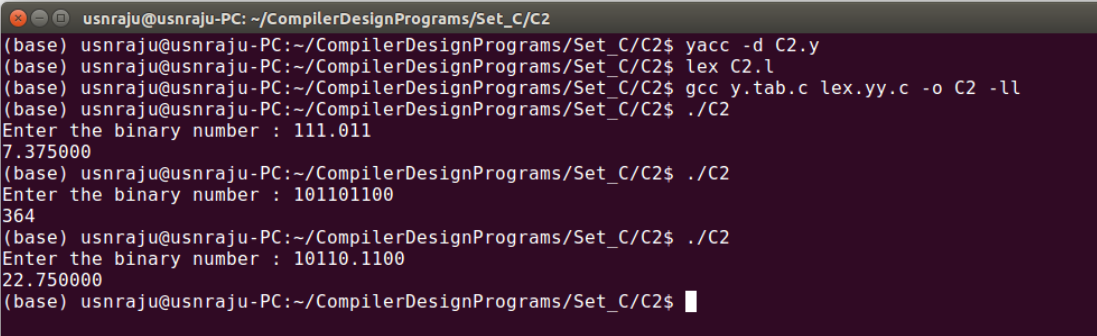
void yyerror(char \*s)

{

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

}

OUTPUT:



1. **Use YACC to implement, evaluator for arithmetic expressions (Desktop calculator)**

**Lex:**

%{

#include<stdio.h>

#include "y.tab.h"

extern int yylval;

%}

%%

[0-9]+ {

yylval=atoi(yytext);

return NUMBER;

}

[\t] ;

[\n] return 0;

. return yytext[0];

%%

int yywrap()

{

return 1;

}

**Yacc:**  
%{

#include<stdio.h>

int flag=0;

%}

%token NUMBER

%left '+' '-'

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

%left '(' ')'

/\* Rule Section \*/

%%

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;}

;

%%

//driver code

void main()

{

printf("\nEnter Any Arithmetic Expression: \n");

yyparse();

if(flag==0)

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

}

void yyerror()

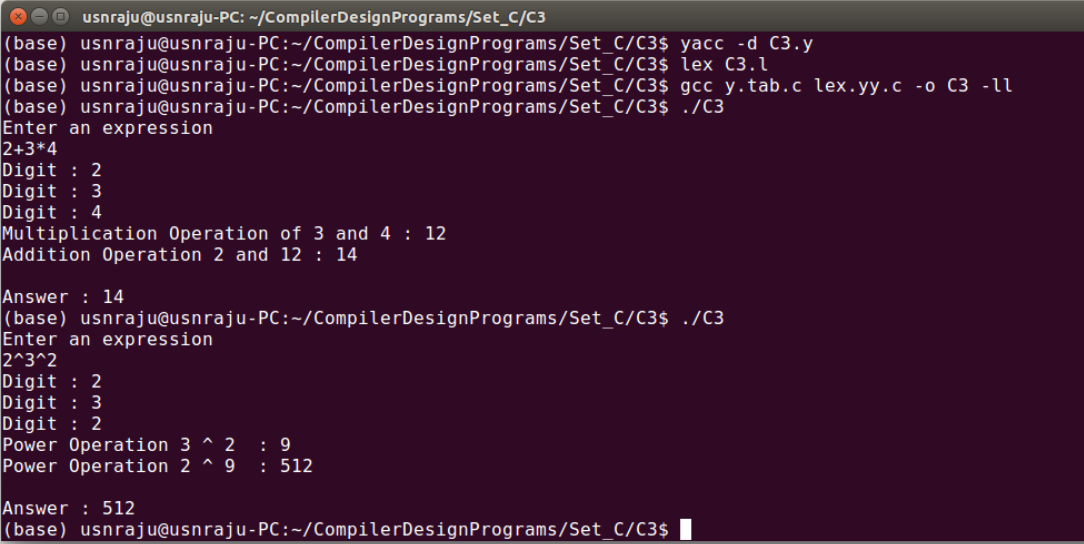
{

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

flag=1;

}

OUTPUT:



1. **Use YACC to convert: Infix expression to Postfix expression.**

**Lex:**

%{

#include "y.tab.h"

extern int yylval;

%}

%%

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

[\t] ;

[\n] return 0;

. return yytext[0];

%%

int yywrap()

{

}

**Yacc:**

%{

#include <ctype.h>

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

%}

%token digit

%%

S: E { printf("\n\n"); }

;

E: E '+' T { printf("+"); }

| E '-' T { printf("-"); }

| T

;

T: T '\*' F { printf("\*"); }

| T '/' F { printf("/"); }

| F

;

F: F '^' G { printf("^"); }

| G

;

