**EXP2:**

**Write a program to count number of characters, words, sentences, lines, tabs, numbers and blank spaces present in input using LEX.**

**Code:**

Jay Vora TE3 - 77

%{

int ch=0, bl=0, ln=0, wr=0

%}

%%

[/n]{ln++;wr++}

[\t]{bl++;wr++}

[“”]{bl++;wr++}

[^\n\t]{ch++;}

%%

Int main()

{

FILE \*fp;

Char file[10];

Printf(“Enter file name”);

Scanf(“%s”,file);

Yyin=fp;

Yylex();

Printf(“character=%d \n Blank=%d \n Lines=%d \n Words=%d”, ch, bl, ln, wr); Return 0;

}

**Output:**

$lex prog.l

$cc lex.yy.c –ll

$a.out

Enter File Name: sample

Character=16

Blank=2

Lines= 2

Words = 3

**EXP3A:** **Jay Vora TE3 - 77**

**Compiler: Scanner: Write a program to recognize identifier and operators using LEX.**

**Code:**

%{

#include <stdio.h>

%}

%%

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

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

[0-9]\* {printf("number");}

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

%%

main()

{

yylex();

return 0;

}

int yywrap()

{

}

**Output:**

# lex prog.l

# gcc -lfl lex.yy.c

# ./a.out

if

keyword

+

invalid

1

number

abc

identifier

while

keyword

^C

**EXP3A: Jay Vora TE3 - 77**

**Write a Program to implement Lexical Analyzer.**

**Code:**

#Declaration for all types of token list

spe=["#","<",">",";","@","\_",","]

oper=['+','-','\*','/','%','=','!',"++"]

key=["int","float","char","double","bool","void","extern","unsigned","goto","static","class","struct","for", "if","else","return","register","long","while","do","printf","scanf"]

predirect=["include","define"]

header=["stdio.h","conio.h","malloc.h","process.h","string.h","ctype.h"]

bracket=["(",")","[","]","{","}"]

quote=["'"]

m=input("Enter file name:")

f=open(m,"r")

m\_lines=0

for line in f:

words = line.split()

m\_lines += 1

print("\nLINE",m\_lines)

token=0

for i in words:

if(i in spe):

print(i,"is special character")

token =token+ 1

elif(i in oper):

print(i,"is Operator")

token =token+ 1

elif(i in quote):

print(i,"is Quote")

token =token+ 1

elif(i in key):

print(i,"is keyword")

token =token+ 1

elif(i in predirect):

print(i,"is Pre-Processor")

token =token+ 1

elif(i in header):

print(i,"is Header")

token =token+ 1

elif(i in bracket):

print(i,"is Bracket")

token =token+ 1

elif((i>='a' and i<= 'z') or (i>='A' and i<='Z')):

print(i, "is an identifier")

token =token+ 1

elif(i>='0' and i<= '9'):

print(i, "is number")

token =token+ 1

print("Number of tokens are",token)

**Input:**

# include < stdio.h >

# include < conio.h >

void main ( )

{

int a = 5 ;

a ++ ;

printf ( ' Value of a is ' , a ) ;

}

**Output:**

Enter file name:qwe.txt

LINE 1

# is special character

include is Pre-Processor

< is special character

stdio.h is Header

> is special character

Number of tokens are 5

LINE 2

# is special character

include is Pre-Processor

Jay Vora TE3 - 77

< is special character

conio.h is Header

> is special character

Number of tokens are 5

LINE 3

void is keyword

main is an identifier

( is Bracket

) is Bracket

Number of tokens are 4

LINE 4

{ is Bracket

Number of tokens are 1

LINE 5

int is keyword

a is an identifier

= is Operator

5 is number

; is special character

Number of tokens are 5

LINE 6

a is an identifier

++ is Operator

; is special character

Number of tokens are 3

LINE 7

printf is keyword

( is Bracket

' is Quote

Value is an identifier

of is an identifier

a is an identifier

is is an identifier

' is Quote

, is special character

a is an identifier

) is Bracket

; is special character

Number of tokens are 12

LINE 8

} is Bracket

Number of tokens are 1

**EXP4:**

**Assembler: Write a Program to Implement Pass-I of 2 pass Assembler**

**Code:**

import re

class MOT:

def \_\_init\_\_(self,mnemonic,binaryop,insLength,insFormat):

self.mnemonic = mnemonic

self.binaryop = binaryop

Jay Vora TE3 - 77

self.insLength = insLength

self.insFormat = insFormat

Mlist = []