G: '(' E ')'

| digit { printf("%d", $1); }

;

%%

int main()

{

printf("Enter infix expression: ");

yyparse();

}

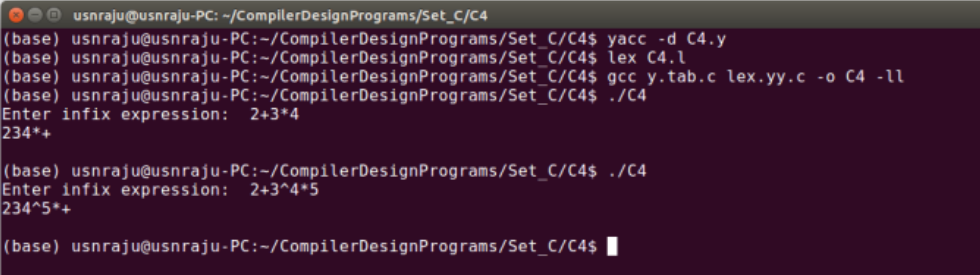
yyerror()

{

printf("Error");

}

OUTPUT:



1. **Use YACC to generate Syntax tree for a given expression**

**Lex:**

%{

#include "y.tab.h"

extern int yylval;

%}

%%

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

[\t] ;

[\n] return 0;

. return yytext[0];

%%

int yywrap()

{

}

**Yacc:**

%{

#include <math.h>

#include<ctype.h>

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

struct tree\_node

{

char val[10];

int lc;

int rc;

};

int ind;

struct tree\_node syn\_tree[100];

void my\_print\_tree(int cur\_ind);

int mknode(int lc,int rc,char val[10]);

%}

%token digit

%%

S:E { my\_print\_tree($1); }

;

E:E'+'T { $$= mknode($1,$3,"+"); ; }

|T { $$=$1; }

;

T:T'\*'F { $$= mknode($1,$3,"\*"); ; }

|F {$$=$1 ; }

;

F:'('E')' { $$=$2; }

|digit {char buf[10]; sprintf(buf,"%d", yylval); $$ = mknode(-1,-1,buf);}

%%

int main()

{

ind=0;

printf("Enter an expression\n");

yyparse();

return 0;

}

int yyerror()

{

printf("NITW Error\n");

}

int mknode(int lc,int rc,char val[10])

{

strcpy(syn\_tree[ind].val,val);

syn\_tree[ind].lc = lc;

syn\_tree[ind].rc = rc;

ind++;

return ind-1;

}

/\*my\_print\_tree function to print the syntax tree in DLR fashion\*/

void my\_print\_tree(int cur\_ind)

{

if(cur\_ind==-1) return;

if(syn\_tree[cur\_ind].lc==-1&&syn\_tree[cur\_ind].rc==-1)

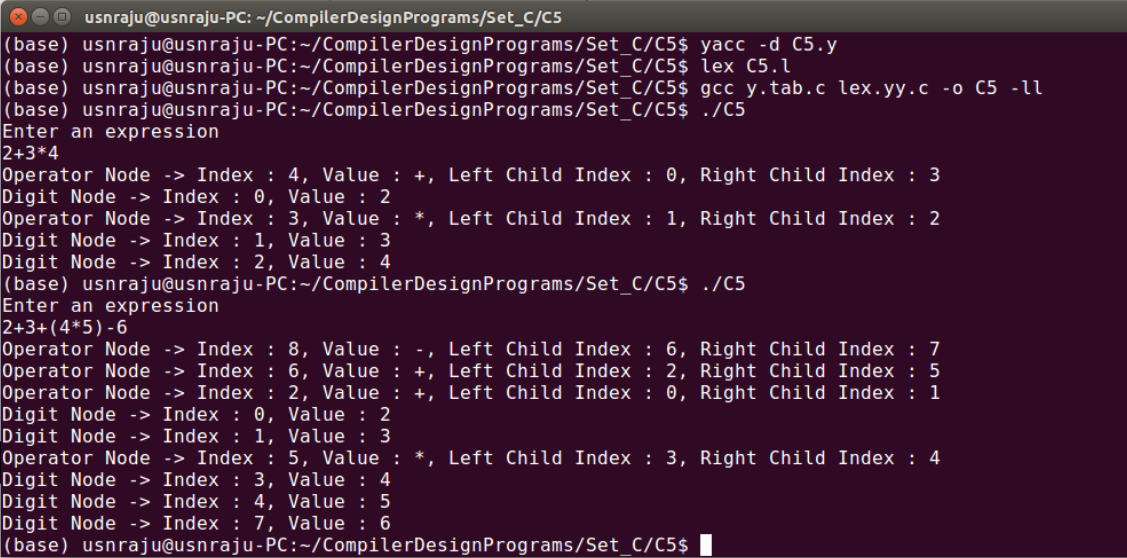
printf("Digit Node -> Index : %d, Value : %s\n",cur\_ind,syn\_tree[cur\_ind].val); else

printf("Operator Node -> Index : %d, Value : %s, Left Child Index : %d,Right Child Index : %d \n",cur\_ind,syn\_tree[cur\_ind].val, syn\_tree[cur\_ind].lc,syn\_tree[cur\_ind].rc); my\_print\_tree(syn\_tree[cur\_ind].lc);

my\_print\_tree(syn\_tree[cur\_ind].rc);

}

OUTPUT:



1. **Use YACC to generate 3-Address code for a given expression**

**Lex:**

d [0-9]+

a [a-zA-Z]+

%{

#include<stdio.h>

#include<stdlib.h>

#include"y.tab.h"

extern int yylval;

extern char iden[20];

%}

%%

{d} { yylval=atoi(yytext); return digit; }

{a} { strcpy(iden,yytext); yylval=1; return id;}

[ \t] {;}

\n return 0;

. return yytext[0];

%%

int yywrap()

{

}

**Yacc:**

%{

#include <math.h>

#include<ctype.h>

#include<stdio.h>

int var\_cnt=0;

char iden[20];

%}

%token id

%token digit

%%

S:id '=' E { printf("%s=t%d\n",iden,var\_cnt-1); }

E:E '+' T { $$=var\_cnt; var\_cnt++; printf("t%d = t%d + t%d;\n", $$, $1, $3 ); }

|E '-' T { $$=var\_cnt; var\_cnt++; printf("t%d = t%d - t%d;\n", $$, $1, $3 ); }

|T { $$=$1; }

;

T:T '\*' F { $$=var\_cnt; var\_cnt++; printf("t%d = t%d \* t%d;\n", $$, $1, $3 ); } |T '/' F { $$=var\_cnt; var\_cnt++; printf("t%d = t%d / t%d;\n", $$, $1, $3 ); } |F {$$=$1 ; }

F:P '^' F { $$=var\_cnt; var\_cnt++; printf("t%d = t%d ^ t%d;\n", $$, $1, $3 );} | P { $$ = $1;}

;

P: '(' E ')' { $$=$2; }

|digit { $$=var\_cnt; var\_cnt++; printf("t%d = %d;\n",$$,$1); } ;

%%

int main()

{

var\_cnt=0;

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

yyparse();

return 0;

}

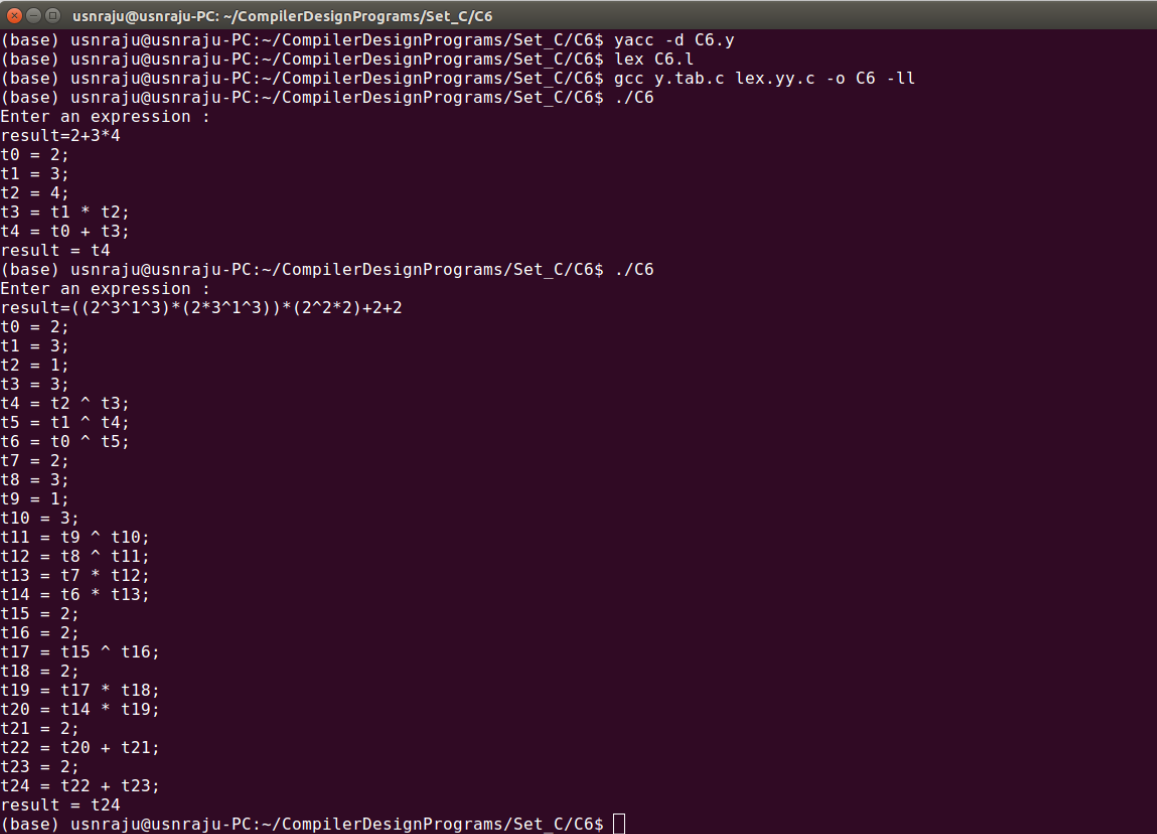
yyerror()