Mlist1 = []

Mlist.append(MOT('L','58','10','001'))

Mlist.append(MOT('A','5A','10','001'))

Mlist.append(MOT('ST','50','10','001'))

class POT:

def \_\_init\_\_(self,psop,address):

self.psop = psop

self.address = address

Plist = []

Plist1 = []

Plist.append(POT('START','P1START'))

Plist.append(POT('USING','P1USING'))

Plist.append(POT('DC','P1DC'))

Plist.append(POT('DS','P1DS'))

Plist.append(POT('END','P1END'))

class ST:

def \_\_init\_\_(self,symbol,value,length,relocation):

self.symbol = symbol

self.value = value

self.length = length

self.relocation = relocation

STlist=[]

def remove\_values\_from\_list(the\_list, val):

return [value for value in the\_list if value != val]

f = open("inputToassembler.txt", "rt")

addr = 0

for line in f:

s = re.split(" |\t|\n",line)

s=remove\_values\_from\_list(s,"")

print(s)

if(len(s) == 2):

operands = s[1].split(',')

if(s[0]!='USING'):

addr+=4

for item in Plist:

if(item.psop == s[0]):

Plist1.append(POT(s[0],item.address))

for item in Mlist:

if(item.mnemonic == s[0]):

Mlist1.append(MOT(s[0],item.binaryop,item.insLength,item.insFormat))

else:

if(len(s)==3):

if(s[1]=='START'):

Jay Vora TE3 - 77

STlist.append(ST(s[0],str(hex(addr)),0,'R'))

else:

STlist.append(ST(s[0],str(hex(addr)),4,'R'))

if(s[1]!='START'):

addr+=4

for i in range(len(s)):

for item in Plist:

if(item.psop == s[i]):

Plist1.append(POT(s[i],item.address))

for item in Mlist:

if(item.mnemonic == s[i]):

Mlist1.append(MOT(s[i],item.binaryop,item.insLength,item.insFormat))

print("SYMBOL TABLE")

for item in STlist:

print(item.symbol+"\t\t"+item.value+"\t\t"+str(item.length)+"\t\t"+item.relocation+"\t\t")

print("PSEUDO OPERATION TABLE")

for item in Plist1:

print(item.psop+"\t\t"+item.address)

print("MACHINE OPERATION TABLE")

for item in Mlist1:

print(item.mnemonic+"\t\t"+item.binaryop+"\t\t"+item.insLength+"\t\t"+item.insFormat)

**Input:**

JOHN START 0

USING \*,15

L 1,FIVE

A 1,FOUR

ST 1,TEMP

FOUR DC F'4'

FIVE DC F'5'

TEMP DS 1F

END

**Output:**

['JOHN', 'START', '0']

['USING', '\*,15']

['L', '1,FIVE']

['A', '1,FOUR']

['ST', '1,TEMP']

['FOUR', 'Dc', "F'4'"]

['FIVE', 'DC', "F'5'"]

['TEMP', 'DS', '1F']

['END']

SYMBOL TABLE

JOHN 0x0 0 R

FOUR 0xc 4 R

FIVE 0x10 4 R

TEMP 0x14 4 R

Jay Vora TE3 - 77

PSEUDO OPERATION TABLE

START P1START

USING P1USING

DC P1DC

DS P1DS

END P1END

MACHINE OPERATION TABLE

L 58 10 001

A 5A 10 001

ST 50 10 001

**EXP5:**

**Macro Processor: Write a program to generate Macro Name Table for given code**

**Code:**

class MDT():

def \_\_init\_\_(self,index,card):

self.index = index

self.card = card

def \_\_repr\_\_(self):

return ""+str(self.index)+"\t"+self.card

class MNT():

def \_\_init\_\_(self,index,card,mdtindex):

self.index = index

self.card = card

self.mdtindex = mdtindex

def \_\_repr\_\_(self):

return ""+str(self.index)+"\t"+self.card+"\t"+str(self.mdtindex)

class ALA():

def \_\_init\_\_(self,index\_marker,args):

self.index\_marker = index\_marker

self.args = args

def \_\_repr\_\_(self):

return ""+str(self.index\_marker)+"\t\t"+self.args

def remove\_values\_from\_list(the\_list, val):

return [value for value in the\_list if value != val]

import re

indexmnt = 0

MNT\_list = []