{

printf("error");

}

OUTPUT:



1. **Use YACC to generate the 3-Address code which contains Arrays.**

**Lex:**

%{

#include "y.tab.h"

#include <stdlib.h>

%}

d[0-9]

c[a-z]

extern char yylval;

/\*

Rules:

If an alphabet from a to z is matched, it is sent as a token.

If a tab character is encountered, nothing is done.

If a new line character is encountered, code stops running.

For anything else, the first character of the matched word is

sent as token.

\*/

%%

{c} { yylval=yytext[0]; return(id); }

[\t] ;

[\n] return 0;

. return yytext[0];

%%

**Yacc:**

/\* definitions \*/

%{

// including required header files

#include<stdio.h>

#include<stdlib.h>

#include"y.tab.h"

// declaring a external variable yylval

extern int yylval;

%}

/\* rules

if 0 is matched ,make yylval to 0 and return ZERO which is

variable in Yacc program

if 1 is matched ,make yylval to 1 and return ONE which is

variable in Yacc program

if . is matched ,return POINT which is variable in Yacc program

if line change , return 0

otherwise ,ignore\*/

%%

0 {yylval=0;return ZERO;}

1 {yylval=1;return ONE;}

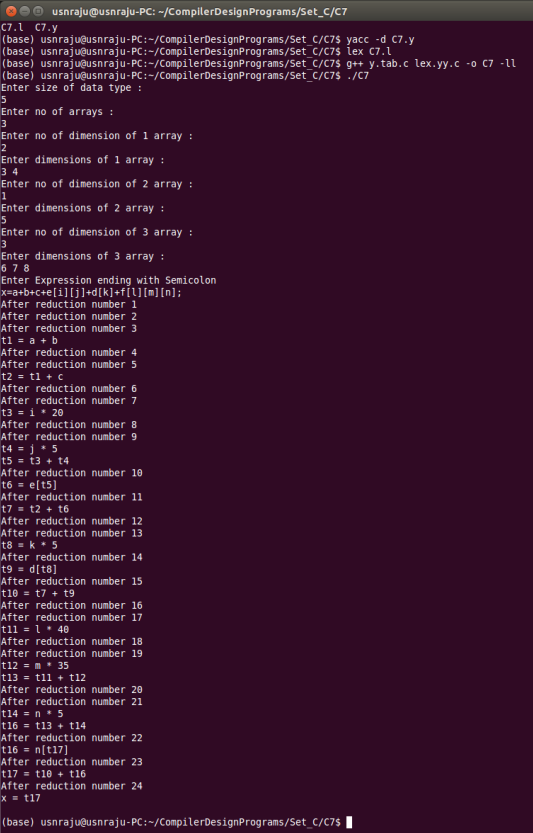
"." {return POINT;}

[ \t] {;}

\n return 0;

%%

OUTPUT:

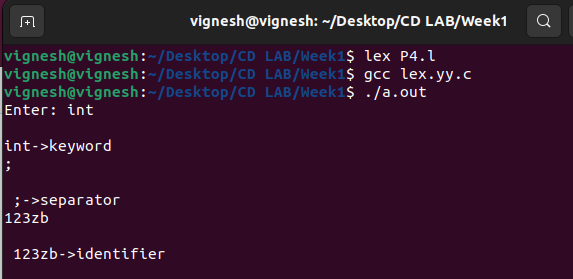


**PRACTICE PROGRAMS (Lex)**

**WEEK 1**

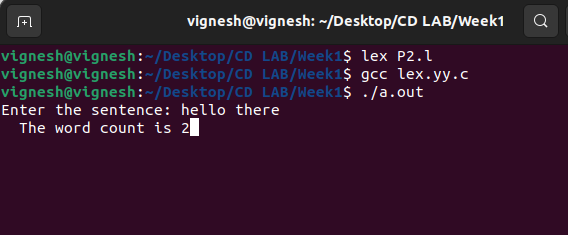
1. **Lex program to check entered character is either number or operator**

%option noyywrap  
%{  
#include<stdio.h>  
%}  
%%  
[0-9]+  {printf("number:%s\n",yytext);}  
[+-]  {printf("operator:%s\n",yytext);}  
[ \t\n]  {/\*ignore whitespaces and newline\*/}  
[a-zA-Z]\*  {printf("invalid character:%s\n",yytext);}  
%%  
  
int main()  
{  
printf("enter");  
yylex();  
return 0;  
}

OUTPUT:  


1. **Lex program to count the number of words in the sentence**

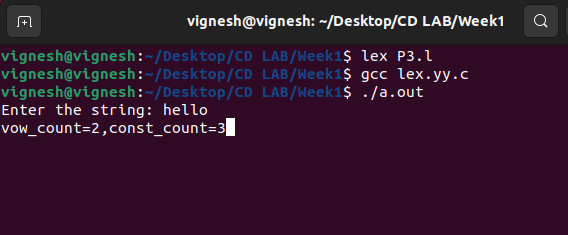
%{  
#include<stdio.h>  
int c=0;  
%}  
%%  
[a-zA-Z0-9]+  {c++;}  
\n {printf("the count is %d",c);}  
%%  
int yywrap()  
{  
}  
int main()  
{  
printf("enter the sentence");  
yylex();  
return 0;  
}

OUTPUT:  


1. **Lex program to count vowels and consonants in a sentence**

%{  
#include<stdio.h>  
int vow\_count=0;  
int const\_count=0;  
%}  
%%  
[aeiouAEIOU] {vow\_count++;}  
[a-zA-Z] {const\_count++;}  
\n {printf("vow\_count=%d,const\_count=%d",vow\_count,const\_count);}  
%%  
int yywrap()  
{  
}  
int main()  
{  
printf("enter the string of vowels and consonants:");  
yylex();  
return 0;  
}

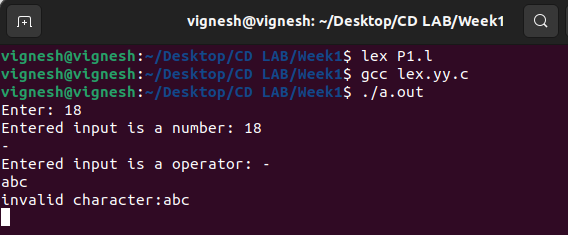
OUTPUT:



1. **Lex program to check the type of entered word**

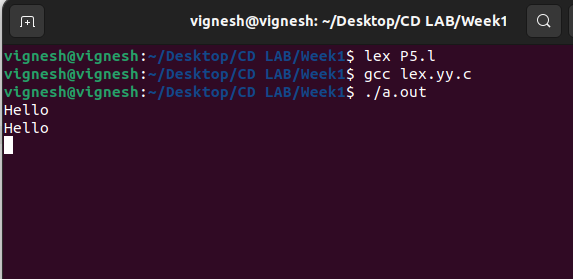
%option noyywrap  
%{  
#include<stdio.h>  
%}  
%%  
int|char|float {printf("\n%s->keyword",yytext);}  
,|; {printf("\n %s->separator",yytext);}  
[a-zA-Z0-9]\* {printf("\n %s->identifier",yytext);}  
%%  
int wrap()  
{  
}  
int main()  
{  
printf("enter");  
yylex();  
return 0;  
}

OUTPUT:



1. **Lex program to print the input as it is**

%%  
. ECHO;  
%%  
int yywrap(void)  
{  
}  
int main(void)  
{  
yylex();  
return 0;  
}

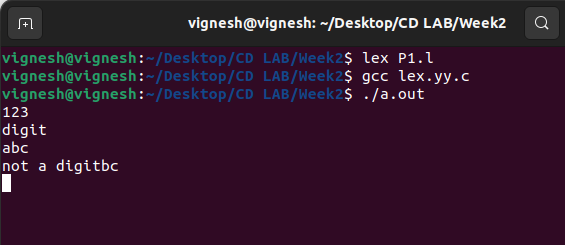
OUTPUT:  