ALA\_list = []

indexala = 0

MDT\_list = []

done = False

line\_list = []

if \_\_name\_\_ == '\_\_main\_\_':

f = open("MACRO.txt", "rt")

addr = 0

Jay Vora TE3 - 77

index = 0

for line in f:

s = re.split(" |\t|\n",line)

s=remove\_values\_from\_list(s,"")

line\_list.append(s)

args\_list = []

for i,line in enumerate(line\_list):

wordString = ""

for word in line:

if("&" in word):

if("," not in word and (word not in args\_list)):

ALA\_list.append(ALA(indexala,word))

indexala+=1

args\_list.append(word)

wordString+=word+" "

if(word == 'PROG'):

done = True

break

if done == True:

break

if "MACRO" not in wordString:

MDT\_list.append(MDT(index,wordString))

index = index+1

if "MACRO" in line\_list[i-1]:

MNT\_list.append(MNT(indexmnt,line\_list[i][0],index-1))

indexmnt+=1

print("MDT Table")

print("index\tcard")

print(\*MDT\_list,sep="\n")

print()

print("MNT Table")

print("index\tcard\tmdtindx")

print(\*MNT\_list,sep ="\n")

print()

print("ALA Table")

print("index\_marker\tArguments")

print(\*ALA\_list,sep ="\n")

**Input:**

MACRO

XYZ &a

ST 1,&a

MEND

MACRO

MIT &z

MACRO

&z &w

AR 4,&w

XYZ ALL

MEND

ST &w,ALL

Jay Vora TE3 - 77

MEND

PROG START

USING \*,15

MIT HELLO

ST 2,3

HELLO YALE

YALE EQU 5

ALL DC f'3'

END

**Output:**

MDT Table

index card

0 XYZ &a

1 ST 1,&a

2 MEND

3 MIT &z

4 &z &w

5 AR 4,&w

6 XYZ ALL

7 MEND

8 ST &w,ALL

9 MEND

MNT Table

index card mdtindx

0 XYZ 0

1 MIT 3

2 &z 4

ALA Table

index\_marker Arguments

0 &a

1 &z

2 &w

**EXP6:**

**Compiler-Parser: Top Down Parser. Write a Program to remove Left Recursion from given context free Grammar**

**Code:**

count=0

temp = ['X','Y', 'Z']

n=int(input("Enter number of productions to be entered:"))

print (n)

while(count < n):

left=input("Enter left hand side of production" + str(count))

right=input("Enter left hand side of production" + left)

if (right[0] == left):

print("Left Recursion Present")

Jay Vora TE3 - 77

str1 = right.split('|')

alpha = str1[0]

beta = str1[1]

print(beta + temp[count])

print(alpha[1:] + temp[count] + '|9')

else:

print("Left Recursion Not Present")

count=count+1

**Output:**

Enter number of productions to be entered:3

3

Enter left hand side of production0E

Enter left hand side of productionEE+T|T

Left Recursion Present

TX

+TX|9

Enter left hand side of production1T

Enter left hand side of productionTT\*F|F

Left Recursion Present

FY

\*FY|9

Enter left hand side of production2F

Enter left hand side of productionF(E)|i

Left Recursion Not Present

**EXP7:**

**Compiler-Parser: Top Down Parser. Write a Program to Implement FIRST () for a given Grammar**

**Code:**

NT = []

T = []

temp = []

P = {}

first1 = {}

t1 = []

# number of elements as input

n\_NT = int(input("Enter number of Non-terminals : "))

n\_T = int(input("Enter number of Terminals : "))

#Enter Non-terminal

print("Enter List of Non-terminals")

for i in range(0, n\_NT):

item = input()

NT.append(item)

#Enter Terminals

print("Enter List of Terminals")

print("Enter 9 for epsilon")

Jay Vora TE3 - 77

for i in range(0, n\_T):

item = input()

T.append(item)

#Enter Productions

print("Enter Production")

for i in range(0,n\_NT):

print("Enter production for" + NT[i])

ele = input()

P[NT[i]]=ele

print(NT)

print(T)

print(P)

n\_P = len(P)

#Check for Computing FIRST()

for i in range(n\_NT):

nonter=NT[i]

pro = (P.get(nonter))

#check whether Non-terminal has multiple productions

if '|' in pro:

str1 = pro.split('|')

l = len(str1)

for i in range(l):

t = str1[i]

#Check if first character is Terminal

if(t[0] in T):

temp.append(t[0])

#Check if first character is Non-terminal

elif(t[0] in NT):

print("NT")

s = t[0]

s1 = first1.get(s)

temp.append(s1)

else:

print("Not")