**WEEK 2**

1. **Write a lex program to check whether input is digit or not**

%{

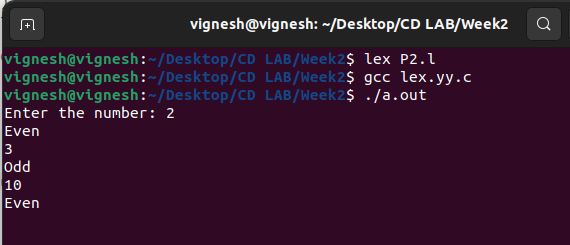
#include<stdio.h>  
#include<stdlib.h>  
%}  
%%  
^[0-9]\* printf("digit");  
^[^0-9]|[0-9]\*[a-zA-Z] printf("not a digit");  
.;  
%%  
int yywrap()  
{  
}  
int main()  
{  
yylex();  
return 0;  
}

OUTPUT:  


1. **Write a lex program to check whether the given number is even or odd.**

%{  
#include<stdio.h>  
int i;  
%}  
   
%%  
   
[0-9]+     {i=atoi(yytext);  
          if(i%2==0)    
               printf("Even");  
          else  
         printf("Odd");}  
%%  
     
int yywrap(){}  
    
int main()  
{  
     
    yylex();  
    return 0;  
}

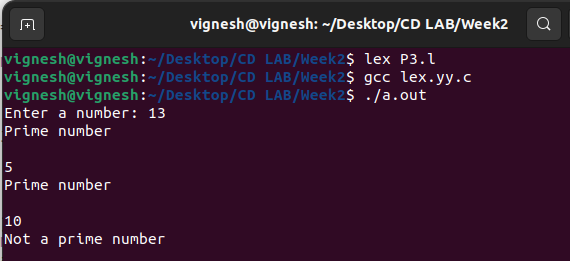
OUTPUT:



1. **Write a lex program to check whether a number is Prime or not.**

%{  
     
   #include<stdio.h>  
   #include<stdlib.h>  
   int flag,c,j;  
%}  
   
%%  
[0-9]+ {c=atoi(yytext);  
         if(c==2)  
         {  
           printf("\n Prime number");  
         }  
         else if(c==0 || c==1)  
         {  
           printf("\n Not a Prime number");  
         }  
         else  
         {  
           for(j=2;j<c;j++)  
         {    
         if(c%j==0)  
           flag=1;  
         }  
         if(flag==1)  
           printf("\n Not a prime number");  
         else if(flag==0)  
           printf("\n Prime number");  
         }  
       }  
%%  
int yywrap()  
{  
}  
   
int main()  
 {  
  yylex();  
  return 0;  
 }

OUTPUT:



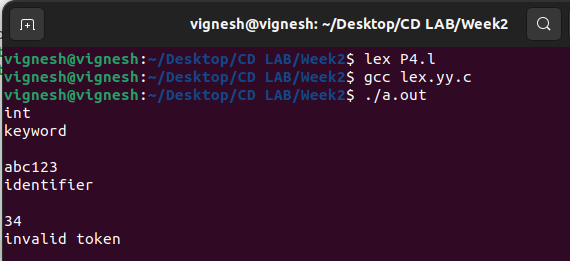
1. **Write a lex program to recognize a) identifiers**

**b) keyword-int and float**

**c) anything else as invalid tokens.**

%{  
     
   #include<stdio.h>  
%}  
alpha[a-zA-Z]  
digit[0-9]  
%%  
(float|int) {printf("\nkeyword");}  
{alpha}({digit}|{alpha})\* {printf("\nidentifier");}  
{digit}({digit}|{alpha})\* {printf("\ninvalid token");}  
%%  
int yywrap()  
{  
}  
int main()  
{  
yylex();  
return 0;  
}

OUTPUT:



1. **Write a lex program to identify a) identifiers**

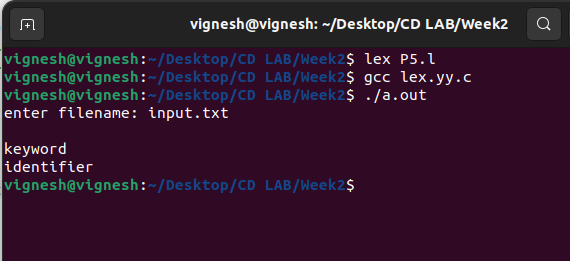
**b) keyword-int and float**

**c) anything else as invalid tokens**

**Read these from a text file.**

%{  
     
   #include<stdio.h>  
   char fname[25];  
%}  
alpha[a-zA-Z]  
digit[0-9]  
%%  
(float|int) {printf("\nkeyword");}  
{alpha}({digit}|{alpha})\* {printf("\nidentifier");}  
{digit}({digit}|{alpha})\* {printf("\ninvalid token");}  
%%  
int yywrap()  
{  
}  
int main()  
{  
printf("enter filename");  
scanf("%s",fname);  
yyin=fopen(fname,"r");  
yylex();  
return 0;  
fclose(yyin);  
}

OUTPUT:

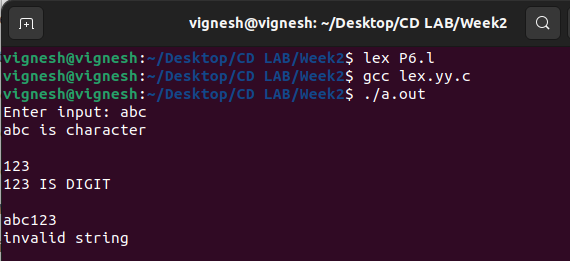


1. **Write a Program to print invalid string if a Alpha-Numeric string is entered as input.**

%{

#include<stdio.h>  
%}  
alpha [a-zA-Z0-9]\*  
%%  
[0-9]\* {printf("%s IS DIGIT",yytext);}  
[a-zA-Z]\* {printf("\n%s is character",yytext);}  
{alpha} {printf("invalid string");}  
%%  
int yywrap()  
{  
}  
int main()  
{  
printf("enter input");  
yylex();  
return 0;

}

OUTPUT:  


**WEEK 3**

1. **Lex program to count the number of comment lines (multi line comments or single line) in a program. Read the input from a file called input.txt and print the count in a file called output.txt**

%{

#include <stdio.h>

int cc=0;

%}

%x CMNT

%%

"/\*" {BEGIN CMNT;}

<CMNT>. ;

<CMNT>"\*/" {BEGIN 0; cc++;}

%%

int yywrap() { }

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

{

if(argc!=3)

{

printf("Usage : %s <scr\_file> <dest\_file>\n",argv[0]);

return 0;

}

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

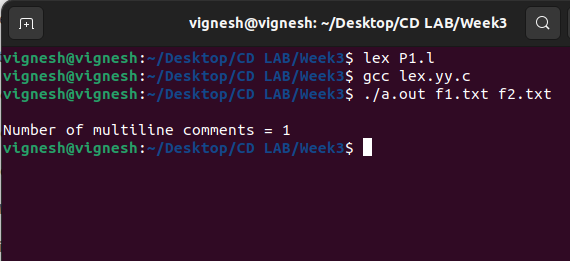
yyout=fopen(argv[2],"w");

yylex();

printf("\nNumber of multiline comments = %d\n",cc);

return 0;

}

OUTPUT:  


1. **Write a program in LEX to recognize Floating Point Numbers.**

%{

#include<stdio.h>

int cnt=0;

%}

sign [+|-]

num [0-9]

dot [.]

%%

{sign}?{num}\*{dot}{num}\* {printf("Floating point no.");cnt=1;}

{sign}?{num}\* {printf("Not Floating point no.");cnt=1;}

%%

int yywrap()

{

}

int main()

{

yylex();

if(cnt==0){

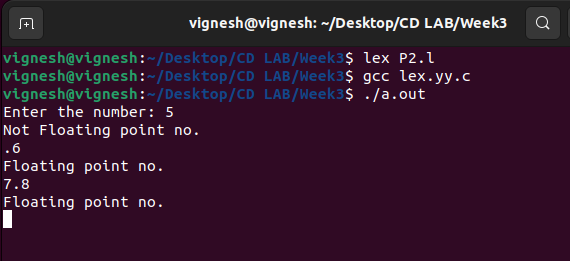
printf("Not floating pnt no.");

}

return 0;

}

OUTPUT:



1. **Write a program to read and check if the user entered number is signed or unsigned using appropriate meta character**

%{

#include<stdio.h>

int cnt=0;

%}

sign [+|-]

num [0-9]

dot [.]

%%

{sign}{num}\*{dot}\*{num}\* {printf("Signed no.");cnt=1;}

{num}\*{dot}\*{num}\* {printf("Unsigned no.");cnt=1;}

%%

int yywrap()

{

}

int main()

{

yylex();

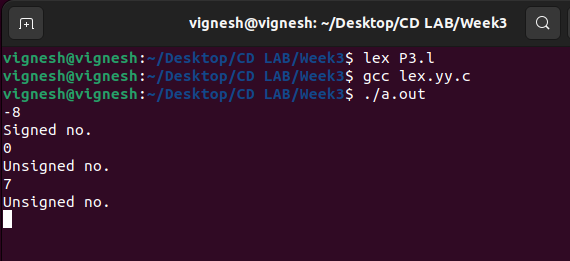
if(cnt==0){

printf("Not floating pnt no.");

}

return 0;

}

OUTPUT:  


1. **Write a program to check if the input sentence ends with any of the following punctuationmarks ( ? , fullstop , ! )**