#if there is only one production no need to split

else:

if(pro[0] in T):

temp.append(pro[0])

elif(pro[0] in NT):

print("NT")

sNT = pro[0]

sNT1 = first1.get(sNT)

temp.append(sNT1)

else:

print("Not")

first1[nonter] = temp

Jay Vora TE3 - 77

temp =[]

#Non-terminal whose first is remaining i.e First=None

for i in range(len(first1)):

check = first1.get(NT[i])

if (None in check):

t1.append(NT[i])

for k in range(len(t1)):

p1 = P.get(t1[k])

check = p1[0]

if(check in t1):

pro1 = P.get(check)

pro1\_ch = pro1[0]

value = first1.get(pro1\_ch)

first1[t1[k]] = value

else:

value = first1.get(check)

first1[t1[k]]=value

print("FIRST")

print(first1)

**Output:**

Enter number of Non-terminals : 5

Enter number of Terminals : 6

Enter List of Non-terminals

S

X

T

Y

F

Enter List of Terminals

Enter 9 for epsilon

+

\*

(

)

i

9

Enter Production

Enter production forS

TX

Enter production forX

+TX|9

Enter production forT

FY

Enter production forY

\*FY|9

Enter production forF

(S)|i

['S', 'X', 'T', 'Y', 'F']

Jay Vora TE3 - 77

['+', '\*', '(', ')', 'i', '9']

{'S': 'TX', 'X': '+TX|9', 'T': 'FY', 'Y': '\*FY|9', 'F': '(S)|i'}

NT

NT

FIRST

{'S': ['(', 'i'], 'X': ['+', '9'], 'T': ['(', 'i'], 'Y': ['\*', '9'], 'F': ['(', 'i']}'''

**EXP8:**

**Compiler-Parser: Top Down Parser. Write a Program to Implement FOLLOW () for a given Grammar**

**Code:**

NT = []

T = []

temp = []

temp\_follow = []

P = {}

first1 = {}

follow = {}

t1 = []

start='S'

# number of elemetns as input

n\_NT = int(input("Enter number of Non-terminals : "))

n\_T = int(input("Enter number of Terminals : "))

#Enter Non-terminal

print("Enter List of Non-terminals")

for i in range(0, n\_NT):

item = input()

NT.append(item)

#Enter Terminals

print("Enter List of Terminals")

print("Enter 9 for epsilon")

for i in range(0, n\_T):

item = input()

T.append(item)

#Enter Productions

print("Enter Production")

for i in range(0,n\_NT):

print("Enter production for" + NT[i])

ele = input()

P[NT[i]]=ele

print(NT)

print(T)

print(P)

n\_P = len(P)

#Check for Computing FIRST()

for i in range(n\_NT):

nonter=NT[i]

Jay Vora TE3 - 77

pro = (P.get(nonter))

#check whether Non-terminal has multiple productions

if '|' in pro:

str1 = pro.split('|')

l = len(str1)

for i in range(l):

t = str1[i]

#Check if first character is Terminal

if(t[0] in T):

temp.append(t[0])

#Check if first character is Non-terminal

elif(t[0] in NT):

print("NT")

s = t[0]

s1 = first1.get(s)

temp.append(s1)

else:

print("Not")

#if there is only one production no need to split

else:

if(pro[0] in T):

temp.append(pro[0])

elif(pro[0] in NT):

print("NT")

sNT = pro[0]

sNT1 = first1.get(sNT)

temp.append(sNT1)

else:

print("Not")

first1[nonter] = temp

temp =[]

#Non-terminal whose first is remaining

for i in range(len(first1)):

check = first1.get(NT[i])

if (None in check):

t1.append(NT[i])

for k in range(len(t1)):

p1 = P.get(t1[k])

check = p1[0]

if(check in t1):

pro1 = P.get(check)

pro1\_ch = pro1[0]

value = first1.get(pro1\_ch)

first1[t1[k]] = value

else:

value = first1.get(check)

first1[t1[k]]=value

print(first1)

Jay Vora TE3 - 77

#Computing follow function

for i in range(n\_NT):

s = NT[i]

print(s)

#Follow of starting symbol is '$'

if(s == start):

temp\_follow.append('$')

follow[s] = temp\_follow

#checking RHS whether the NT is there or not

for j in range(n\_NT):

pro = (P.get(NT[j]))

if(s in pro):

pos = pro.find(s)

next1 = pos+1

if(next1 == (len(pro))):

print("its last")

else:

next\_ch = pro[next1]

#if for A -> αBβ, β is terminal add it in follow of B

if(next\_ch in T):

temp\_follow.append(next\_ch)

follow[s] = temp\_follow

#if for A -> αBβ, β is non-terminal find If ∈∈First(β), then Follow(B) = { First(β) – ∈ } ∪ Follow(A)

elif(next\_ch in NT):

check = first1.get(next\_ch)

if('9' in check):

check.remove('9')

add\_lhs = follow[NT[j]]

check.extend(add\_lhs)

follow[s] = check

#if for A -> αB,FOLLOW(B) = FOLLOW(A)

else:

add\_lhs = follow.get(NT[j])

follow[s] = add\_lhs

next\_ch=""

print(follow)

**Output:**

Enter number of Non-terminals : 5

Enter number of Terminals : 6

Enter List of Non-terminals

S

X

T

Y

Jay Vora TE3 - 77

F

Enter List of Terminals

Enter 9 for epsilon

+

\*

(

)

i

9

Enter Production

Enter production forS

TX

Enter production forX

+TX|9

Enter production forT

FY

Enter production forY

\*FY|9

Enter production forF

(S)|i

['S', 'X', 'T', 'Y', 'F']

['+', '\*', '(', ')', 'i', '9']

{'S': 'TX', 'X': '+TX|9', 'T': 'FY', 'Y': '\*FY|9', 'F': '(S)|i'}

NT

NT

{'S': ['(', 'i'], 'X': ['+', '9'], 'T': ['(', 'i'], 'Y': ['\*', '9'], 'F': ['(', 'i']}

S

X

its last

T

Y

its last

F

{'S': ['$', ')'], 'X': ['$', ')'], 'T': ['+', '$', ')'], 'Y': ['+', '$', ')'], 'F': ['\*', '+', '$', ')']}

**EXP9:**

**Compiler: Code Optimization. Write a Program to Implement Code Optimization Technique: Compile Time Evaluation and Dead Code Elimination**

**Code: Compile Time Evaluation**

#Program for code optimization: Compile Time Evaluation(Constant Propagation)

#If some variable has been assigned some constant value, then it replaces that variable with its

constant value in the further program during compilation.

#pi = 3.14, radius = 10 , Area of circle = pi x radius x radius

#Directly put 3.14 in area instead of assigning it to variable pi

ans = 'y'

line = []

temp = []

def isNumber(s) :

for i in range(len(s)) :

if s[i].isdigit() != True :

return False

Jay Vora TE3 - 77

return True

def isFloat(s):

try :

float(s)

return True

except :

return False

while(ans == 'y'):

t1 = input("Enter the instruction")

line.append(t1)

ans = input("Do you want to continue")

print(line)

l = len(line)

for i in range(l):

if '=' in line[i]:

str1 = line[i].split('=')

if isNumber(str1[1])or isFloat(str1[1]):

no = str1[1]

for k in range(l):

if '=' in line[k]:

s1 = line[k].split('=')

if str1[0] in s1[1]:

subs = [str1[0], no]

line[k]=line[k].replace(subs[0], subs[1])

temp.append(line[i])

for j in range(len(temp)):

t1 = temp[j]

line.remove(t1)

print(line)

**Output:**

Enter the instructionp=3.14

Do you want to continuey

Enter the instructiona=p\*r\*r

Do you want to continuen

['p=3.14', 'a=p\*r\*r']

['a=3.14\*r\*r']

--------------------------------------------------------------------------------------------------------------------------------------

**Code: Dead code Elimination**

#Program for code optimization: Dead Code Elimination

#The statements of the code which either never executes or are unreachable or their output is never used are eliminated.

program = ['a=b\*c', 'x=a', 'd=b\*c+15']

Jay Vora TE3 - 77

op = ['+', '\*', '-', '/']

final = []

flag = 'False'

for i in range(len(program)):

temp = program[i]

if('=' in temp):

str1 = temp.split('=')

check = str1[1]

#check whether any operation is performed, i.e. the instruction gets executed

#add it in new list

res = any(ele in check for ele in op)

flag = str(res)

if(flag == 'True'):

final.append(temp)

print("Program before Dead Code Elimination")

print(program)

print("Program After Dead Code Eminination")

print(final)

**Output:**

Program before Dead Code Elimination

['a=b\*c', 'x=a', 'd=b\*c+15']

Program After Dead Code Eminination

['a=b\*c', 'd=b\*c+15']