%{

#include<stdio.h>

int cnt=0;

%}

punc [?|,|.|!]

chars [a-z|A-Z|0-9|" "|\t]

%%

{chars}\*{punc} {printf("Sentence ends with punc");}

{chars}\* {printf("Sentence does not end with punc");}

%%

int yywrap()

{

}

int main()

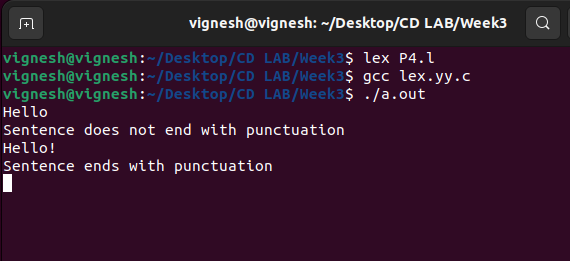
{

yylex();

return 0;

}

OUTPUT:



1. **Write a program to read an input sentence and to check if the sentence begins with English articles (A, a,AN,An,THE and The). If the sentence starts with the article appropriate message should be printed. If the sentence does not start with the article appropriate message** should be printed

%{

#include<stdio.h>

int cnt=0;

%}

chars [a-z|A-Z|0-9]

check [A|a|AN|An|THE|The]

%%

{check}+{chars}\* {printf("Begins with %s",yytext);}

{chars}\* {printf("The sentence does not begins with articles");}

%%

int yywrap()

{

}

int main()

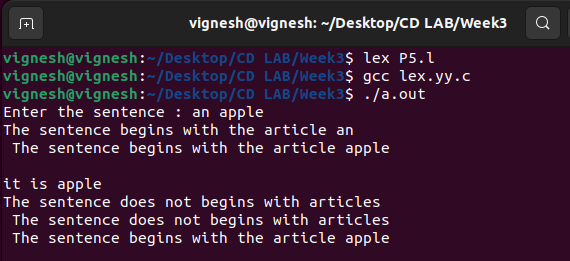
{

printf("Enter the sentence : ");}

yylex();

return 0;

}

OUTPUT:  


**PRACTICE PROGRAMS (YACC)**

**WEEK 6**

1. **Design a suitable grammar for evaluation of arithmetic expression having + and – operators.**

**+ has least priority and it is left associative**

**- has higher priority and is right associative**

**Lex:**

%{

#include "y.tab.h"

%}

%%

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

[\t] ;

\n return 0;

. return yytext[0];

%%

int yywrap()

{

}

**Yacc:**

%{

#include<stdio.h>

%}

%token NUM

%left '+'

%right '-'

%%

expr:e {printf("Valid Expression\n"); printf ("Result: %d\n",$$); return 0;}

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

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

| NUM {$$=$1;}

;

%%

int main()

{

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

yyparse();

return 0;

}

int yyerror()

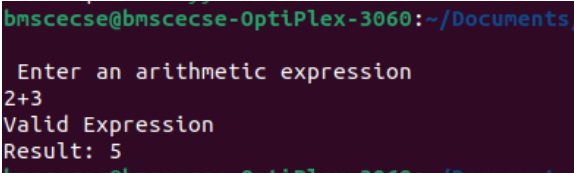
{

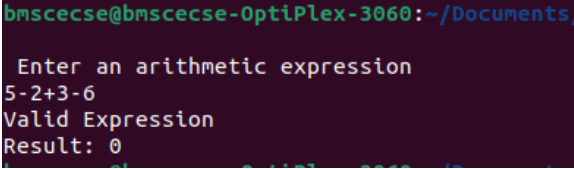
printf("\nInvalid expression\n");

return 0;

}

OUTPUT:





1. **Design a suitable grammar for evaluation of arithmetic expression having + , –, \*, /, %, ^ operators.**

**^ having highest priority and right associative**

**% having second highest priority and left associative**

**\*, / have third highest priority and left associative**

**+, - having least priority and left associative**

**Lex:**

%{

#include "y.tab.h"

%}

%%

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

[\t] ;

\n return 0;

. return yytext[0];

%%

int yywrap()

{

}

**Yacc:**

%{

#include<stdio.h>

%}

%token NUM

%left '+' '-'

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

%right '^'

%%

expr: e { printf("Valid expression\n"); printf("Result: %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 '^' e {

int result = 1;

for (int i = 0; i < $3; i++) {

result \*= $1;

}

$$ = result;

}

| NUM {$$ = $1;}

;

%%

int main()

{

printf("\nEnter an arithmetic expression:\n");

yyparse();

return 0;

}

int yyerror()

{

printf("\nInvalid expression\n");

return 0;

}

OUTPUT:

